Année universitaire	**2023-2024**		
Filière	**Data Science**	**Année**	**M2**
Matière	**Machine Learning**		
Enseignant	**Haytham Elghazel, Thomas Ranvier**		
Intitulé TD/TP :	**Atelier 2 : Mise en production et déploiement continu**		
Contenu	• **Conteneurisation des processus** • **Mise en production d'algorithmes de Machine learning** • **Reporting** • **Entraînement et déploiement continu**		

Dans cet atelier pratique, vous allez travailler sur la mise en production du modèle de détection d'anomalies précédemment conçu et entraîné. Vous devrez implémenter une API de déploiement du modèle prédictif (dite API de serving), une interface utilisateur sous la forme d'une page web, mettre en place un reporting des performances du modèle et de l'évolution des données, et un processus d'entraînement et déploiement continu, le tout conteneurisé grâce à Docker.

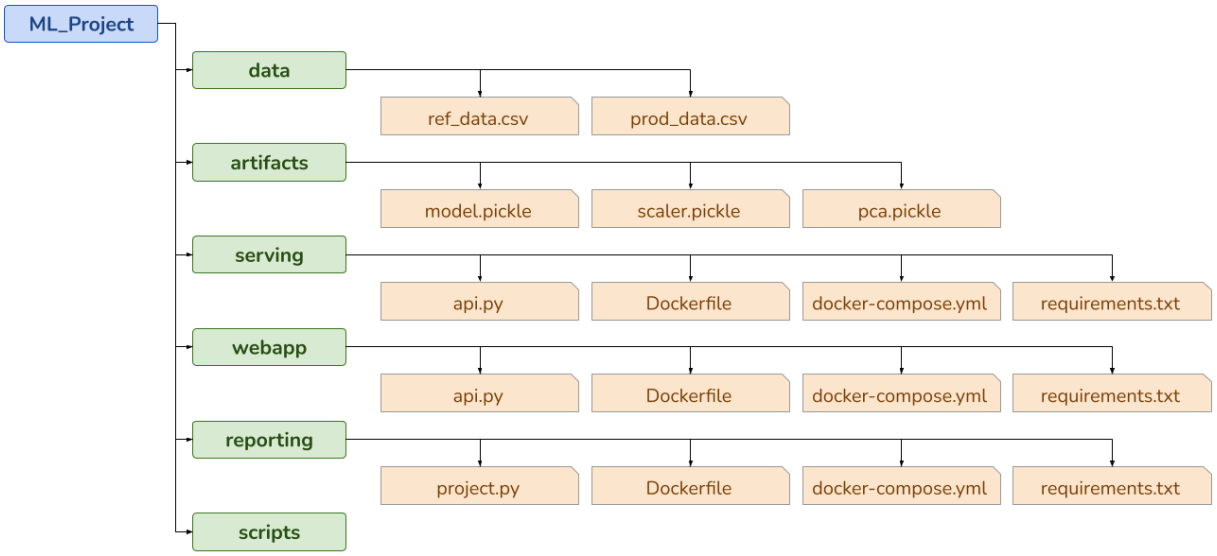
1. Préparation du projet

Tous les composants implémentés lors de ce TP seront conteneurisés à l'aide de Docker. Docker permet de créer une espèce de boîte dans laquelle on peut faire fonctionner une application : un conteneur. Un conteneur permet d'emballer une application avec tout ce dont elle a besoin pour fonctionner sur n'importe quel type d'ordinateur. On y regroupe tous les composants et dépendances nécessaires au fonctionnement de l'application. Cela permet de garantir que votre application fonctionne dans des conditions identiques sur n'importe quelle machine, facilitant ainsi le partage, la collaboration et le déploiement de votre application sur différentes plateformes.

L'orchestration des différents conteneurs Docker se fera via Docker Compose (ou Kubernetes pour ceux souhaitant une version plus avancée). Docker Compose permet de manager un ensemble de conteneurs Docker, afin de définir la manière dont ils peuvent interagir les uns avec les autres.

Vous allez installer Docker et Docker Compose sur votre machine, nous vous conseillons d'installer Docker Desktop qui contient l'engine Docker, ainsi que Compose : https://docs.docker.com/desktop/.

