NOTE METHODOLOGIQUE

Projet 07 : Implémenter un modèle de scoring

Ce document présente la démarche suivie lors de l'élaboration du modèle de scoring, il expose les différentes étapes, et les résultats d'analyses.

Dai TENSAOUT
Parcours Data Science Openclassroom 2023

Contents

Rappel de la mission

Le projet « Implémentez un modèle de scoring » comporte différentes missions essentielles qui visent à répondre aux points suivants :

Dans un premier temps la construction d'un modèle de scoring qui permettrait de calculer la probabilité de défaut d'un client automatiquement en s'appuyant sur des données variées relatives aux clients (données comportementales, données provenant d'autres institutions financières, etc.).

Ensuite, mettre en place un Dashboard interactif destiné aux chargés de relation client, ce dashboard leur permettra d'obtenir des informations détaillées sur les raisons de l'acceptation ou du refus d'un crédit. De plus permettre au client de disposer de ses informations personnelles et de les explorer facilement.

Méthodologie d'entraînement du modèle

Dans le but de produire un pipeline de construction d'un modèle de Machine Learning et de son déploiement, nous avons adopté une approche orientée objet pour la structure générale des dossiers. Cette structure comprend plusieurs classes qui offrent des fonctionnalités liées aux différentes étapes de notre pipeline, telles que l'ingestion des données, la génération de rapports, mais aussi des fonctions de journalisation et de gestion des exceptions sont également intégrées pour faciliter la gestions des erreurs. Ceci nous a permis de produire un pipeline réutilisable et modulables.

Pipeline d'entraînement

Notre pipeline d'entraînement est composé de plusieurs étapes :

Acquisition des données, nous avons récupéré les données en suivant le lien Home Credit Default Risk. Une fois les données récupérées les premières étapes à ne pas négliger sont **l'analyse exploratoire** et le **traitement des données brutes**, en effet pour espérer produire un ***modèle de classification*** performant nous avons effectuer une étape de **prétraitement** des données dans le but de réduire le bruit et de nettoyer le plus possible notre base de données, nous citons par exemple l'étape du ***feature engineering*** qui consiste à produire de nouvelles variables explicatives, mais aussi des étapes **d'imputation** de valeurs manquantes et de **normalisation** des données.

Dans un projet de Machine Learning les étapes d'exploration et de nettoyage des données sont chronophages, étant donné que le projet est long et demande beaucoup de temps, nous avons pris la décision d'utiliser un Kernel Kaggle voir kernel adapté à notre problématique. Cela nous permet de bénéficier d'un gain de temps significatif dans ces étapes cruciales du projet.

Pour illustrer les différentes étapes je vous propose de voir le schéma suivant :

Figure 1 : pipeline d'entraînement d'un modèle de Machine Learning

Les étapes du pipeline sont réalisées de manières itératives, ce qui permet une amélioration continue du modèle.

Dans ce qui suit, nous exposerons les différents modules utilisés dans les principales étapes du pipeline :

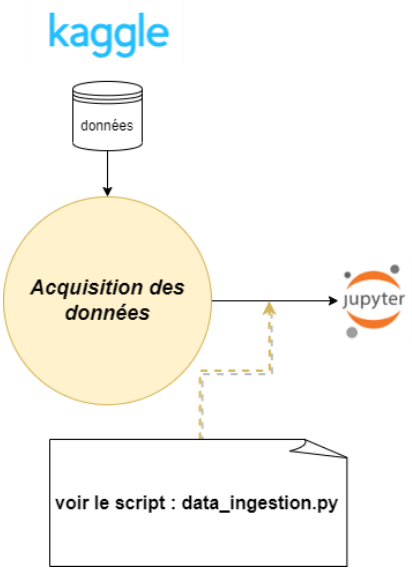


Figure 2 : acquisition des données de la base

data_ingestion.py

Classe "DataIngestion":

Cette classe est utilisée pour importer les données à partir d'un fichier CSV spécifié.
La méthode "get_files_names" extrait les noms de fichiers présents dans le répertoire spécifié, en supprimant les extensions.
La méthode "import_file" importe le fichier CSV spécifié et retourne un DataFrame pandas correspondant aux données.
Le paramètre optionnel "reduce_memory_usage" permet de réduire l'utilisation de la mémoire en optimisant les types de données du DataFrame.

