PROJET 2 : Déploiement du modèle

Introduction

L'objectif de ce projet est de mettre en production un modèle de prédiction de désabonnement des clients d'une entreprise de télécommunication.

1) Présentation du jeu de données

Le jeu de données a 5986 lignes et 21 colonnes.

Il existe 17 caractéristiques catégorielles :

- CustomerID : ID client unique pour chaque client
- Genre : si le client est un homme ou une femme
- SeniorCitizen : Si le client est un senior ou non (1, 0)
- Partenaire : Si le client a un partenaire ou non (Oui, Non)
- Dépendant : Si le client a des personnes à charge ou non (Oui, Non)
- PhoneService : si le client dispose d'un service téléphonique ou non (Oui, Non)
- MultipeLines : Si le client a plusieurs lignes ou non (Oui, Non, Pas de service téléphonique)
- InternetService : fournisseur de services Internet du client (DSL, fibre optique, non)
- OnlineSecurity : si le client dispose ou non d'une sécurité en ligne (Oui, Non, Pas de service Internet)
- OnlineBackup : Si le client dispose ou non d'une sauvegarde en ligne (Oui, Non, Pas de service Internet)
- DeviceProtection : si le client dispose ou non d'une protection d'appareil (Oui, Non, Pas de service Internet)
- TechSupport : Si le client dispose d'un support technique ou non (Oui, Non, Pas de service Internet)
- StreamingTV : si le client dispose ou non de la télévision en streaming (oui, non, pas de service Internet)
- StreamingMovies : si le client a des films en streaming ou non (Oui, Non, Pas de service Internet)
- Contrat : la durée du contrat du client (mensuel, un an, deux ans)
- PaperlessBilling : la durée du contrat du client (mensuel, un an, deux ans)
- PaymentMethod : Mode de paiement du client (Chèque électronique, Chèque postal, Virement bancaire (automatique), Carte de crédit (automatique))

Ensuite, il y a 3 caractéristiques numériques :

- Ancienneté : nombre de mois pendant lesquels le client est resté dans l'entreprise
- MonthlyCharges : Le montant facturé au client mensuellement
- TotalCharges : le montant total facturé au client

Enfin, il y a une fonction de prédiction :

- Désabonnement : si le client s'est désabonné ou non (oui ou non)

Ces fonctionnalités peuvent également être subdivisées en :

- Informations démographiques sur les clients :
  - sexe, citoyen senior , partenaire , personnes à charge
- Services auxquels chaque client s'est inscrit :
  - Phone Service, MultipleLines , InternetService , OnlineSecurity , OnlineBackup , DeviceProtection , TechSupport , StreamingTV , StreamingMovies,
- Informations sur le compte client :
  - Mandat, contrat, facturation sans papier, méthode de paiement, charges mensuelles, charges totales

2) Environnement de développement

Pour la réalisation de ce projet, j'ai utilisé le système d'exploitation linux plus précisément la distribution Ubuntu 18.04. J'ai créé un environnement virtuel nommé project_venv dans lequel, j'ai installé l'ensemble des paquets contenu dans le fichier requirements.txt situé à la racine du répertoire Project.

->mkdir project

```
->cd project
-> python3 -m venv project_venv
-> source env/bin/activate
```

Pour installer les paquets faire :

->pip install -r requirements.txt

3) Le modèle

A la racine de notre projet nous avons deux fichiers sérialisés model.sav et model_nb.sav.

Model.sav est le modèle de regression logistic sérialisé avec la bibliothèque joblib

Model_nb.sav est le modèle gaussian NB.

A la racine nous avons un fichier **preprocessing** qui, comme son nom l'indique permet d'effectuer le prétraitement des données avant la prédiction.

4) API

Le but de notre projet était de déployer nos modèles de prédiction en production sans les ré-entrainer. Pour ce faire : j'ai sérialisé mes modèles v1 et v2 réalisés respectivement à partir des algorithme Gaussian NB et LogisticRegression

Pour la réalisation de ce projet, j'ai choisi FastAPI car elle dispose des avantages suivants :

- Très rapide : La performance est supérieure à Flask
- Rapide à coder : Créer des API 2 à 3 fois plus rapidement
- Facile : Facile à utiliser et à apprendre
- Annotation de Type (Type hints) : Facilite la validation, permet l'autocomplétion et facilite le debugging
- Documentation automatique : FastAPI génère la documentation en format Swagger UI et ReDoc automatiquement.

a. Description de l'API

Notre fichier api.py situé à la racine du projet, contient le code de l'api du modèle. Il est constitué de 2 classes :

La class Item : elle permet de récupérer les paramètres à envoyer aux modèles afin d'effectuer la prédiction.

