Création d'une architecture micro-services contenant un algorithme de Machine learning (Prédiction des prix des maisons) avec Docker & Flask

Réalisé par:

- Alycia KARA
- Amel Loualia

Encadré par:

- Mr. Manad Otman

I. Objectif de l'apprentissage:

L'objectif de notre application est de prédire la somme d'une maison selon certaines caractéristiques que l'utilisateur aurait saisies au préalable.
Donc l'algorithme est capable de prédire ces prix en se basant sur les attributs suivants saisis au niveau de l'interface graphique :

- bedrooms : représente le nombre de chambres de nuit.
- bathrooms : nombre de salles de bain.
- sqft_living : superficie de la maison.
- sqft_lot : superficie du terrain.
- floors : nombre d'étages.
- waterfront : maison au bord de mer ou non.
- view : S'il s'agit d'une belle vue (vue sur la mer ou montagnes ...etc).
- zipcode : le code postal pour juger la localisation.

	A	B	C	D	E	F	G	H	I
1	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	zipcode	price
2	3	1	1180	5650	1	0	0	98178	221900
3	3	2.25	2570	7242	2	0	0	98125	538000
4	2	1	770	10000	1	0	0	98028	180000
5	4	3	1960	5000	1	0	0	98136	604000
6	3	2	1680	8080	1	0	0	98074	510000
7	4	4.5	5420	101930	1	0	0	98053	1.225e+006
8	3	2.25	1715	6819	2	0	0	98003	257500
9	3	1.5	1060	9711	1	0	0	98198	291850
10	3	1	1780	7470	1	0	0	98146	229500
11	3	2.5	1890	6560	2	0	0	98038	323000
12	3	2.5	3560	9796	1	0	0	98007	662500
13	2	1	1160	6000	1	0	0	98115	468000

II. Structure des données E/S:

Comme tous les modèles de machine learning, afin de pouvoir fournir un résultat l'algorithme passe par une étape d'apprentissage. Ce qui fait que les données sont réparties en données train et test.
Les données d'entrée sont de type numérique et contiennent les attributs représentés récemment et en ce qui concerne les données d'output, il s'agit également d'une donné numérique qui est le prix prédit pour la maison (en dollars).

III. Technique de machine learning adoptée :

L'algorithme utilisé pour la prédiction est GradientBoostingRegressor. C'est un algorithme d'apprentissage supervisé dont le principe est de combiner les résultats d'un ensemble de modèles plus simple et plus faibles afin de fournir une meilleur prédiction. On parle d'ailleurs de méthode d'agrégation de modèles. Il fait partie des algorithmes les plus populaires pour ce genre de prédiction.

IV. Fonctionnement de l'application :

Le fonctionnement de notre application se fait par ordre dans les étapes suivantes :

1. Une interface web graphique est affichée à l'adresse : 127.0.0.1:5000, le user doit rentrer les informations demandées et cliquer sur le bouton predict.
2. Avec Flask, les données de l'interface sont récupérées comme input au modèle de machine learning.
3. L'algorithme fait ses calculs et produit une prédiction.
4. Le résultat est affiché au niveau de la même interface Web.

Pour ce faire, nous avons créé de conteneurs :

- algo_container : qui contient les données ainsi que l'algorithme de machine learning.
- ui_container : qui contient le programme Python qui utilise Flask ainsi que l'index.html pour afficher les résultats récupérés dans l'interface web.

Nous allons détailler le fonctionnement de chaque conteneur.

Exercice 1 :

1) **algo_container:**

Ce conteneur contient les données (fichier Csv) , model.py: qui est notre algorithme de prédiction, Requirements.txt (représente les dépendances requises pour notre programme, nécessaire pour le build de l'image), ainsi qu'un Dockerfile qui contient toutes les commandes nécessaires pour lancer le conteneur.

model.py

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.model_selection import train_test_split
import pickle
import os

def model_prices():
    data = pd.read_csv("data.csv")
    labels = data['price']
    train1 = data.drop(['price'],axis=1)
    x_train, x_test, y_train, y_test = train_test_split(train1, labels, test_size=0.10,       random_state=2)
    clf = GradientBoostingRegressor(n_estimators=400, max_depth=5, min_samples_split=2,
          learning_rate=0.1, loss='ls')
    clf.fit(x_train, y_train)
    download_dir = os.environ.get('DOWNLOAD_DIR', '.')
    file_pkl = os.path.join(download_dir, 'model.pkl')
    pickle.dump(clf, open(file_pkl,'wb'))

if __name__ == '__main__':
    model_prices()
```

Requirements.txt

```
pandas
numpy
scikit-learn
pickle4
```

Dockerfile

```
FROM ubuntu:latest
COPY . /app
WORKDIR /app
RUN apt-get update
RUN set -xe && apt-get update && apt-get install -y python3-pip
RUN pip3 install --upgrade pip
RUN pip3 install -r Requirements.txt
CMD python3 model.py
```

Pour lancer ce conteneur tout seul il suffit de se placer dans /projet_docker/model et lancer les commandes suivantes:

1. sudo docker build -t algo_container .
2. Sudo docker run -i -e DOWNLOAD_DIR=/data -v $HOME/Bureau/projet_docker/model:/data -d algo_container.
   PS:HOME/{}/projet_docker/model: faudrait mettre à la place des accolades le dossier ou le chemin vers le projet décompressé comme c'est le cas dans la commande ci dessus.

