Machine Learning API Predictions of churn Formation project P2 DataScientest

Objectif

L'objectif de ce projet est de déployer un modèle de Machine Learning. Ce repo Github contient les fichiers suivants :

- le fichier source de l'API : main.py
- le Dockerfile de l'API et le fichier requirements.txt associé
- l'ensemble des fichiers utilisés pour créer les tests
- les fichiers de déploiements de Kubernetes dans le repertoire kubernetes
- le fichier machine_learning.py contenant le code pour entrainer et stocker les modèles KNN et régression logistique
- les fichiers model_knn.joblib et model_logistic.joblib contenant les modèles entrainés, pour réutilisation dans main.py
- le fichier features.json contenant l'ensemble des features des données utilisées pour entrainer les modèles, pour réutilisation dans main.py

Modèles de Machine learning

Cf notebook Projet churn_mar22bcde_KParra_AdeOliveira livré pour le projet 1

- Audit + Exploration des données
- Visualisation
- Entraînement et évaluation de modèles de machine learning

L'API

L'API est construite avec FastAPI dans le fichier main.py. Cette API permet d'interroger 2 modèles : KNN et Regression Logistique.

Routes disponibles :

/	Message de bienvenue
/status	Test de connexion Accessibles pour les administrateurs via une page d'authentification
/models	Liste des noms des différents modèles disponibles Accessibles pour les utilisateurs et administrateurs via une page d'authentification
/models/{model_name}/prediction	Prédiction du Churn d’un client à partir d'un des modèles référencés au point précédent, et performance de l'algorithme sur les jeux de tests.

Une identification basique est utilisée. La liste d'utilisateurs/mots de passe suivante est initialisée :

- alice: wonderland – type : user
- bob: builder – type : user
- clementine: mandarine – type : user
- axel : axdeo – type : admin
- karine : kparra – type : admin

Le container

Un container Docker a été créé pour déployer facilement l'API.
Les librairies Python à installer ainsi que leurs différentes versions sont détaillées dans le fichier requirements.txt.
Commandes pour construire le container docker dans le répertoire racine :

- `docker build -t api_churn .`

Commandes pour lancer l'API en local :

- `docker container run -p 8000:8000 api_churn`
- swagger disponible sur : http://127.0.0.1:8000/docs
- API disponible sur : http://127.0.0.1:8000/

Commandes pour ajouter l'API sur dockerhub :

- `docker login --username krineparra --password ...`
- `docker tag api_churn krineparra/api_churn`
- `docker image push krineparra/api_churn:latest`
- penser à redémarrer le déploiement kubernetes s'il tourne déjà pour prendre en compte les modifications de l'API

Les tests

Une série de tests a été créée dans le dossier **Tests** pour tester l'API conteneurisée.
On a pour cela créé un fichier docker-compose.yml associé à un shell.

Kubernetes

Pour permettre le déploiement de l'API sur 3 Pods un fichier de déploiement, un service et un ingress ont été créés dans le répertoire **Kubernetes.**

Commandes pour construire le déploiement k8s :

- `minikube start`
- `minikube addons enable ingress`
- `minikube dashboard --url=true`
- VM : `kubectl proxy --address='0.0.0.0' --disable-filter=true`
- `kubectl create -f deployment.yml`
- `kubectl create -f service.yml`
- `kubectl create -f ingress.yml`

- VM : ouvrir un tunnel :
  `ssh -i "data_enginering_machine.pem" ubuntu@(ip VM) -fNL 8000:192.168.49.2:80`
- API disponible sur : http://127.0.0.1:8000/docs
- VM : Dashboard k8s disponible sur : http:///(ip_VM):8001/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/