La class Models : récupère-le model(v1/v2) dont on souhaite afficher les performances.

Nous avons cinq entpoints :

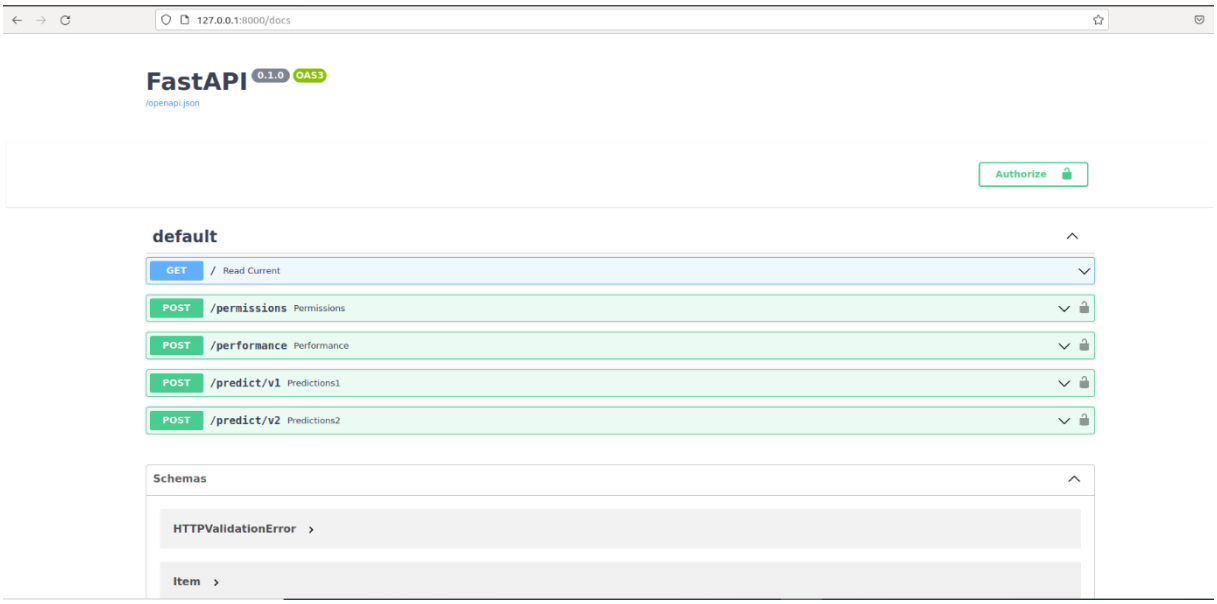
/ : qui permet de tester si l'API est fonctionnelle

/permissions : qui vérifie les permissions des utilisateurs

/performance : qui renvoie les performances de chaque modèle

/predict/v1 : renvoie les prédictions en s'appuyant le modèle GAUSSIAN NB

/predict/v2 : renvoie les prédictions en s'appuyant sur le modèle LogisticRegression



5) Test API

Le dossier test situé à la racine du projet contient un fichier test.py qui permet de vérifier que :

- L'api est fonctionnelle
- Les versions v1 et v2 du modèle sont fonctionnelle et retourne la prédiction et la probabilité maximal de la prédiction
- Les performances du modèle sont accessibles

Ensuite le dossier contient un fichier Dockerfile qui permet de construire l'image qui sera utilisée pour déployer le conteneur de test.

Enfin un répertoire permissions contenant un fichier Dockerfile et un fichier authentication.py qui permet de tester la connexion d'un utilisateur à l'api. Le dockerfile permet de construire l'image qui sera utilisé dans le troisième conteneur du docker-compose.yml

a. Les commandes :

Pour construire nos différentes images j'ai utilisé les commandes :

Docker build -t churn:latest .

Docker build -t test:latest .

Docker build -t authentication:latest .