Au cours de votre progression dans le TP vous allez ajouter des fichiers à votre projet, veuillez vous référer à l'architecture de fichiers suivante afin d'organiser votre projet :



Le dossier *scripts* comprend tous les scripts/notebooks avec lesquels vous avez travaillé précédemment, ainsi qu'un fichier python contenant votre code factorisé.

Le dossier *data* contient un fichier appelé *ref_data.csv* qui servira de référence pour le reporting futur et que vous pouvez utiliser pour entraîner votre modèle. Créez un script qui transforme les données de votre dataset en vecteurs via le modèle d'embedding choisi et les sauvegarde en un fichier nommé *ref_data.csv*. Les données doivent être organisées de la manière suivante au sein de votre fichier csv (le modèle d'embedding de l'exemple est une PCA, le votre sera peut-être différent) :

PCA_1	PCA_2	...	PCA_n	target
0.59	-1.04	...	1.32	True
-0.25	0.68	...	-0.89	False
...	...	...	...	...

Utilisez ce fichier *ref_data.csv* afin d'entraîner votre modèle, enregistrez sous forme de fichiers pickle votre modèle entraîné, le modèle d'embedding, et le scaler (si applicable) au sein du dossier *artifacts*.

2. API de serving du modèle avec FastAPI

Maintenant que vous avez setup le projet et entraîné un modèle sur votre dataset, vous allez implémenter une API de serving pour votre modèle. Une API de serving permet à une application externe d'appeler votre modèle pour effectuer une prédiction sur une donnée. L'intérêt d'implémenter une telle API est de définir une interface standardisée avec laquelle une application souhaitant utiliser le modèle pourra communiquer et recevoir des prédictions. Ainsi, l'API de serving de votre modèle sera indépendante et pourra être utilisée par une ou plusieurs applications pour fournir des prédictions.

Les fichiers générés à cette étape doivent être placés au sein du dossier *serving*. Vous allez utiliser FastAPI pour le développement de votre API de serving. Créez le fichier *api.py*, renseignez vous sur l'utilisation du package de FastAPI sous python et implémentez un endpoint nommé *predict* contactable via la méthode POST. L'endpoint de prédiction recevra une donnée sous format JSON, votre code devra utiliser votre modèle d'embedding, scaler et modèle de prédiction pour traiter la donnée et obtenir une prédiction. Cette prédiction est retournée par votre endpoint. Le modèle d'embedding, scaler et modèle de prédiction utilisé par l'API sont ceux stockés dans le dossier *artifacts*. Chargez-les dans des variables globales au lancement de l'API afin qu'ils soient disponibles instantanément pour les prédictions.

Générez un fichier *requirements.txt* et ajoutez-y les dépendances python nécessaires au fonctionnement de votre API, un nom de package par ligne. Initialement votre fichier contiendra les deux lignes suivantes, vous le mettrez à jour lorsque vous ajouterez des fonctionnalités et dépendances à l'API :

```
fastapi
uvicorn
```

Générez un fichier *Dockerfile* et ajoutez-y le code suivant :

FROM python:3.10-slim	*Indique au conteneur de baser son environnement sur l'image Docker de Python 3.10, trouvée à l'adresse suivante : https://hub.docker.com/_/python*
WORKDIR /app	*Défini le directory de travail*
COPY api.py . **COPY requirements.txt .**	*Copie le code de votre API et le fichier de requirements*
RUN pip3 install --upgrade pip **RUN pip3 install -r requirements.txt**	*Met à jour pip et installe les packages indiquées dans le fichier de requirements*
CMD uvicorn api:app --host 0.0.0.0 --port 8080	*Héberge l'API sur le port 8080 de l'IP externe du conteneur*

Utilisez la commande **docker build …**, puis **docker run -p 8080:8080 …** pour exécuter le conteneur de l'API de serving, l'option **-p 8080:8080** permet de lier le port 8080 du conteneur au port 8080 de votre machine. Ainsi vous pouvez vérifier la bonne exécution de votre API en accédant à la page : http://localhost:8080/docs.

Maintenant que votre API est accessible, vous allez implémenter une simple interface web qui communiquera avec l'API.