Durant l'étape de **d'exploration des données** un déséquilibre importante entre les deux classes 0 : bons clients et 1 : moins bons clients ont été identifier, pour remédier à ce problème un ***undersampling*** est réalisé au niveau du script **transformation des données.**

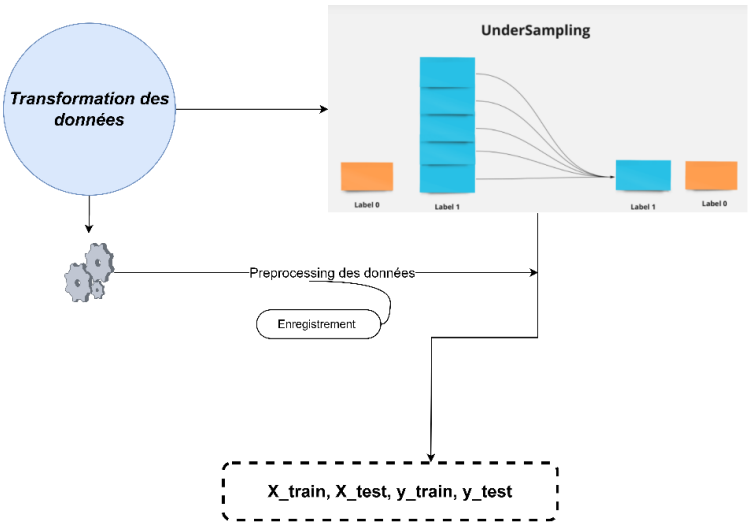
data_transformation.py

Méthode initiate_data_transformation : Cette méthode effectue les transformations principales sur les données. Elle prend en paramètres les ensembles d'entraînement et de test, ainsi que des paramètres pour l'échantillonnage *Undersampling*.

Les étapes de transformation incluent :

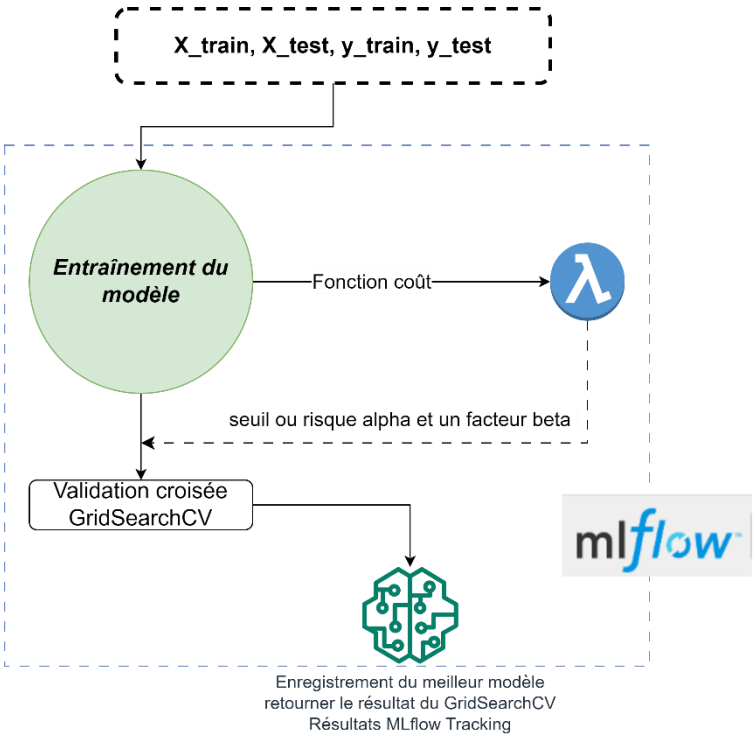
a. *Undersampling* (optionnel) : Si le param undersampling est activé, les données sont sous-échantillonnées à l'aide de la méthode RandomUnderSampler de imblearn.

b. Extraction des colonnes numériques et catégorielles : Les colonnes numériques et catégorielles sont extraites à l'aide de la classe RapportDataFrame définie dans un autre module.



Un préprocesseur est créé en utilisant la classe ColumnTransformer de **sklearn**. Il contient deux étapes de transformation : une pour les colonnes numériques, qui comprend l'imputation des valeurs manquantes avec la moyenne et la mise à l'échelle des *featurees*, et une pour les colonnes catégorielles, qui comprend l'imputation des valeurs manquantes avec la valeur "missing" et l'encodage one-hot-encoder.