```
amel@amel:~/Bureau/projet_docker/model$ sudo docker build -t algo_container .
Sending build context to Docker daemon  2.298MB
Step 1/8 : FROM ubuntu:latest
 ---> f643c72bc252
Step 2/8 : RUN apt-get update
 ---> Using cache
 ---> 0affbea7ee9f
Step 3/8 : RUN set -xe && apt-get update && apt-get install -y python3-pip
 ---> Using cache
 ---> 654a028b4bdc
Step 4/8 : RUN pip3 install --upgrade pip
 ---> Using cache
 ---> 65550a6d7e28
Step 5/8 : COPY . /app
 ---> Using cache
 ---> 6be23df30159
Step 6/8 : WORKDIR /app
 ---> Using cache
 ---> 8e5c3663775f
Step 7/8 : RUN pip3 install -r Requirements.txt
 ---> Running in 9ce6cb4adf5b
Collecting numpy
  Downloading numpy-1.19.5-cp38-cp38-manylinux2010_x86_64.whl (14.9 MB)
Collecting pandas
  Downloading pandas-1.2.0-cp38-cp38-manylinux1_x86_64.whl (9.7 MB)
Collecting python-dateutil>=2.7.3
  Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
Collecting pytz>=2017.3
  Downloading pytz-2020.5-py2.py3-none-any.whl (510 kB)
Collecting six>=1.5
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting pickle4
  Downloading pickle4-0.0.1.tar.gz (19 kB)
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (from pickle4->-r Requirements.txt (line 4)) (45.2.0)
Collecting scikit-learn
  Downloading scikit_learn-0.24.0-cp38-cp38-manylinux2010_x86_64.whl (24.9 MB)
Collecting joblib>=0.11
  Downloading joblib-1.0.0-py3-none-any.whl (302 kB)
```

```
  Downloading pickle4-0.0.1.tar.gz (19 kB)
Requirement already satisfied: setuptools in /usr/lib/pyth
Collecting scikit-learn
  Downloading scikit_learn-0.24.0-cp38-cp38-manylinux2010_
Collecting joblib>=0.11
  Downloading joblib-1.0.0-py3-none-any.whl (302 kB)
Collecting scipy>=0.19.1
  Downloading scipy-1.6.0-cp38-cp38-manylinux1_x86_64.whl
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.1.0-py3-none-any.whl (12 kB)
Building wheels for collected packages: pickle4
  Building wheel for pickle4 (setup.py): started
  Building wheel for pickle4 (setup.py): finished with sta
  Created wheel for pickle4: filename=pickle4-0.0.1-py2.py
a92cee57919d2
  Stored in directory: /root/.cache/pip/wheels/be/be/f4/bc
Successfully built pickle4
Installing collected packages: six, numpy, threadpoolctl,
Successfully installed joblib-1.0.0 numpy-1.19.5 pandas-1.
six-1.15.0 threadpoolctl-2.1.0
Removing intermediate container 9ce6cb4adf5b
 ---> 2a75a2f1f18e
Step 8/8 : CMD python3 model.py
 ---> Running in 40e163168204
Removing intermediate container 40e163168204
 ---> 7bf069e255e2
Successfully built 7bf069e255e2
Successfully tagged algo_container:latest
```

L'output est le fichier 'model.pkl' qui se trouve dans le dossier 'model'qui contient le modèle de notre algorithme d'apprentissage.Ce fichier est donc généré une fois que l'image est lancée.

Exercice 2 :

2) **ui_container :**

Ce conteneur contient le fichier app.py : qui représente notre programme où on utilise Flask qui sert à récupérer les données prédites et créer une route afin d'afficher les résultats de l'algorithme dans une interface Web, il utilise derrière le fichier index.html où nous spécifions le conteneur de la page Web (Template).
Il contient également un Dockerfile ainsi qu'un requirements.txt .

app.py

```python
from flask import Flask, request
from flask import render_template, make_response
import os
import numpy as np
import pickle
import os


APP = Flask(__name__)

@APP.route('/')
def index():
    return render_template('indexx.html')

@APP.route('/predict',methods=['POST'])
def predict():

    int_features = [float(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    model = pickle.load(open('model.pkl', 'rb'))
    prediction = model.predict(final_features)
    output = round(prediction[0], 2)
    return render_template('indexx.html', prediction_text='The price for this house is :{} $'.format(output))

if __name__ == '__main__':