6) Docker-compose

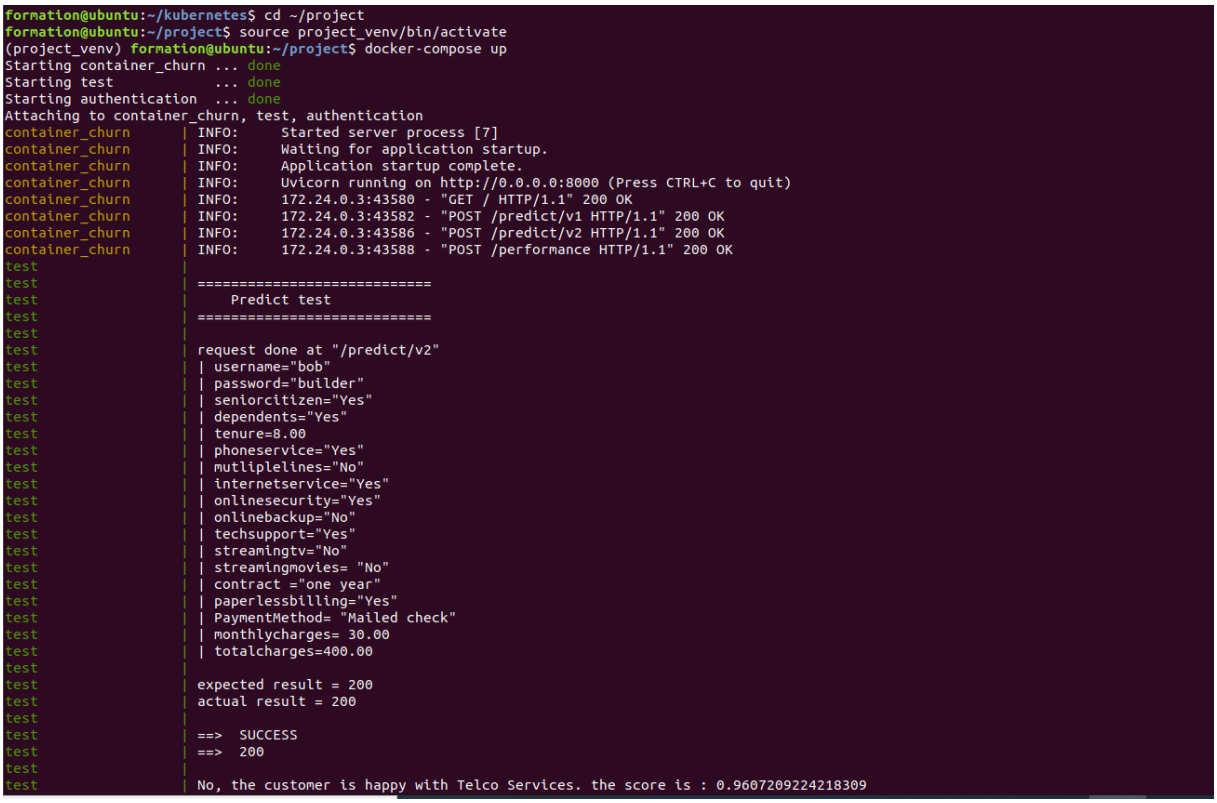
Le fichier docker-compose.yml se trouve à la racine du dossier project et permet de déployer plusieurs conteneurs au même moment notamment le conteneur de test, d'api, et de permissions.

Container_churn : il permet de déployer notre api et s'exécute en premier

Les conteneurs test et authentication dépendent de container_churn et s'exécutent 5 secondes après l'exécution du premier conteneur.

Pour lancer le conteneur faire :

-> docker-compose up



7) Kubernetes

Les fichiers Kubernetes sont stockés dans le sous-répertoire /k8s/

1) Deployment

a) Présentation

Un fichier deployment.yml est créé. Il permet le lancement du conteneur de l'API qui répondra aux requêtes sur le port 8000 du cluster. Ce conteneur sera répliqué 3 fois. Le conteneur est chargé depuis l'image stockée en ligne dans Docker Hub.

