Rapport Projet 2

Introduction

Ce projet fait suite à un premier projet dans lequel des modèles de machine learning ont été entraîné sur des données préalablement préparées. Ces procédures de traitements et d'entraînement ainsi que les modèles sauvegardés sont réutilisés pour ce projet afin de déployer une API permettant d'interroger ces modèles.

Le projet est disponible sur Github à cette adresse

Partie API

L'API permet d'interroger des modèles de machine learning préalablement entraînés pour obtenir des prédictions et le score des modèles sur les données de tests.

Entraînement des modèles

La première étape a été l'entraînement des différents modèles. Pour ce faire un script **train.py** a été développé. Il récupère le fichier *rain.csv* qui contient les données météo brute, les nettoyer puis entraîner les différentes modèles avec ces données.

Lors du nettoyage certaines lignes non pertinentes son retirées notamment celles contenant des valeurs nulles pour la variable cible. Le script va ensuite créé un pipeline de pre-processing pour remplacer les valeurs nulles des variables catégorielle et quantitative, standardiser les données et encoder les variables catégorielle.

Ce pipeline est sauvegardé dans un fichier *preprocessor* qui est utilisé dans l'API pour appliquer les même traitements aux données fournies par l'utilisateur lors des prédictions.

Les données sont séparées en jeu d'entraînement et de test, les modèles sont entraîné puis sauvegardé dans un fichier pour être utilisé par l'API.

API

L'API est dévelopée avec Python 3 et le framework FastAPI

Une fois le projet récupérer et les paquets du fichier *requirements.txt* installés, il est possible de tester l'API via la commande `python3 api_models.py` qui est alors accessible à l'adresse *http://localhost:8000*

Authentification

Hormis pour les endpoints ***/*** et ***/docs*** il est nécessaire de s'authentifier via le header **authorization-header**. Le header doit être de cette forme : *Basic b64string*

b64string est une chaîne de caractère en base64 correspond correspondant à un couple *user:password*

Par exemple on peut s'authentifier comme **authorization-header** : *Basic YWxpY2U6d29uZGVybGFuZA==* avec *YWxpY2U6d29uZGVybGFuZA==* étant la chaîne de caractères *alice:wonderland* encodée en base64.

Les différents couples d'utilisateur et mot de passe autorisés se trouve le fichier de l'API. Cette méthode d'authentification reste basique et pourrait être améliorée par exemple en stockant les identifiants dans une base données et en demandant l'authentification via un formulaire.

Endpoints

Cette partie liste les différents endpoints de l'API et leur description.

/

Cet endpoint permet de contrôler si l'API est en fonctionnement. Il renvoie un objet JSON indiquant que l'API est en fonctionnement et peut notamment être utilisé par des tâches automatisées pour vérifier que l'API est toujours disponible

/docs

Fastapi fournit cet endpoint qui permet d'accéder à une documentation de l'API et de tester les différents endpoints. On peut y trouver une description des différents endpoints et des schemas et il est possible d'effectuer des requêtes depuis l'interface et d'obtenir le retour des endpoints.

/info

Renvoie la liste des modèles entraînés qui sont disponibles dans l'API.

/score

Cet endpoint nécessite de sélectionner le nom d'un modèle et de l'inclure dans le corps de la requête. Le résultat renvoyé est le score obtenu par ce modèle sur les données de test.

/predict/{model}

Ici le routage dynamique de FastAPI permet de générer un endpoint pour chaque modèle défini dans la classe **ModelsName.** Il est donc possible d'ajouter de nouveau modèle dans l'API qui seront directement disponible via de nouveaux endpoints.

L'API attend l'envoie d'un fichier CSV contenant les données pour lesquels on veut obtenir une prédiction. Ces données seront nettoyées et standardisées de la même manière que les données d'entraînement et de tests grâce au pipeline enregistré par le script **train.py**.

Le résultat renvoyé est une liste contenant la prédiction obtenue pour chaque entrée présente dans le fichier.

Tests

Des tests développés avec le framework Pytest sont présents dans le dossier **tests** du dépôt.

Ces tests ont servis pour le développement de l’API afin d’éviter d’éventuelles régressions.