Entraînement d'un modèle avec une validation croisé **GridSearchCV**



model_trainer.py

Ce script évalue un modèle en utilisant une validation croisée avec GridSearchCV. Il prend les données d'entraînement (**X_train** et **y_train**) ainsi que les données de test (**X_test** et **y_test**). Les autres paramètres incluent le modèle à évaluer, les hyperparamètres à tester (**params**), ainsi que les paramètres **alpha** et **beta** pour la fonction fonction_cout_metier.

Une étape de tracking est implémentée avec **mlflow** afin de garder en trace les différents résultats de l'étapes d'entraînement et du fin tuning des hyperparamètres.

Parmi les deux points spécifiques au contexte métier, la gestion du déséquilibre coût métier entre un faux négatif (FN – mauvais client prédit bon, 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), dans l'optique de la gestion de se déséquilibre nous avons construit un score métier qui nous permet de minimiser le coût d'erreur de prédiction des FN et FP : fonction_cout_metier.

Nous supposons que le coût d'un FN est dix fois supérieur au coût d'un FP. Ce qui revient à minimiser la fonction :

$$f = FP + 10\,FN$$

Nous savons que le rappel (Recall) et la précision (precision) sont deux mesures de performances couramment utilisées dans l'évaluation des modèles de classification. La précision évalue la capacité d'un modèle à effectuer des prédictions positives correctes, tandis que le rappel évalue la capacité du modèle à trouver tous les exemples réellement positifs. Nous utiliserons ce principe afin d'essayer de minimiser la fonction précédente de la manière suivante :

$$f = \left(\frac{1}{precision}\right) + 10\,\left(\frac{1}{Recall}\right)$$

Cette fonction calcule le coût d'une prédiction en utilisant les vraies valeurs (**y_true**) et les prédictions (**y_pred**). Les paramètres **alpha** et **beta** sont utilisés pour ajuster le calcul du coût :

1. Binarise les prédictions **y_pred** en utilisant le seuil **alpha**, de sorte que les valeurs supérieurs ou égales à **alpha** soient définies comme 1, et les autres comme 0, rappelons que la classe 1 sont les clients non solvables.
2. Calcule les éléments de la matrice de confusion (True Negative, False Positive, False Negative, True Positive) en comparant les valeurs binarisées des prédictions avec les vraies valeurs **y_true**.
3. Calcule la précision en divisant le nombre de vrais positifs (TP) par la somme des vrais positifs et des faux positifs (TP + FP).
4. Calcule le recall en divisant le nombre de vrais positifs (TP) par la somme des vrais positifs et des faux négatifs (TP + FN).
5. Calcule et renvoie le coût en utilisant la formule (1 / précision + beta / recall).

Nous utilisons cette fonction dans la partie GridSearchCV pour optimiser les hyperparamètre du modèle.

Interprétabilité globale et locale du modèle

Dans un souci de transparence lors de la prise de décision d'octroyer ou non un crédit pour un client donnée, l'interprétabilité de cette décision est essentielle dans ce contexte. En effet il est primordial qu'un chargé de relation clientèle lorsqu'il est en face à face avec le client puisse lui expliquer de manière claire la décision envisagée concernant son crédit. Afin de simplifier cette explication nous avons mis en place un graphique qui permet de tracer l'importance de certaines données qui ont pesées dans la décision. Pour répondre à cette exigence nous utilisons la libraire SHAP.

Exemple de sortie :

