Note méthodologique

La note technique est un des livrables du projet 7 « Implémentez un modèle de scoring » et permet de présenter toute la démarche d'élaboration du modèle.

Le projet consiste à mettre en œuvre un outil de « scoring crédit », pour l'entreprise *Prêt à dépenser*, pour calculer la probabilité qu'un client rembourse son crédit, puis classifie la demande en crédit accordé ou refusé.

Sommaire

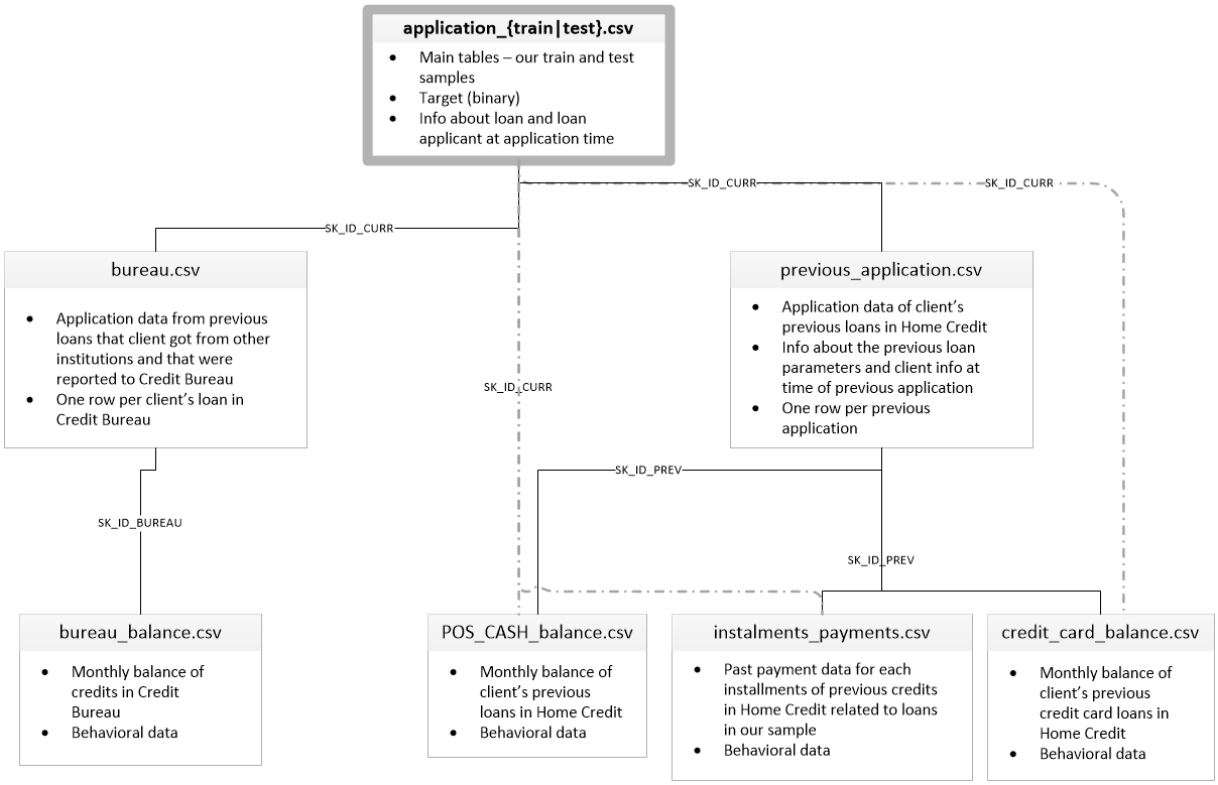
Références :

Consignes du projet : https://openclassrooms.com/fr/paths/164/projects/632/assignment
Base de données : https://www.kaggle.com/c/home-credit-default-risk/data
GitHub : https://github.com/OceaneYYT/P7-OCR
API : https://p7-ocr-fastapi-95768180a01f.herokuapp.com/
Dashboard : https://p7-ocr-dashboard-a790d1a0f622.herokuapp.com/

I – Méthodologie d'entraînement du modèle

1. Traitement des données

Les données mises à disposition sont issues d'une base de données Kaggle de la compétition *Home Credit Default Risk* disponible ici. Elle comporte 10 fichiers, soit 346 colonnes. Chacune des tables est reliée par une clé comme indiqué ci-dessous.