    APP.run(host='0.0.0.0',debug=True)
```

indexx.html

```html
<!DOCTYPE html>
<html>
<body>

<h1>Predicting Home Pricing by its attributes</h1>

<form method="POST" action="{{ url_for('.predict')}}" style="background-color:#CCCCCC">
    <pre>
    bedrooms <input name="bedrooms"  type=number  required /><br>
    bathrooms  <input name="bethrooms" type=number   required/><br>
    sqft_living <input name="sqft_living" type=number  required/><br>
    sqft_lot  <input name="sqft_lot" type=number   required/><br>
    floors  <input name="floors" type=number   required/><br>
    water_front  <input name="water_front" type=number   required/><br>
    view  <input name="view" type=number   required/><br>
    zip_code  <input name="zip_codeh" type=number   required/><br>
    <button type="submit" class="btn">Predict</button>
    </pre>
</form >

<br>
    <br>
    {{ prediction_text}}

</body>
</html>
```

Dockerfile

```
FROM ubuntu:latest
COPY .  /app
WORKDIR /app
RUN apt-get update
RUN set -xe && apt-get update && apt-get install -y python3-pip
RUN pip3 install --upgrade pip
RUN pip3 install -r requirements.txt
EXPOSE 5000
CMD  python3 app.py
```

requirements.txt

```
Flask==0.10.1
numpy
pickle4
scikit-learn
```

Afin de lancer ce conteneur et qui derrière lancera algo_container vu qu'il dépend de son résultat, il suffit de se placer dans /projet_docker/app et lancer les commandes suivantes :

1. sudo docker build -t ui_container .
2. sudo docker run -i -p 5000:5000 -d ui_container

Pour accéder à l'interface il faut aller sur le lien suivant :
0.0.0.0:5000

```yaml
version: "3.2"

services:
  #container_name: user_interface

  model:
    #container_name: model

    build: ./model/.
    volumes:
      - ./model:/app
    stdin_open: true
    tty: true
  web:
    build: ./app/.
    volumes:
      - ./app:/app

    ports:
      - "5000:5000"
```

Exercice 4 :

Le répertoire **bonus** contient le fichier **docker-compose.yml :**
qui les deux service model et web (les deux conteneurs).
Le service 'model' génère le modèle d'apprentissage et la prédiction du prix puis le service 'web' récupère les résultats de la prédiction et l'affiche sur son interface.
Pour lancer le docker compose : Il suffit de :

1. Se positionner dans le répertoire bonus.
2. Ouvrir le terminal.
3. Lancer la commande docker-compose up
4. Une fois lancé, aller à cette adresse http://127.0.0.1:5000 dans le navigateur.

```
alycia@alycia-VivoBook-S14-X430UA:~/Musique/projet_docker_finale(1)/projet_docker/Docker-compose$ sudo docker-compose up
Building web
Step 1/9 : FROM ubuntu:latest
 ---> f643c72bc252
Step 2/9 : COPY .  /app
 ---> cd29863f07d2
Step 3/9 : WORKDIR /app
 ---> Running in f1bfcd105c2a
Removing intermediate container f1bfcd105c2a
 ---> a327bc1013f6
Step 4/9 : RUN apt-get update
 ---> Running in b6ea5bab8828
```

```
Successfully built 57aabf585f7d
Successfully tagged docker-compose_web:latest
WARNING: Image for service web was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Creating docker-compose_web_1   ... done
Creating docker-compose_model_1 ... done
Attaching to docker-compose_model_1, docker-compose_web_1
web_1    |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1    |  * Restarting with stat
web_1    |  * Debugger is active!
web_1    |  * Debugger PIN: 144-386-823
docker-compose_model_1 exited with code 0
web_1    | 172.21.0.1 - - [12/Jan/2021 15:33:37] "GET / HTTP/1.1" 200 -
web_1    | 172.21.0.1 - - [12/Jan/2021 15:33:37] "GET /favicon.ico HTTP/1.1" 404 -
web_1    | 172.21.0.1 - - [12/Jan/2021 15:34:22] "POST /predict HTTP/1.1" 200 -
web_1    |  * Detected change in '/app/app.py', reloading
web_1    |  * Restarting with stat
web_1    |  * Debugger is active!
web_1    |  * Debugger PIN: 144-386-823
web_1    | 172.21.0.1 - - [12/Jan/2021 16:21:23] "GET / HTTP/1.1" 200 -
```

Exercice 3 :

Dans cet exercice, nous avons créé un Vagrantfile dans le but d'isoler l'environnement de développement, ci-dessous notre Vagrantfile, mais en lançant vagrant up, nous obtenons des erreurs liées à la connexion ssh que nous n'avons pas pu résolu.
Cependant, nous avons effectué la partie concernant Dockercompose.

```
Vagrant.configure(2) do |config|
  config.vm.box = 'ubuntu/trusty64'
  config.ssh.insert_key = false
  config.vm.define "projetf"
  config.vm.network "forwarded_port", guest: 5000, host: 5000, host_ip: "127.0.0.1"
  config.vm.provider "virtualbox" do |vb|
    vb.name = "projetf"
    vb.memory = 4096
    vb.cpus = 1
  end
  config.vm.synced_folder ".", "/vagrant", type: "rsync"
  config.vm.provision :docker
  config.vm.provision :docker_compose, yml: "/vagrant/docker-compose.yml", run:"always"

end
```

Résultat final de l'application: