**NOTE METHODOLOGIQUE**

L'objectif de notre projet est de prédire, dans un premier temps, le score d'un client donné et à partir de ce score classé le client comme client défaillant ou honnête dans le remboursement d'un prêt. Pour ce faire, nous allons passer par plusieurs étapes :

-Nettoyage des données : traitement préliminaire des données car les données sont reparties dans plusieurs dataframe, identification et traitements des NaN, évaluation du balancement des données,

-Définition du score métier grace à une fonction que nous allons créer

-Mise en place de MLFLOW afin de stocker et de comparer les paramètres et les métriques de nos modèles

-Entrainement et comparaison de plusieurs modèles pour ressortir le meilleur pour notre étude

-Feature importance globale des différentes variables et feature importance locale.

La présente note méthodologique donnera des éclaircissements sur :

* + La méthodologie d'entraînement du modèle
  + Le traitement du déséquilibre des classes
  + La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation
  + Un tableau de synthèse des résultats
  + L’interprétabilité globale et locale du modèle
  + Les limites et les améliorations possibles
  + L’analyse du Data Drift

**Méthodologie d’entrainement du modèle :**

Tout d’abord, nous allons importer les dataframes dont nous aurons besoin pour notre étude. Les données seront ensuite traitées : les valeurs manquantes, les outliers ... Les variables catégorielles seront aussi encodées en binaire pour êtres comprises par nos modèles.

Ensuite, il y aura création de certaines fonctions pour le traitement des thèmes spécifiques relatifs à des dataframes spécifiques pour les variables (somme, moyenne,)pour faciliter l’obtention d’un dataframe unique regroupant toutes les informations utiles que l’on va utiliser pour la modélisation.

Comme nos individus sont les clients d’une banque, il est essentiel de vérifier s’il n’y a pas de doublons ni de valeurs manquantes au niveau de leur identifiant unique SK\_ID\_CURR. Il est également important de voir la distribution de la variable TARGGET pour mettre en évidence un éventuel déséquilibre des classes. Nous constatons effectivement un déséquilibre entre la TARGET :0(client ayant remboursé leur prêt) avec un pourcentage de 91,93% et la TARGET : 1( client défaillant) avec un pourcentage de 8,07%. Une dernière vérification et traitement des NaN s’impose également.

Une fois cela fait, nous pouvons commencer la modélisation. Pour notre cas, nous choisirons de tester 3 types de modèles (et comme modèle de base on choisira le DummyClassifier) :

* Catboost classifier
* RandomForest Classifier
* XGBoost Classifier

Pour chacun de ces modèles, nous avons d’abord spécifié les variables d’entrainement et de test(à raisons de 20% de test et 80% d’entrainement). Puis, pour standardiser nos variables nous avons utilisé le StandardScaler. Ensuite, comme nous allons faire un GridSearchCv pour optimiser nos mos hyperparamètres, il nous faut spécifier les valeurs d’itération des différents hyperparamètres. Ensuite, nous allons entrainer le modèle.

Après cela, nous ferons un prédict-proba sur X-test et sur X\_train pour calculer l’AUC pour le test et pour le train également afin de vérifier qu’il n’y a pas de dataleakage. En effet, il ne doit pas trop avoir de différence entre les valeurs de AUC pour le train et pour le test (cette valeur doit-être comprise entre 0 et 1 mais pour notre cas le meilleur résultat obtenu est de 0,82 donc un AUC de 0,65 à 0, 8 serait déja bon). Combiné à l’AUC, nous avons défini une métrique score obtenu avec cette formule :

*Score = false\_positive\_cost \* false\_positives + false\_negative\_cost \* false\_negatives*

Avec comme hypothèse :

*false\_positive\_cost = 1 # Coût d'un faux positif*

*false\_negative\_cost = 10 # Coût d'un faux négatif*

En d’autres termes, additionné avec l’AUC, nous allons également voir le nombre de faux positifs dans chaque modèle pour pouvoir trancher (obtenu avec la matrice de confusion).

Et comme le seuil est de 0,5 par défaut pour la probabilité de classe positive, nous allons également voir pour chaque modèle quel est le meilleur seuil pour la probabilité de classe positive. Pour ce faire, nous allons créer une fonction , qui utilisera le score et une itération du seuil de 0,1 à 0,9 avec un pas de 0,1, pour stocker le meilleur score rencontré au fur et à mesure que le code parcourt différents seuils (thresholds) et évalue les performances du modèle pour chaque seuil.

A la fin, nous avons créer une fonction permettant directement de faire le tracking de donnés et l’enregistrement des modèles dans MLFlow. :

Une image contenant capture d’écran, texte, nombre

Description générée automatiquement

**Le traitement du déséquilibre des classes :**

Comme évoqué plus haut, il existe un déséquilibre entre la TARGET :0(client ayant remboursé leur prêt) avec un pourcentage de 91,93% et la TARGET : 1(client défaillant) avec un pourcentage de 8,07%. Dans ce cas, il est préférable de traiter ce déséquilibre car vu son importance, les modèles auront toujours tendance à prédire la classe majoritaire, ce qui peut fausser nos résultats.