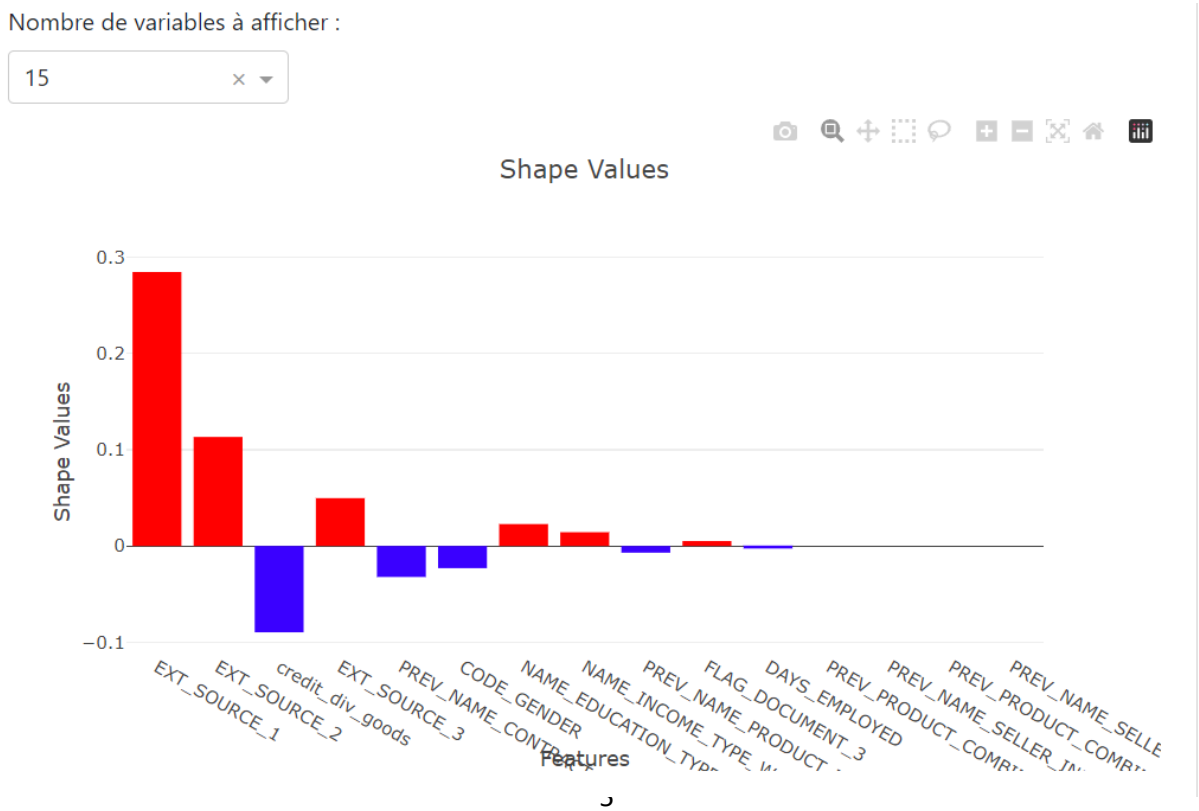
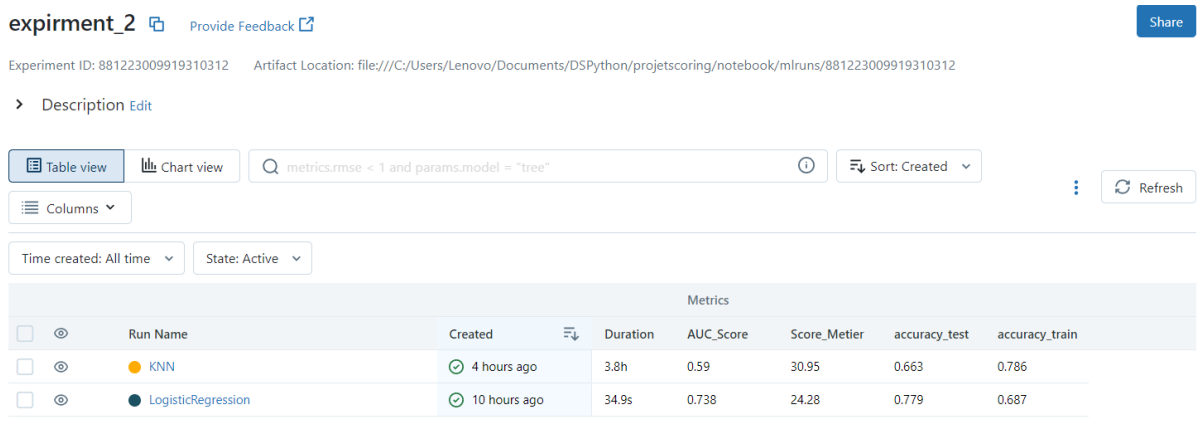


Figure 3 Exemple de sortie SHAP Values explication locale pour un client donnée : ici client Insolvable