La table *HomeCredit_columns_description.csv* décrit les différentes variables et la table *sample_submission.csv*, quant à elle, est seulement composé des ID clients et des target.

Afin de faciliter l'analyse exploratoire, la préparation des données et le feature engineering nécessaires à l'élaboration du modèle de scoring, nous nous basons sur ce kernel Kaggle.

À la suite de ce traitement, nous avons 307 507 observations et 797 variables pour le dataset d'entrainement et 48 744 observations et 796 variables pour le dataset de test (qui ne comporte pas de colonne « *TARGET* »).
Nous traitons désormais les données manquantes qui sont présente à environ 25% dans nos dataset. Pour commencer, nous supprimons les variables ayant plus de 50% de données manquantes. Ensuite, nous traitons les valeurs aberrantes avant de procéder à l'imputation. Nous appliquons différentes méthodes d'imputation simples (0, médiane et mode).

Nous avons appliqué un preprocessing à nos données à savoir un one hot encoder pour les variables catégorielles et un Standard scaler pour les variables quantitatives.

Le dataset de test ne contient pas de colonne « TARGET » et sera utilisé lors du déploiement de notre modèle. Pour la modélisation, nous n'utilisons que le dataset d'entrainement qui

sera séparé en plusieurs parties afin de disposer d'un jeu d'entrainement (246 005 observations), d'un jeu de validation (49 201 observations) et d'un jeu de test (12 301 observations) pour entraîner et tester nos différents modèles.

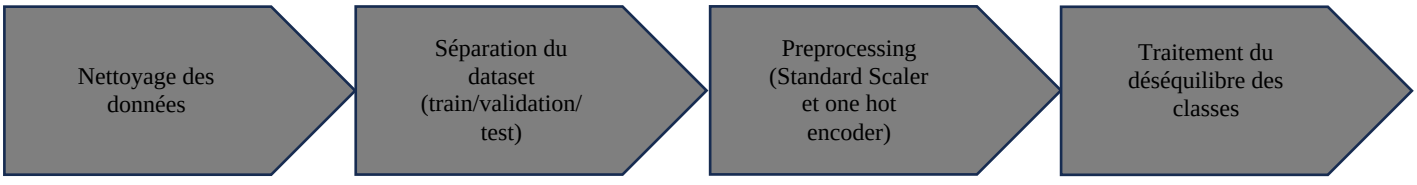
2. Application de différents modèles

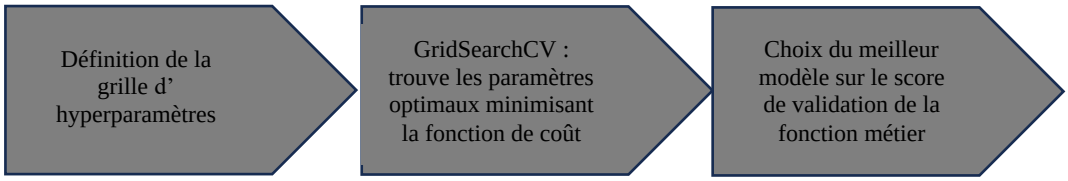
Nous avons sélectionné quelques algorithmes de classification que nous allons appliquer par la suite :