b) Commandes

kubectl create -f deployment.yml :crée un deployment

kubectl get deployment : affiche la liste des deployments

kubect delete deployments my-deployment-eval

Source :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment-eval
  labels:
    app: my-api-eval
spec:
  replicas: 3
  selector:
    matchLabels:
      my_label: container-api
  template:
    metadata:
      labels:
        my_label: container-api
    spec:
      containers:
      - name: my-api-eval
        image: lekeufackalida/churn:latest
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 8000

```

2) Service

a) Présentation

Un fichier service.yml est créé afin de rendre disponible les 3 répliques sur le port 8000 qui sera mappé au port 8000 défini pour le conteneur dans le Deployment.

b) Commandes

->docker push lekeufackalida/churn:latest : push l'image de notre api sur dockerhub

Démarrer kubernetes :

minikube start

minikube dashboard --url=true

puis cd k8s

->kubectl create -f service.yml : crée un service

->kubectl get service : affiche la liste des services

->kubect delete service my-service-eval

c) Source

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service-eval
  labels:
    app: my-api-eval
spec:
  type: ClusterIP
  ports:
  - port: 8002
    protocol: TCP
    targetPort: 8000
  selector:
    app: my-api-eval
```

3) Ingress

a) Présentation

Un fichier ingress.yml est créé afin de rendre le service disponible depuis l'extérieur.

b) Source

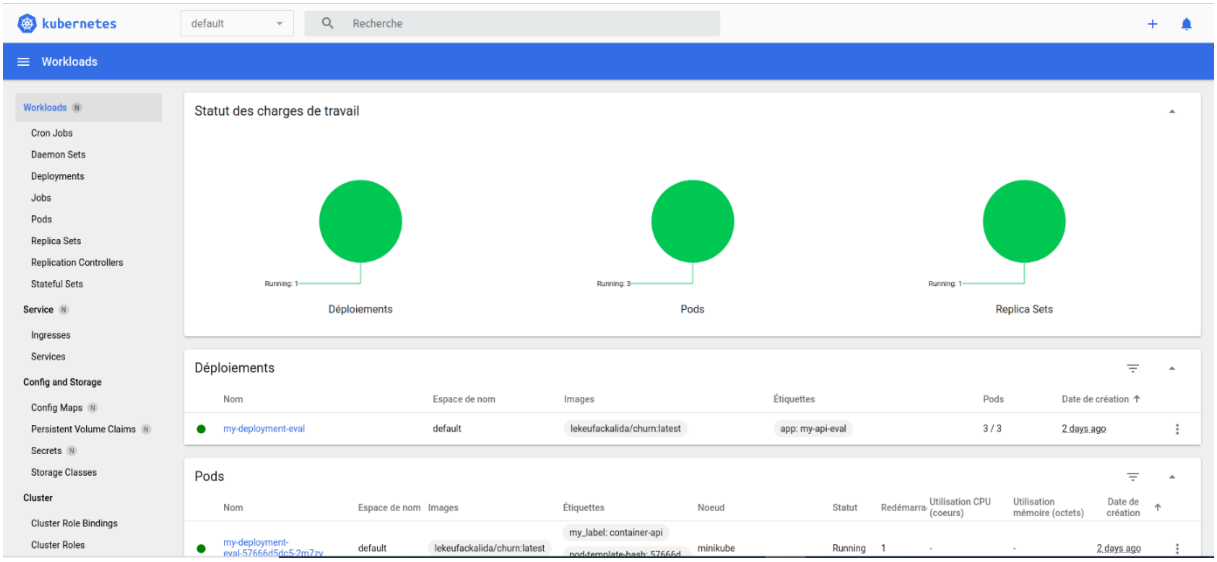
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
spec:
  defaultBackend:
    service:
      name: service
      port:
        number: 8002
```

c) Commandes

kubectl create -f ingress.yml

kubectl get ingress

kubect delete ingress ingress



Lancement

1. Accès au à cluster kubernates

->minikube start

->minikube dashboard --url=true

http://127.0.0.1:38463/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/

2. regarder la documentation de l'api(navigateur)

->http://127.0.0.1:8000/docs

3. lien du repositorie

-> https://github.com/lekeufackalida/final_project

Conclusion

L'objectif de ce travail était de déployer notre modèle de machine learning sur un cluster kubernetes tout en réalisant des tests d'intégration en utilisant docker. Au vu de ce qui précède, je peux conclure que cet objectif a été atteint. En plus, ce travail m'a permis d'améliorer ma compréhension et mes pratiques sur docker et kubernetes. En perspective je pense continuer ce travail afin de déployer mon modèle sur le cloud.