3. Interface web avec Streamlit

Vous allez implémenter une interface web avec Streamlit. L'interface doit être très simple, on doit pouvoir y uploader une donnée de test (image/vidéo/audio selon le dataset choisi), et cliquer sur un bouton afin d'appeler l'endpoint de prédiction de l'API de serving. La prédiction obtenue en retour est affichée à l'utilisateur. Vous pouvez utiliser le package python **requests** afin d'envoyer une requête à l'API de serving. Après avoir suivi la suite des explications de cette section vous pourrez requêter l'endpoint de prédiction de votre API de serving via l'URL http://serving-api:8080/predict dans le code de votre interface web.

Créez un fichier *api.py* au sein du dossier *webapp*, il contiendra votre code pour l'interface. Comme pour l'API de serving vous devez créer et compléter un fichier *requirements.txt* et *Dockerfile*, la commande exécuter au sein du Dockerfile devra cette fois utiliser **streamlit run …** afin d'exécuter votre application Streamlit.

Une particularité de travailler avec Docker est que chaque conteneur possède une IP dynamique qui lui est attribuée à son lancement, rendant complexe la communication entre deux conteneurs. Afin de pallier ce problème vous utiliserez Docker Compose pour définir un réseau permettant de facilement connecter des conteneurs.

Créez un fichier *docker-compose.yml* dans le dossier *serving* et ajoutez-y le code suivant :

version: '3.8'	*Indique la version de Docker Compose*
networks: **prod_net:** **driver: bridge**	*Défini le réseau "prod_net" qui permettra à la webapp de communiquer avec l'API de serving*

services: **serving-api:** **container_name: serving-api**	*Crée le service serving-api, on utilisera le nom de ce service pour communiquer avec l'API de serving*
build: **context: .** **dockerfile: Dockerfile**	*Indique l'emplacement du Dockerfile*
volumes: **- ../data:/data** **- ../artifacts:/artifacts**	*Partage les dossier data avec l'API de serving, toute modification de /data depuis l'API sera effectif sur le dossier data de votre projet, le conteneur pourra également accéder aux artifacts pour les utiliser*
ports: **- "8080:8080"**	*Equivalent de l'option **-p 8080:8080** de la commande **Docker run** que vous utilisiez précédemment*
networks: **- prod_net**	*Indique le ou les network(s) auquel le service serving-api peut accéder*

Créez un fichier *docker-compose.yml* dans le dossier *webapp* et ajoutez-y le code suivant :

version: '3.8'	*Indique la version de Docker Compose*
networks: **serving_prod_net:** **external: true**	*Récupère le réseau précédemment défini pour l'API, le préfixe "serving_" s'ajoute automatiquement lors de la création du réseau, c'est le nom du dossier de l'API*
services: **webapp:** **container_name: webapp**	*Crée le service webapp*
build: **context: .** **dockerfile: Dockerfile**	*Indique l'emplacement du Dockerfile*
ports: **- "8081:8081"**	*Lie le port 8081 du conteneur au port 8081 de votre machine*
networks: **- serving_prod_net**	*Connecte l'application web au réseau de l'API de serving*

Vous pouvez désormais créer le réseau "serving_prod_net" et démarrer votre API de serving avec la commande suivante :

$ docker compose -f serving/docker-compose.yml up

Démarrez votre interface web en utilisant la commande **docker compose …**, une fois lancée vous pourrez y accéder sur votre navigateur à l'adresse http://localhost:8081/. Ajoutez les options **--build --force-recreate** aux commandes **docker compose …** en cas de modification de votre code, cela permet de forcer la recréation de l'image Docker, et ainsi, prendre en compte les modifications apportées depuis la dernière exécution.

4. Reporting avec Evidently

Vous allez maintenant utiliser la librairie Evidently afin de générer un rapport sur l'état de santé de votre modèle, ainsi que sur l'évolution des données. Pour le calcul de ses métriques, Evidently se basera sur le fichier *ref_data.csv* préalablement défini et le fichier *prod_data.csv* que vous allez créer. Ce dernier contiendra des feedbacks obtenus via votre application web. Implémentez un champ de saisie et un bouton "feedback" à votre application web sur lequel vous pourrez appuyer une fois une prédiction obtenue. L'appuie sur ce bouton

enverra la donnée uploadée, la prédiction reçue et la cible réelle entrée par l'utilisateur à l'endpoint "feedback" de votre API de serving que vous allez maintenant implémenter. Ce nouvel endpoint devra utiliser votre modèle d'embedding pour transformer la donnée sous format vectoriel et enregistrera le vecteur, la cible réelle, ainsi que la prédiction obtenue par votre modèle au sein du fichier *prod_data.csv*. Prenez garde à ce que les données soient traitées exactement de la même manière entre le fichier *ref_data.csv* et le fichier *prod_data.csv.* Les données doivent être organisées de la manière suivante au sein de *prod_data.csv* :

PCA_1	PCA_2	...	PCA_n	target	prediction
0.59	-1.04	...	1.32	True	True
-0.25	0.68	...	-0.89	False	True
...	...	...	...	...	