- **Dummy Classifier (baseline)** : ne prend pas en compte les caractéristiques du jeu de données et se contente de faire des prédictions en utilisant des règles simples. Ici, on renvoie l'étiquette de classe la plus fréquente dans l'argument y observé.
- **Logistic Regression** : le but est de trouver une relation mathématique entre les variables d'entrée (features) et la variable de sortie (classe prédite). Cette relation est généralement exprimée sous la forme d'une fonction logistique qui transforme la sortie en une probabilité.
- **SVC** : L'algorithme fonctionne en trouvant un hyperplan optimal qui sépare les données d'entraînement en différentes classes. L'hyperplan est déterminé de manière à maximiser la marge entre les points de données de chaque classe. Les points de données les plus proches de l'hyperplan sont appelés vecteurs de support.
- **Decision Tree** : algorithme d'apprentissage automatique qui prend un ensemble de données en entrée et construit un modèle prédictif sous forme d'arbre hiérarchique. Chaque nœud de l'arbre représente une caractéristique de l'ensemble de données, chaque branche représente une règle de décision basée sur cette caractéristique, et chaque feuille représente une classe ou une valeur prédite.
- **Random Forest** : algorithme d'apprentissage automatique qui combine plusieurs arbres de décision pour effectuer des prédictions. Chaque arbre de la forêt donne une prédiction et la classe prédite est déterminée par un vote majoritaire.
- **XG Boost** : utilise un ensemble de modèles d'arbres de décision pour effectuer des prédictions. L'algorithme fonctionne en itérations successives pour minimiser une fonction de perte spécifiée et ajouter des arbres qui réduisent cette perte.
- **Light GBM** : algorithme d'apprentissage automatique basé sur le gradient boosting qui est conçu pour offrir une exécution rapide et des performances élevées. Il utilise une technique d'échantillonnage basée sur le gradient pour sélectionner les échantillons les plus informatifs pendant le processus d'apprentissage.

Par la suite, chacun de ces algorithmes sera testé avec recherche d'hyperparamètres et cross validation (via une *GridSearchCV*).

Voici ci-dessous un schéma récapitulatif de la méthodologie appliquée.

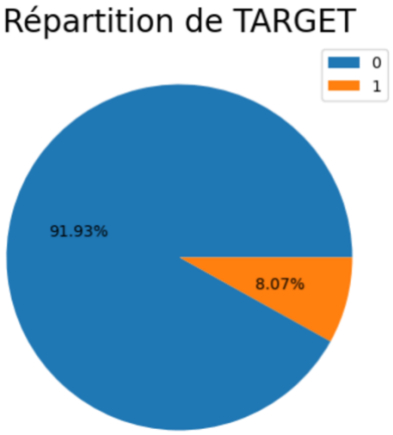




Un pipeline a également été créée afin de rassembler les étapes de traitement du déséquilibre des données, preprocessing et entraînement du modèle.

II – Traitement du déséquilibre des classes

Le problème est un problème de classification binaire avec une classe sous représentée (8.07 % de clients en défaut contre 91.93 % de clients sans défaut).



Ce déséquilibre des classes doit être pris en compte dans l'entraînement des modèles puisqu'un modèle « naif » prédisant systématiquement que les clients sont sans défaut aurait une accuracy de 0.92 et pourrait être considéré à tort comme un modèle performant alors qu'il ne permettrait pas de détecter les clients à risque.
Plusieurs approches pour rééquilibrer les deux classes ont été testées :

- **Aucun rééquilibrage** -> Score métier : 0.13 et Accuracy score : 0.92.
- **Class_weight** : attribue des poids différents aux différentes classes de notre dataset, en donnant un poids plus important aux classes minoritaires, afin d'influencer le modèle lors de son entraînement -> Score métier : 0.407 et Accuracy score : 0.70.
- **SMOTE** : suréchantillonnage des observations minoritaires. Génère de nouveaux individus minoritaires qui ressemblent aux autres, sans être strictement identiques. -> Score métier :0.13 et Accuracy score : 0.92.
- **Tomek link** : approche de sous-échantillonnage de la classe majoritaire. L'idée est de chercher les points de la classe majoritaire qui sont assez proches d'un point de la classe minoritaire et les supprimer. -> Score métier : 0.13 et Accuracy score : 0.92.
- **SMOTETomek** : technique combinant l'oversampling (SMOTE) et l'undersampling (Tomek Links) -> Score métier : 0.13 et Accuracy score : 0.92.
- **RandomUnderSampler** : Sous-échantillonner la classe majoritaire en choisissant des échantillons au hasard. -> Score métier : 0.404 et Accuracy score : 0.70.