Plusieurs moyens existent pour corriger ce déséquilibre comme :

* **sur-échantillonnage (oversampling)** : Augmenter le nombre d'échantillons de la classe minoritaire.
* **Sous-échantillonnage (undersampling)** : Réduire le nombre d'échantillons de la classe majoritaire.
* **Génération synthétique de données** : Créer de nouveaux exemples pour la classe minoritaire à l'aide de techniques comme SMOTE (Synthetic Minority Over-sampling Technique).
* **Pondération des classes** : Donner un poids différent aux classes lors de l'apprentissage du modèle pour compenser le déséquilibre.

Pour notre cas nous allons utiliser la méthode SMOTE (Synthetic Minority Over-sampling Technique) pour créer de nouveaux exemples pour la classe minoritaire par des données synthétiques. De ce fait, la classe minoritaire sera agrandie jusqu'à atteindre 75% de la taille de la classe majoritaire.

La méthode SMOTE présente des avantages comme améliorer la performance du modèle ou encore éviter l’overfitting. Mais il présente également des inconvénients notamment sur des modèles sensibles aux bruits du fait de la surgénération de données.

**La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation :**

La fonction cout métier :

En banque, dans l'octroi des prêts, un faux négatif (FN - mauvais client prédit bon client : donc crédit accordé et perte en capital) cause 10 fois plus de perte que le faux positif (FP - bon client prédit mauvais : donc refus crédit et manque à gagner en marge)

Pour cela, nous allons créer un score pour chaque client en tenant en compte ces valeurs false\_positive\_cost = 1(Coût d'un faux positif) et false\_negative\_cost = 10 (Coût d'un faux négatif) et aussi une fonction permettant d'évaluer pour chaque modèle le pourcentage des clients dans chaque type Faux Négatifs, Faux Positifs, True Négatifs, True Positifs.

Algorithme d’optimisation et métrique d’évaluation : Pour trancher sur le meilleur modèle, à part la valeur de l’AUC, nous allons également tenir compte du nombre de faux positifs et du seuil de prédiction. C’est pour cela qu’on a effectué une itération du seuil de prédiction pour voir le score correspondant et l’optimiser pour avoir le meilleur seuil.

Finalement, pour bien illustrer ceci, nous allons un exemple de matrice de confusion

**Tableau de synthèse des résultats**

Le tableau suivant regroupe les métriques et les commentaires qui nous ont amené à choisir le meilleur modèle en occurrence XGBoost.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modèle(Classifier) | AUC | score | Nbre FN | Meilleur seuil | Commentaires |
| Dummy | 0.49 |  |  |  | Modèle de base |
| Catboost | 0.74 | -9953 | 9886 | 0.80 | Données deséquilibrées |
| XGBoost | 0.76 | -9952 | 9809 | 0.7 | Données deséquilibrées |
| RandomForest | 0.67 | -9953 | 9953 | 0.30 | Présence de O dans FP et TN |
| CatBoost Smote | 0.65 | -9763 | 7843 | 0.80 | Pourcentage de TN |
| XGBoost Smote | 0.64 | -9763 | 8889 | 0.6 | Peut aller à 0,8 |
| RandomForest Smote | 0.62 | -9763 | 8339 | 0.70 | AUC faible |

**Interprétabilité globale et locale du modèle**

La feature importance globale permet de voir le degré d’importance des variables pour le modèle choisi. Pour notre cas, les variables les plus significatives pour le modèle sont :

**Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement**

Quant à la feature importance locale, il s’agit de voir les variables qui ont le plus d’influence dans la prédiction du score pour un individu donné. Par exemple, pour le client N° 100002, on a :

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

**Limites et les améliorations possibles**

Notre modèle n’est pas parfait, mais il réussit quand même à bien prédire le score. Toutefois, des améliorations sont encore possibles par exemple :

* Dans le prétraitement des données, vu que ‘ai utilisé ici le prétraitement déjà sur kaggle, peut-être qu’i y a des axes d’améliorations aussi à faire dans ce sens .
* Dans le choix des différents modèles : il y a peut-être d’autres modèles mieux adaptés à ce problème précis que l’on n’a peut-être pas testé
* Dans choix également des hyperparamètres : avoir plus de diversité d’hyperparamètres à tester pour améliorer le pouvoir prédictif du modèle

**Analyse du Data Drift**

L’analyse du datadrift est effectuée pour détecter des changements significatifs des données qui se produisent entre un ensemble de données actuelles et un ensemble de données de référence pour maintenir la qualité prédictive du modèle. Pour notre cas, nous utiliserons les données dans ‘application\_train ‘ comme données de références et ‘application\_test’ comme données actuelles pour faire l’analyse. On constate qu’il n’y a pas de datadrift dans nos données car le seuiest de 50%, alors il n’y a que 20 colonnes où du datadrift a été détecté. Il n’y a pas de décalage entre les données d’entraînement d’un modèle et les données du monde réel.

Une image contenant texte, reçu, capture d’écran, Police

Description générée automatiquement