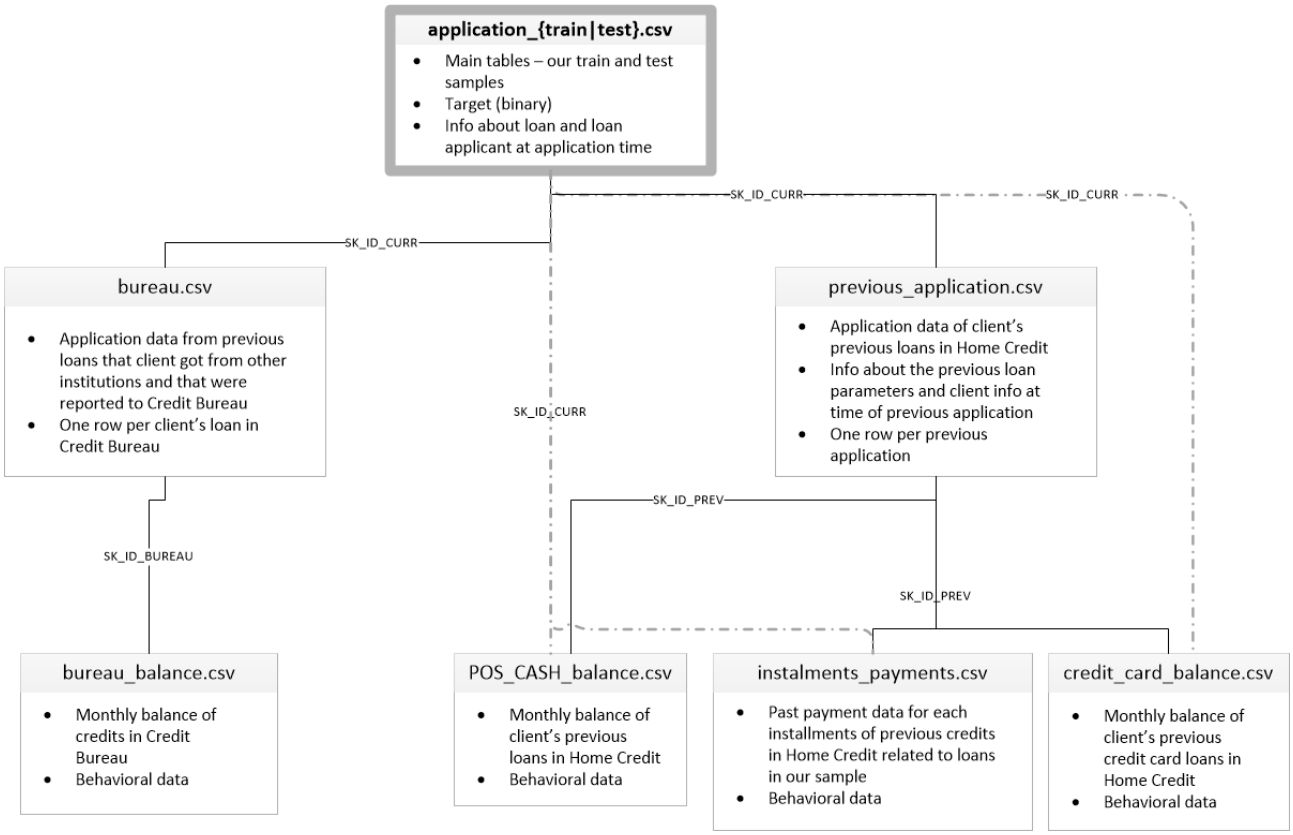
Note Méthodologique

I- Introduction

L'entreprise souhaite **développer un modèle de scoring de la probabilité de défaut de paiement du client** pour étayer la décision d'accorder ou non un prêt à un client potentiel en s'appuyant sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).

L'entreprise met à disposition 8 fichiers au format CSV. Nous avons un fichier « train » contenant une variable « cible » utiliser pour entraîner le modèle. Nous avons également un fichier « test » qui sera utilisé pour faire des prédictions. J'utiliserai donc le fichier « train » pour l'entrainement du modèle et « test » pour simuler la base de données des clients réels.

Architecture des fichiers mis à disposition :



Je repars d'un kernel pour pouvoir directement commencer la phase de modeling, l'analyse exploratoire, le nettoyage des données et le feature engineering est déjà réaliser dans le kernel. Ci-dessous le kernel duquel je suis parti :

https://www.kaggle.com/willkoehrsen/introduction-to-manual-feature-engineering

II- Méthodologie d'entraînement du modèle

1) Dummy classifier

Nous faisons face à jeu d'entrainement non équilibré, c'est à dire que 92 % des données ont pour cible 0 et 8 % ont pour cible 1. Cela pose des problèmes au niveau de l'entrainement du modèle. Je m'explique, en utilisant un simple « dummy classifier » avec une stratégie où le 0 est toujours prédit en sortie, on pourrait s'attendre à une précision d'environ 50 % si le jeu de données était équilibré. Or on obtient ceci :

	precision	recall	f1-score	support
0	0.92	1.00	0.96	84806
1	0.00	0.00	0.00	7448
accuracy			0.92	92254
macro avg	0.46	0.50	0.48	92254
weighted avg	0.85	0.92	0.88	92254

On observe une précision de 92 % ce qui pourrait signifier que l'on a un très bon modèle, or lorsque l'on regarde avec intérêt, on observe que l'on a un recall et une précision de 0 % pour les 1, cela signifie que l'on n'a jamais réussi à trouver les clients en défaut.