On choisit la méthode d'undersampling avec *RandomUnderSampler* pour le rééquilibrage de nos données. En effet les méthodes *class_weight* et *RandomUnderSampler* offrent les meilleurs résultats de score métier. Cependant, l'argument *class_weight* n'est pas disponible pour tous les modèles. De plus, le temps d'exécution de *RandomUnderSampler* est bien inférieur à ceux des autres méthodes.

III - La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation

1. La fonction de coût métier

Une fonction de coût métier sur mesure a été créée pour la société Prêt à dépenser. Nous créons donc un nouveau score métier qui a pour objectif de pénaliser plus sévèrement les Faux Négatifs qui sont donc des clients ayant fait défaut mais qui ont été prédit comme ne faisant pas défaut. En effet, ce type d'erreur engendre une perte en capital pour l'entreprise.

Nous basons donc notre score sur la matrice de confusion.

		Classe réelle	
		-	+
Classe prédite	-	True Negatives *(vrais négatifs)*	False Negatives *(faux négatifs)*
	+	False Positives *(faux positifs)*	True Positives *(vrais positifs)*

L'objectif est donc ici de pénaliser plus fortement les Faux Négatifs. On attribut donc un coefficient à chaque catégorie de la matrice de confusion.

```
TP_coeff = 0       # Vrais positifs
FP_coeff = 0       # Faux positifs (prédit comme faisant défaut (1) mais ne fait pas défaut (0))
FN_coeff = -10     # Faux négatifs (prédit comme ne faisant pas défaut (0) mais font défaut (1))
TN_coeff = 1       # Vrais négatifs
```

Finalement, nous obtenons la métrique suivante :

$$gain = \frac{TP * TP_{coeff} + TN * TN_{coeff} + FP * FP_{coeff} + FN * FN_{coeff}}{(TN + FP + FN + TP)}$$

On cherchera à maximiser cette métrique.

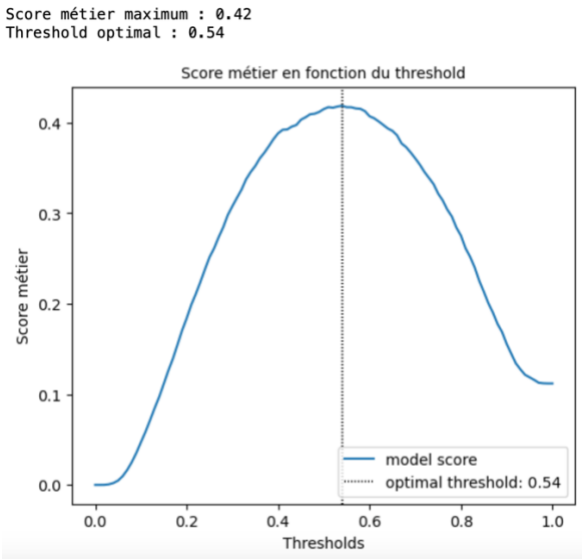
2. L'algorithme d'optimisation

L'algorithme jugé le plus performant est Light GBM. L'optimisation a consisté à tester différents hyperparamètres afin d'améliorer ses performances. Les hyperparamètres testés sont les suivants :

- n_estimators : nombre d'arbres à ajuster -> valeurs : [100, 300, 500, 800]
- max_depth : définit une limite sur la profondeur de l'arbre -> valeurs : [-1, 2, 5, 7]

- num_leaves : spécifie le nombre de feuilles dans un arbre -> valeurs : [7, 15, 31, 63, 127]
- learning_rate : ajuste le taux d'apprentissage du modèle -> valeurs:[0.05,0.1,0.2,0.4].

