NOTE MÉTHODOLOGIQUE

La méthodologie d'entraînement des modèles

L’entreprise "Prêt à dépenser" 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.).

Les données utilisées pour réaliser ce projet sont issues d'une compétition Kaggle: "**Home Credit Default Risk** "

La démarche

1. Analyse exploratoire et **feature engineering**: choix d’un Kernel
2. La fonction de **création du coût métier**
3. Le traitement du **déséquilibre des classes**
4. Développement et simulation de modèles

Commençons par le choix d'un kernel. Il s'agit d'en trouver un l suffisamment intéressant d’un point de vue technique et de performance.

Pour l'analyse exploratoire, et le Feature engineering, nous avons choisi les kernels: "LightGBM with Simple Features". Ils ont permis à leurs auteurs d'obtenir une performance parmi les meilleures, et ils metttent en œuvre des techniques d’agrégation très intéressantes qui permettent un code très concis.

Création d’un score métier

Pour élaborer notre modèle, nous devons prendre en compte Le déséquilibre du coût métier entre un faux négatif (FN - mauvais client prédit bon client : donc crédit accordé et perte en capital) et un faux positif (FP - bon client prédit mauvais : donc refus crédit et manque à gagner en marge).Un faux négatif est environ 10 fois plus coûteux qu’un faux positif.

Le score “métier” consiste à calculer une fonction de coût métier de type 10*FN + FP (où FN = nombre de FN dans la matrice de confusion pour un seuil donné, FP = nombre de FP) et de trouver son minimum pour un seuil donné.

Nous allons créer le score via “make_scorer” qui calcule le coût des erreurs de prédiction (donc les FN et FP). Les modèles et leur hyperparamètres seront optimisés via un GridSearchCV ou équivalent sur ce score.

```
[41]: #Création score métier
      from sklearn.metrics import classification_report, confusion_matrix, roc
      from sklearn.metrics import fbeta_score, make_scorer, auc,roc_auc_score

      def custom_metric(y_true, y_pred):
          """Fonction qui extraie de la matrice de confusion les
          valeurs des 4 mesures et retourne un calcul sur base
          d'une pondération de FP et FN"""
          tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel().tolist()
          score=(10*fn + fp)/len(y_true)
          return score
      score_metier=make_scorer(custom_metric, greater_is_better=False)

      scoring = {"AUC": "roc_auc", "score_metier": score_metier}
```

Le traitement du déséquilibre des classes

L'analyse exploratoire nous permet de remarquer le déséquilibre entre le nombre de bons et de moins bons clients. En effet, nos données contiennent 92% de bons clients. Pour élaborer un modèle performant, nous devons prendre en compte ce déséquilibre dans notre modélisation en appliquant un traitement de rééquilibrage.

Nous choisissons la méthode par oversampling, via la librairie Smote, pour générer de nouveaux enregistrements de classe 1. Pour éviter le data leakage lors de la cross validation, l’oversampling se fera pour chaque jeu de données entraîné via la pipeline imblearn.

Développement et simulation de modèles

Nous avons effectué des simulations complètes des algorithmes suivants:

- **DummyClassifier**: utilisé comme base de référence simple pour comparer avec d'autres classificateurs,

- **LogisiticRegression**: algorithme de classification binaire. Il utilise une fonction sigmoide pour prédire la probabilité d'appartenance à une classe.

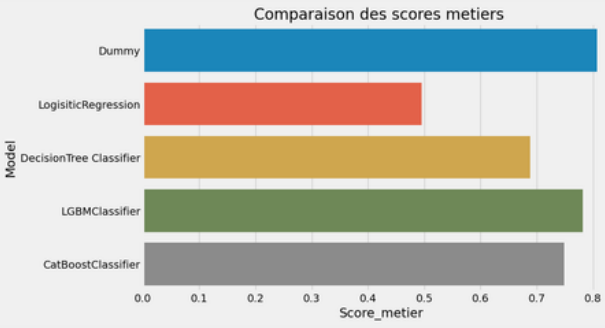
- **DecisionTree** Classifier: L'arbre prédit la variable cible à l'aide d'une succession de règles if-then-else (si/sinon).Lors de l'apprentissage, l'algorithme de l'arbre de décision cherche à diviser les données d'entraînement en sous-ensembles homogènes, en choisissant les caractéristiques et les seuils de division qui maximisent la séparation entre les classes. Cela permet de construire une séquence de décisions logiques pour classifier de nouvelles instances.

- **LGBMClassifier:** Il utilise des techniques de Gradient Boost pour construire de manière itérative un ensemble d'arbres de décision et effectuer des prédictions en fonction de la sortie combinée de ces arbres.

- **CatBoostClassifier:** construit un ensemble de modèles de prédiction, généralement des arbres de décision, de manière itérative. À chaque itération, le modèle tente de corriger les erreurs faites par les modèles précédents, en mettant davantage l'accent sur les échantillons mal classés.

Le modèle retenu

Le modèle retenu est le **LogisiticRegression.** Le sore métier est à son minimum avec ce modèle

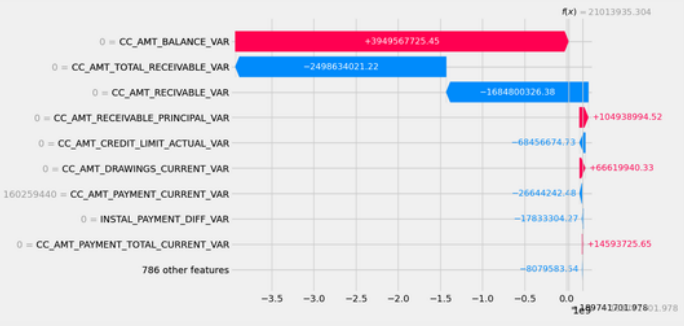


L’interprétabilité globale et locale du modèle

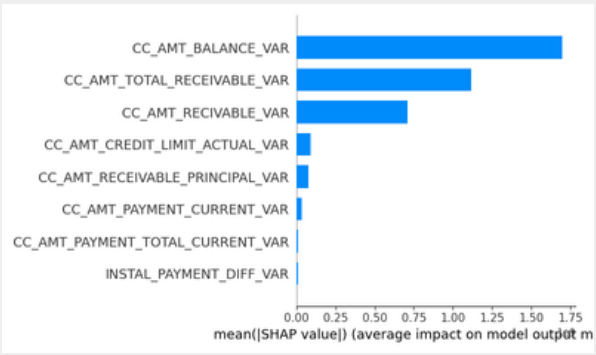
L’objectif métier est de donner des informations pertinentes d’analyse au chargé d’étude, afin qu’il comprenne pourquoi un client donné est considéré comme bon ou mauvais prospect et quelles sont ses données qui expliquent son évaluation. Il est donc nécessaire à la fois, de connaître d’une manière générale les principales features qui contribuent à l’élaboration du modèle, et de manière spécifique pour le client qu’elle est l’influence de chaque feature dans le calcul de son propre score (feature importance locale)

Pour cette partie, nous allons utilisé le module SHapley Additive exPlanations (SHAP). Les valeurs SHAP représentent la contribution de chaque feature en comparant les prédictions réalisées avec et sans la présence.

Feature importance globale

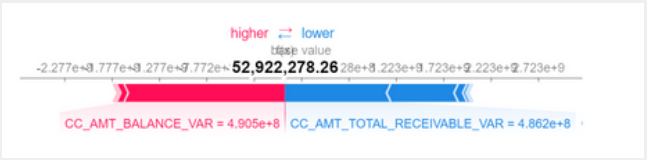


Feature importance locale



SHAP force plot:

La valeur SHAP pour chaque feature est représentée par une barre. La longueur et la couleur de la barre indiquent l'effet de la feature sur la prédiction. Une barre plus longue indique une plus grande influence, tandis qu'une barre rouge indique une influence positive, et une barre bleue, une influence négative sur la prédiction.



Lexique

Out[73]:

	Row	Description
153	AMT_BALANCE	Balance during the month of previous credit
154	AMT_CREDIT_LIMIT_ACTUAL	Credit card limit during the month of the previous credit
156	AMT_DRAWINGS_CURRENT	Amount drawing during the month of the previous credit
160	AMT_PAYMENT_CURRENT	How much did the client pay during the month on the previous credit
161	AMT_PAYMENT_TOTAL_CURRENT	How much did the client pay during the month in total on the previous credit
162	AMT_RECEIVABLE_PRINCIPAL	Amount receivable for principal on the previous credit
164	AMT_TOTAL_RECEIVABLE	Total amount receivable on the previous credit

L’analyse du Data Drift

Le Data Drift ou dérive de donnée survient quand on a une grande différence entre les données d'entrainement et les données d'exécution. Ce problème doit être détecté et anticipé, car il dégrade les performances de prédiction au fur et à mesure du temps. Si un Data Drift n’est pas identifié à temps, les prédictions du modèle seront erronées. Les décisions prises à partir de ces prédictions auront donc un impact négatif.

Nous avons utilisé la librairie Evidently pour détecter éventuellement du Data Drift sur les principales features, entre les datas d’entraînement et les datas de production.

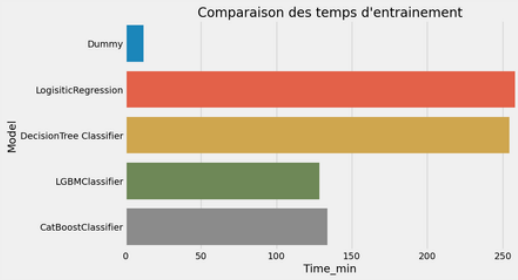
```
[9]: report = Report(metrics=[
         TargetDriftPreset()
     ])

     report.run(reference_data=train_df, current_data=test_df,#column_mapping
                )
     report.save_html('evidently_metrics_report.html')
```

Drift in column 'prediction'

Data drift not detected. Drift detection method: Jensen-Shannon distance. Drift score: 0.036

Les limites et les améliorations possibles

- Le **temps d'entrainement** des différents algorithmes est relativement **long**.



- Le **nombre élevé de features** pour l'entrainement du modèle. Le kernel LightGBM with Simple Features utilisé pour le feature engineering a permis à son auteur d'avoir une performance parmi les meilleures mais contient au total 795 features. Nous aurions pu sélectionner les feaures les plus importants pour faciliter l'analyse.

- Le meilleur modèle a été sélectionné en fonction du minimum du "score métier". En améliorant les paramètres de la construction de la "fonction coût métier " et en les adaptant aux exigences du métier, nous pouvons amélioré nos résultats