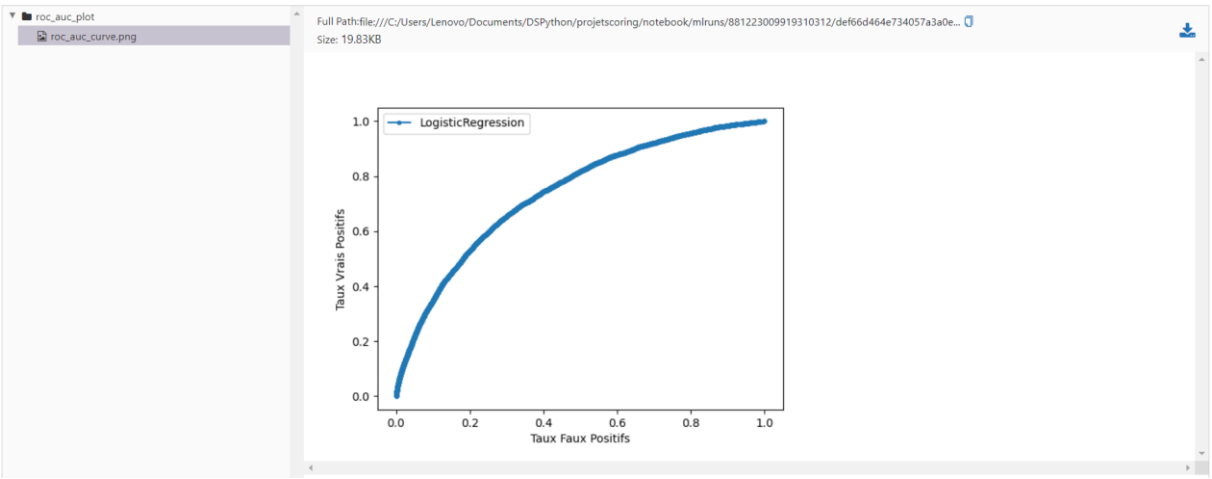
Résultats MLflow

Nous utilisons MLflow à des fins de tracking, en effet pour garder une trace des différents résultats de la modélisation de l'évaluation du modèle, mlflow propose des outils de suivis et la possibilité de la visualisation des résultats via MLflow UI

Exemple de sortie Mlflow :



Possibilité de suivre des artifacts comme des figure et graphiques :



Pour accéder aux résultats des expérimentations veuillez suivre le lien :

https://github.com/DaiTensa/projetscoring/tree/main/notebook/mlruns

L'analyse du Data Drift

Pour pouvoir détecter dans le futur un éventuel **Data Drift** (lien : Evidently) est une bibliothèque Python open source pour les data scientists et les ingénieurs Machine Learning, elle permet d'évaluer, de tester et de surveiller les performances des modèles Machine Learning, de la validation à la production.

Un exemple de rapport produit avec Evidently est disponible ici : https://github.com/DaiTensa/projetscoring/blob/main/notebook/report.html

Liens Github – Dashboard – API – Projet Scoring

Pour accéder au code source du projet 07 scoring : https://github.com/DaiTensa/projetscoring

Pour accéder au code source du Dashboard : https://github.com/DaiTensa/projet7-oc-dashboard

Pour accéder au code source de l'API : https://github.com/DaiTensa/projet7-oc-api

Les limites et les améliorations possibles

Les limites

- L'étapes de modélisation est chronophage, d'où l'utilisation de peu d'algorithmes de Machine Learning.
- Manque de connaissance métier pour déterminer le seuil alpha (niveau de prise de risque, restrictif ou permissif) et le poids (beta le coût réel d'un Faux Négatif).

Améliorations possibles

- Utiliser d'autres modèles notamment des modèle *gradient boosting* tels que LightGBM, XGBoost…etc.
- Ajouter plus de fonctionnalités au dashboard, un formulaire détaillé d'une vingtaine d'entrées pour renseigner les données du client.
- Réduire le nombre de *features* par l'utilisation de méthodes de sélection de features en amont de la modélisation.
- L'élaboration d'un dictionnaire de définition des différentes variables, afin de faciliter la lecture des explications locales d'une prédiction donnée.
- Déploiement du modèle dans un bucket dans le cloud, et mettre en place une approche CI/CD.