De plus, différents seuils de probabilité de défaut ont été testés afin de maximiser notre score métier.



Notre modèle le plus performant est donc Light GBM avec les paramètres *{'learning_rate': 0.05, 'max_depth': -1, 'n_estimators': 500, 'num_leaves': 31}* et un seuil de 0.54.

3. La métrique d'évaluation

La métrique d'évaluation principale est donc notre score métier, que nous avons cherché à maximiser en testant différents algorithmes et combinaisons d'hyperparamètres.
D'autres métriques ont également été calculées et observées telles que :

- **Accuracy** : somme de tous les vrais positifs et vrais négatifs qu'il divise par le nombre total d'instances. Des valeurs élevées de ce paramètre sont souvent souhaitables.
- **Precision** : indique le rapport entre les prévisions positives correctes et le nombre total de prévisions positives.
- **Recall/Rappel** : paramètre qui permet de mesurer le nombre de prévisions positives correctes sur le nombre total de données positives.
- **F1 score** : moyenne harmonique de la précision et du rappel. Sa valeur est maximale lorsque le rappel et la précision sont équivalents.
- **Fbeta score** : généralisation de la F-measure qui ajoute un paramètre de configuration appelé beta. Une valeur bêta < 1, donne plus de poids à la précision et moins au rappel, tandis qu'une valeur bêta > 1 donne moins de poids à la précision et plus de poids au rappel. Ici nous donnons plus de poids au rappel qui minimise les faux négatifs.

- **ROC AUC score** : mesure de façon globale la performance d'un modèle de classification. Il indique à quel point le modèle est capable de faire la distinction entre les classes.

IV – Synthèse des résultats

Un tracking a été effectué pour les différents entraînements de modèles via MLflow.
Tout d'abord, le tracking des différents algorithmes (après optimisation des hyperparamètres) nous indique que Light GBM est le plus performant par rapport au score métier notamment, qui est notre métrique d'évaluation.

Run Name:	Logistic regression optimise	Random forest optimise	XGBoost optimise	Light GBM optimise	Decision Tree optimise	SVC optimise
Start Time:	2023-05-31 17:33:51	2023-05-31 19:05:11	2023-06-02 20:36:28	2023-06-05 20:30:04	2023-07-20 19:17:15	2023-07-24 13:
End Time:	2023-05-31 17:33:52	2023-05-31 19:05:25	2023-06-02 20:36:29	2023-06-05 20:30:05	2023-07-20 19:17:16	2023-07-24 13:
Duration:	0.5s	13.9s	0.5s	0.6s	0.9s	3.6min

› Parameters

⌄ Metrics

Show diff only

val_accuracy	0.7	0.696	0.709	0.708	0.654	0.735
val_f1_score	0.273	0.268	0.279	0.281	0.226	0.236
val_fbeta_score	0.43	0.424	0.436	0.44	0.367	0.348
val_precision	0.17	0.167	0.174	0.175	0.138	0.154
val_recall	0.699	0.69	0.697	0.707	0.626	0.508
val_rocauc	0.699	0.693	0.703	0.708	0.641	0.632
val_score_métier	0.4	0.39	0.408	0.415	0.301	0.297

Après optimisation des hyperparamètres et du seuil de probabilité de Light GBM évoqué précédemment (partie V.2.), nous obtenons les résultats suivants pour le jeu de validation et le jeu de test.