Tests avec docker-compose

Présentation :

3 traitements ont été créés afin de tester l'accès aux requêtes des 3 Endpoint /info, /predict et /score

Test /info

Le traitement “tests/repo_test_acces_info/**test_access_info.py**” permet de vérifier l’accès aux informations selon les cas suivants :

- Utilisation d’une authentification valide en format base64 + user habilité. Le code statut attendu : 200
- Utilisation d’un mauvais format. Le code statut attendu : 400
- Utilisation d’un format en base64 mais d’un user non habilité. Le code statut : 403

Test /predict

Le traitement “tests/repo_test_acces_predict/**test_access_predict.py**” permet de vérifier l’accès aux prédictions du modèle SVC selon les cas suivants :

- En utilisant les données du fichier “test.csv” avec un format “csv”. Le code statut attendu : 200
- En utilisant les données du fichier “test.csv” avec un format autre que “csv”. Le code statut attendu : 400

Test /score

Le traitement “tests/repo_test_acces_score/**test_access_score.py**” permet de vérifier l'accès au score du modèle SVC. Le code statut attendu : 200

Les 3 traitements vont écrire le résultat des tests dans le fichier “log.txt” accessible dans le répertoire du volume **api_rain_log** dans : '/var/lib/docker/volumes/tests_api_rain_log/_data'

Lancement des tests

Fichier “docker-compose.yml”

Le lancement des 3 tests se fait en utilisant le fichier “tests/docker-compose.yml”. Ce fichier permet la création et le lancement des containers pour chacun des 3 tests en plus du container de l’API à interroger.

Les containers créés partagent le même NETWORK : api_rain_network

```
networks:
  api_rain_network:
    driver: bridge
    ipam:
      driver: default
      config:
      - subnet:  172.50.0.0/16
        gateway: 172.50.0.1
```

Et les 3 containers des tests auront comme volume partagé api_rain_log

Script “tests/setup.sh”

Afin d’exécuter les 3 tests, il faut d’abord lancer le traitement “tests/**setup.sh**” qui permet de :

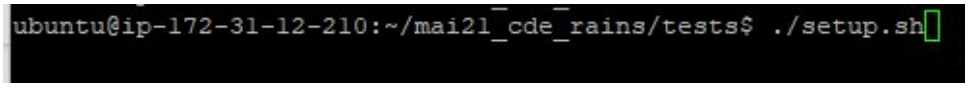
- Construire les différentes images Docker pour :
  - Les 3 tests via les fichiers Dockerfile présents dans chacun des répertoires où se trouvent les traitements.
  - L’API à interroger via le fichier Dockerfile présent dans le répertoire “mai21_cde_rains”
- Exécuter le fichier “yml” via la commande “docker-compose up”

Script “tests/off.sh”

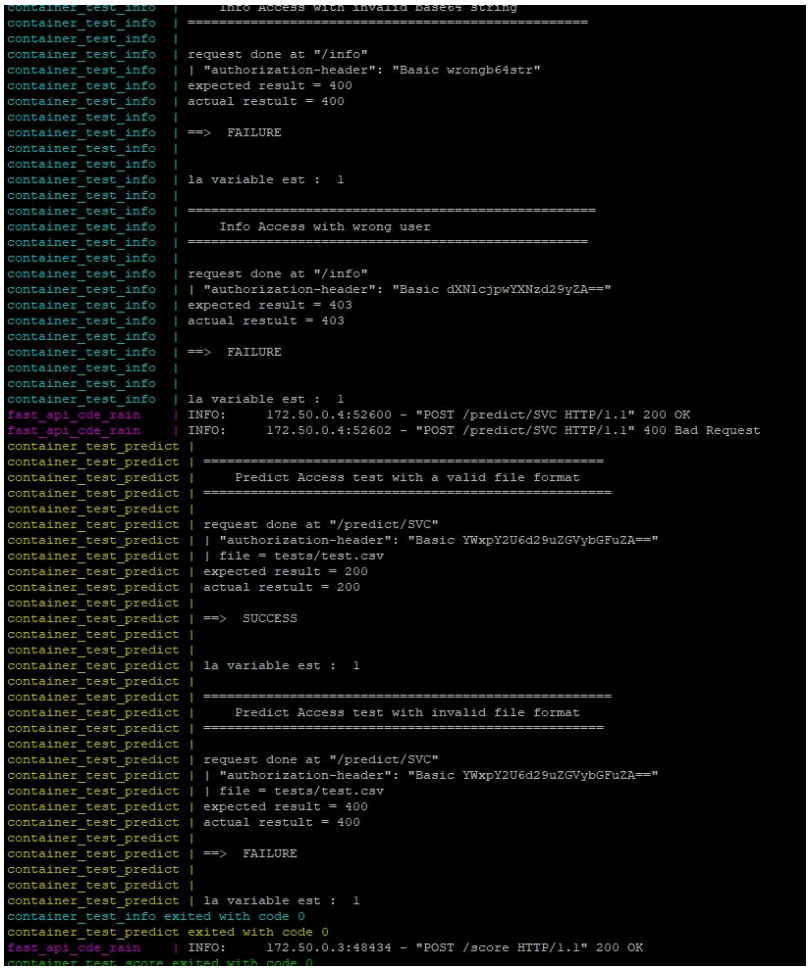
Le traitement “tests/**off.sh**” permet la suppression l’intégralité des containers et des images créées dans le cadre des tests.