Un modèle de ce type signifie qu'on accorderait un crédit à tous les mauvais payeurs.

L'exemple ci-dessus me servira de baseline pour la suite du projet.

L'approche que j'ai choisie pour rééquilibrer les deux classes a été d'utiliser le paramètre suivant : « *class_weight »*. En effet, son mode « balanced » utilise les valeurs de y pour ajuster automatiquement les poids inversement proportionnels aux fréquences de classe dans les données d'entrée.

2) Gradient boost

J'utilise un modèle de gradient boosting pour entrainer mes données. Je laisse présent le déséquilibre des classes et j'observe les résultats suivants :

	precision	recall	f1-score	support
0	0.92	1.00	0.96	84806
1	0.51	0.04	0.08	7448
accuracy			0.92	92254
macro avg	0.72	0.52	0.52	92254
weighted avg	0.89	0.92	0.89	92254

On observe une précision de 51 % pour la classe en défaut et un recall de 4 %. Et on a toujours une précision totale de 92 %.

Je vais maintenant utiliser le paramètre « *class_weight »* avec le mode « balanced » pour tenter d'obtenir de meilleurs résultats.

3) Métrique métier

La question suivante se pose aussi :
Quelle classe faut-il détecter en priorité ? La classe des clients en défauts pour éviter au maximum les pertes d'argents ? Ou bien la classe des bons clients ? En supposant que l'écrasante majorité des bons clients comblera le trou des mauvais clients.

Je me heurte ici à une problématique métier à laquelle je ne peux répondre. Je décide donc de créer une métrique métier. On peut classer les prédictions en 4 catégories :

- False negatives : Mauvais clients que l'on n'a pas réussi à identifier ➔ Perte
- False positives : Bons clients que l'on n'a pas réussi à identifier ➔Manque à gagner
- True negatives : Mauvais clients que l'on a réussi à identifier ➔Argent sauvé
- True positives : Bons clients que l'on a réussi à identifier ➔Gain

A ces catégories, je vais attribuer des poids arbitraires qui vont indiquer leur importance :
fn_coeff = -10
fp_coeff = 0
tp_coeff = 0
tn_coeff = 1

Cela donne la fonction suivante : Gain totale = -10*FN + 0*FP + 0*TP + 1*TN
Gain totale = TN – 10 FN

Je détermine ensuite la fonction Gain_max et Gain_min qui me permet de normaliser la fonction Gain_totale entre 0 et 1

Après entrainement du modèle et test sur le jeu d'entrainement, j'ai de nouveaux résultats :

	precision	recall	f1-score	support
0	0.96	0.72	0.82	84806
1	0.17	0.67	0.28	7448
accuracy			0.72	92254
macro avg	0.57	0.69	0.55	92254
weighted avg	0.90	0.72	0.78	92254

Mais le plus important reste la métrique métier car c'est celle que j'ai utilisé lors de l'entrainement de mon modèle, plus ma métrique est proche de 1 plus on approche du Gain max

```
'credit_metric': 0.6958489760556483}
```

Ma métrique est à 70% de mes gains max ce qui représente un bon résultat. Avec les équipes métiers, on peut imaginer des seuils sous lesquels on ne peut descendre pour être rentable pour l'entreprise.

4) Random search

Je vais maintenant essayer d'optimiser les hyperparamètres de mon modèle, et pour ce faire, je vais utiliser la fonction randomizedsearchCV qui permet de chercher aléatoirement dans un dictionnaire d'hyperparamètres les meilleures combinaisons possibles. Je décide d'essayer 10 combinaisons.

Voici mon dictionnaire d'hyperparamètres :

```
param_test ={
            'num_leaves': [10,50,100,1000],
            'min_child_samples': [100,250,500],
            'min_child_weight': [1e-5, 1e-1, 1, 1e1, 1e3],
            'reg_alpha': [0, 1e-1, 1, 5, 10, 50, 100],
            'reg_lambda': [0, 1e-1, 1, 5, 10, 50, 100]
            }
```

La meilleure combinaison d'hyperparamètres trouvée est la suivante :

```
{'min_child_samples': 100,
 'min_child_weight': 1,
 'num_leaves': 50,
 'reg_alpha': 5,
 'reg_lambda': 0.1}
```

Après utilisation de ces nouveaux hyperparamètres dans mon modèle principal, les résultats sont les suivants :

	precision	recall	f1-score	support
0	0.96	0.73	0.83	84806
1	0.18	0.66	0.28	7448
accuracy			0.72	92254
macro avg	0.57	0.69	0.55	92254
weighted avg	0.90	0.72	0.78	92254

On n'observe qu'une très légère amélioration en comparaison avec le modèle sans optimisation des hyperparamètres.

III- Interprétation du modèle

Le modèle étant destiné à des équipes opérationnelles devant être en mesure d'expliquer les décisions de l'algorithme à des clients réels, le modèle est accompagné d'un module d'explicabilité.

La solution retenue a été d'utiliser LIME qui est une librairie permettant de déterminer pour chaque prédiction, les features qui ont le plus d'impact lors de la prédiction du résultat.

IV- Limites et améliorations

Les améliorations possibles pour l'entrainement du modèle seraient :

- Utiliser une recherche sur grille pour déterminer les meilleurs hyperparamètres mais cela prendrait beaucoup plus de temps que la recherche aléatoire.
- Améliorer le feature engineering
- Utiliser les autres fichiers mis à disposition
- Améliorer la métrique personnalisée avec les équipes métiers
- Obtenir beaucoup plus de données de la classe en défaut
- Utiliser un serveur avec des ressources plus puissantes