Run Name:	pip final	pip jeu test
Start Time:	2023-06-26 19:38:03	2023-06-26 19:49:23
End Time:	2023-06-26 19:38:26	2023-06-26 19:49:24
Duration:	22.7s	434ms

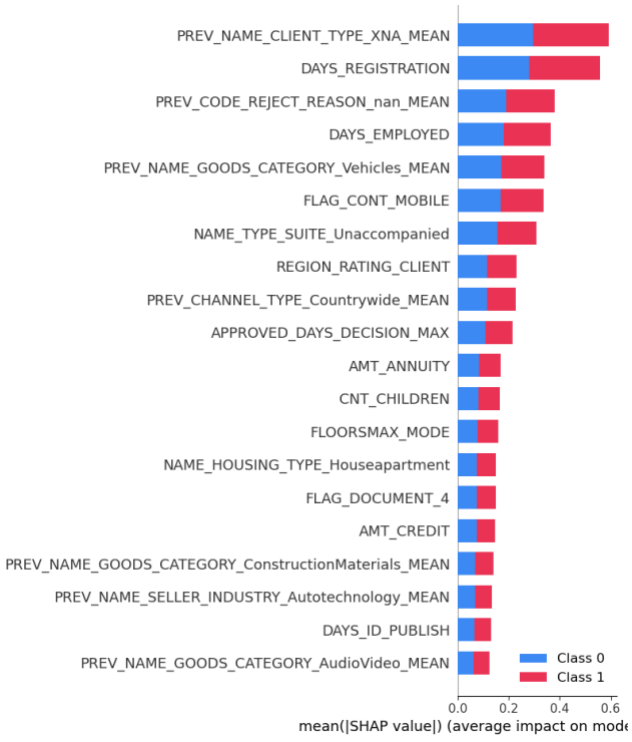
› Parameters

⌄ Metrics

Show diff only

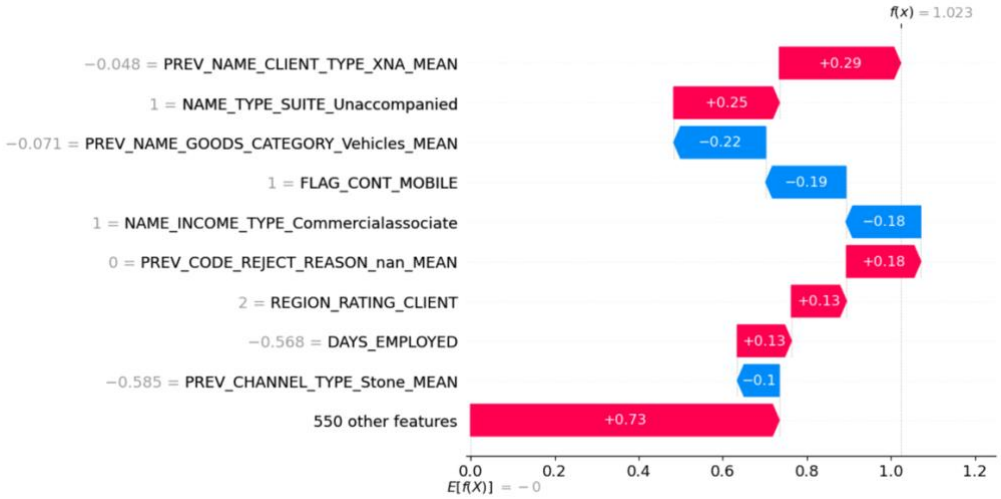
val_accuracy	0.745	0.742
val_f1_score	0.295	0.293
val_fbeta_score	0.443	0.44
val_precision	0.19	0.188
val_recall	0.663	0.663
val_rocauc	0.707	0.706
val_score_métier	0.419	0.416

V - L'interprétabilité globale et locale du modèle



Nous commençons l'analyse d'importance, d'un point de vue global, avec les features les plus importantes pour un modèle basé sur un ensemble de validation. Ici, nous avons utilisé summary_plot. Ce type de tracé agrège les valeurs SHAP pour toutes les features et tous les échantillons de l'ensemble sélectionné. Ensuite, les valeurs SHAP sont triées, de sorte que la première affichée est la feature la plus importante.