Démonstration

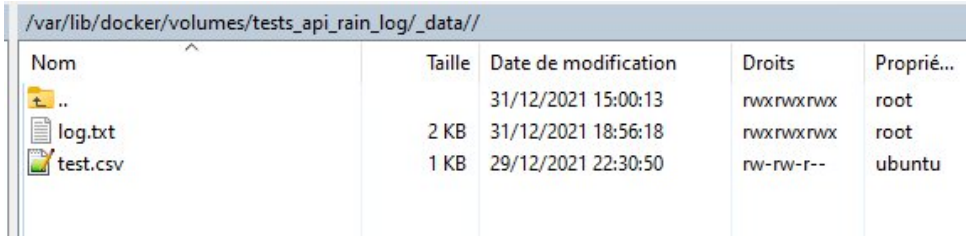
1. Lancement du traitement “setup.sh”



Le détail de création des images + la log générée par les tests vont apparaître dans le terminal :



2. Vérifier l’alimentation du fichier “log.txt” à l’emplacement du volume “api_rain_log” :



Aperçu du contenu du fichier :

```
==================================================
    Info Access with wrong user
==================================================

request done at "/info"
| "authorization-header": "Basic dXN1cjpwYXNzd29yZA=="
expected result = 403
actual restult = 403

==>  FAILURE


==================================================
    Predict Access test with a valid file format
==================================================

request done at "/predict/SVC"
| "authorization-header": "Basic YWxpY2U6d29uZGVybGFuZA=="
| file = tests/test.csv
expected result = 200
actual restult = 200

==>  SUCCESS


==================================================
    Predict Access test with invalid file format
==================================================

request done at "/predict/SVC"
| "authorization-header": "Basic YWxpY2U6d29uZGVybGFuZA=="
| file = tests/test.csv
expected result = 400
actual restult = 400

==>  FAILURE


==========================================
    Score Access test
==========================================

request done at "/score"
| "authorization-header": "Basic YWxpY2U6d29uZGVybGFuZA=="
| "model_name": "SVC"
expected result = 200
actual restult = 200

==>  SUCCESS
```

Déploiement avec KUBERNETES

La mise en place de KUBERNETES s’appuie sur les concepts fondamentaux de celui-ci.

C’est-à-dire :

- Un fichier de déploiement des pods : k8s-deployment-cde-rain.yml
- Un fichier de service : k8s-service-cde-rain.yml
- Un fichier de mise à disposition externe au cluster via INGRESS : k8s-ingress-cd-rain.yml

Cette mise en place est classique dans l’utilisation KUBERNETES.

Par défaut, le port de l’API visible à l’extérieur est 8002 au lieu de 8000 pour éviter conflit avec d’autres logiciels utilisant ce port par défaut.

Le fichier Dockerfile permet de produire un container exposant notre API sur le port 8000.

Le container contient le répertoire tests qui n’est pas obligatoire dans notre cas.

Le fichier README.md partie illustre de manière condensée toutes les étapes à effectuer pour pouvoir lancer sous KUBERNETES en local.

Un chapitre FAQ permet d’avoir une solution à certains dysfonctionnements.

La principale particularité de cette mise en place est de générer le container de l’API en local.

Nous n’avons pas mis à disposition le container de l’API. D’où les ordres contenus dans le fichier README.md pour pouvoir inclure dans le docker local de KUBERNETES l’image du container de l’API.