Créez le fichier *project.py* dans le dossier *reporting*, suivez le tutoriel de setup pour créer un projet Evidently qui sera utilisé pour le dashboard : https://docs.evidentlyai.com/get-started/tutorial-monitoring, code complet : https://github.com/evidentlyai/evidently/blob/main/examples/sample_notebooks/get_started_monitoring.py.

Créez le *Dockerfile*, *requirements.txt* et *docker-compose.yml*, ajoutez le code suivant au *Dockerfile* en complétant les lignes manquantes :

```
...
CMD python project.py && evidently ui --host 0.0.0.0 --port 8082
```

Complétez le fichier *docker-compose.yml* de *reporting* de manière à partager le volume *data* avec le Docker et lier le port 8082 du conteneur au port 8082 de votre machine. Lancez le conteneur avec la commande **docker compose ...**, vous pouvez accéder à l'interface web du dashboard à l'adresse : http://localhost:8082/. Vous devriez voir le projet de démo sur un dataset d'exemple.

Modifiez maintenant le projet Evidently pour l'adapter à vos données. Remplacez les données de références et de production d'exemple par les données de vos fichiers *ref_data.csv* et *prod_data.csv* respectivement. Explorez la documentation et modifiez le code du fichier *project.py* afin de générer un rapport avec des métriques de classification en plus de celles de data drift.

Vous avez mis en place un système de reporting statique, en pratique il serait plus intéressant de programmer la génération automatique de rapports à intervalles de temps réguliers et de mettre en place un système de monitoring en temps réel. Il serait trop long d'intégrer cette partie à ce TP, pour un bonus vous pouvez vous intéresser à l'utilisation de CronJobs avec Kubernetes pour programmer une exécution régulière des rapport et à la mise en place de la stack prometheus+grafana qui permettra de récupérer les métriques extraites par Evidently et de les afficher en temps réel sur un dashboard web.

5. Déploiement d'un nouveau modèle en production

Pour finir ce TP, vous allez automatiser l'entraînement de votre modèle sur la concaténation des données de référence et les nouvelles données provenant de votre environnement de production, puis le déploiement de ce nouveau modèle à la place de l'ancien.

Le réentrainement et la mise en production d'un nouveau modèle peut être déclenché automatiquement selon de nombreux triggers : lorsque les performances du modèle diminuent, lorsqu'un drift est détecté entre les données de référence et celles de production, lorsque assez de nouvelles données de production ont été ajoutées, etc. Vous allez mettre en place un ré-entraînement et déploiement automatique de votre modèle basé sur la quantité de nouvelles données de production.

Vous allez modifier l'endpoint "feedback" de votre API de serving pour déclencher la mise à jour du modèle. Définissez un seuil **k** qui sera un trigger, toutes les **k** données de production reçues, la mise à jour du modèle sera lancée. Si le nombre de ligne dans le fichier *prod_data.csv* après avoir ajouté le feedback est un multiple de **k**, appelez une fonction de votre code factorisé qui entraîne un nouveau modèle de prédictions sur la concaténation des données de référence et de production, et met à jour les fichiers pickle du dossier *artifacts*. Une fois le modèle entraîné, remplacez le modèle de prédiction utilisé par l'API par les nouveaux en modifiant les variables globales. Ainsi, les nouveaux appels de prédiction à l'API utiliseront le nouveau modèle de production.

Bonus :

- Mettre en place un système de monitoring en temps réel avec prometheus+grafana
- Comparer le modèle nouvellement entraîné avec le modèle actuellement en production afin de vérifier la supériorité du nouveau modèle, ne remplacer le modèle en production que si c'est le cas
- Mettre en place un processus de fine-tuning du dernier modèle de production plutôt que de réentrainer un modèle de 0
- Utiliser Kubernetes plutôt que Docker Compose pour la gestion des conteneurs, volumes et réseaux