Les deux features les plus importantes sont *PREV_NAME_CLIENT_TYPE_XNA_MEAN* et *DAYS_REGISTRATION*.



D'un point de vue local, on s'intéresse cette fois-ci à l'impact des features pour la décision du modèle par rapport à une observation en particulier.
Dans le graphe ci-dessus, on peut voir l'impact de chacune des caractéristiques de l'individu choisi et comment ces caractéristiques impactent la prédiction. En bleu sont représentées les caractéristiques ayant une SHAPley valeur négative et donc une contribution négative et en rouge les caractéristiques ayant une SHAPley valeur positive, donc une contribution positive.
Pour le client ID 284800 par exemple, on voit que les deux features les plus influentes dans la décision de la sortie de ce client sont celles citées précédemment.

VI - Les limites et les améliorations possibles

La modélisation est ici essentiellement basée sur la création d'une métrique d'évaluation propre au métier, permettant de pénaliser plus fortement les Faux Négatifs (FN - mauvais client prédit bon client : donc crédit accordé et perte en capital). Cependant, il pourrait être plus pertinent de collaborer avec des équipes métier afin de créer et utiliser une métrique peut être plus pertinente.

De la même manière, les équipes métier pourraient également collaborer dans le cadre du feature engineering afin de créer des variables pertinentes dans un sens métier.

Un autre axe d'amélioration serait de tester d'autres hyperparamètres ou d'autres algorithmes de classification.

VII – L’analyse du Data Drift

Le Data Drift est une variation des données du monde réel par rapport aux données utilisées pour tester et valider le modèle avant de le déployer en production.

L'approche pour détecter le data drift consiste à utiliser des tests statistiques qui comparent la distribution des données de référence (reference data) et des données actuelles ou current data (données de production). Si la différence entre deux distributions est significative alors un drift s'est produit. Pour un dataset volumineux (avec > 1 000 observations dans l'ensemble de données de référence), la Distance de Wasserstein est utilisée pour les colonnes numériques (n_unique > 5), et pour les colonnes catégorielles ou numériques (avec n_unique <= 5) c'est la Divergence de Jensen--Shannon qui est appliquée.

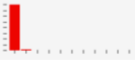
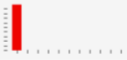
Dataset Drift

Dataset Drift is NOT detected. Dataset drift detection threshold is 0.5

121	10	0.0826
Columns	Drifted Columns	Share of Drifted Columns

Drift is detected for 8.264% of columns (10 out of 121).

Search

Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
SK_ID_CURR	num			Detected	Wasserstein distance (normed)	9.18745
AMT_REQ_CREDIT_BUREAU_QRT	num			Detected	Wasserstein distance (normed)	0.47302
AMT_REQ_CREDIT_BUREAU_MON	num			Detected	Wasserstein distance (normed)	0.280117
AMT_GOODS_PRICE	num			Detected	Wasserstein distance (normed)	0.212161
AMT_CREDIT	num			Detected	Wasserstein distance (normed)	0.209213
AMT_ANNUITY	num			Detected	Wasserstein distance (normed)	0.160231
AMT_REQ_CREDIT_BUREAU_WEEK	num			Detected	Wasserstein distance (normed)	0.152966
NAME_CONTRACT_TYPE	cat			Detected	Jensen-Shannon distance	0.14678
DAYS_LAST_PHONE_CHANGE	num			Detected	Wasserstein distance (normed)	0.137271
FLAG_EMAIL	num			Detected	Jensen-Shannon distance	0.123534

On constate donc qu'il n'y a pas de data drift entre nos données de référence (données d'entrainement) et nos données actuelles (données de test), bien qu'un léger drift ait été détecté dans 10 de nos colonnes.