Machine Learning Model App Deployment Using Google Cloud Platform

Igwebuike Eze

06/05/2024

1 Introduction

This document describes the deployment of a Flask-based web/app application for a machine learning model on Google Cloud Platform. This project has eight major code snipets:

- main.py - This contains Machine Learning model code to predict employee salaries based on trainig data in 'HRRecruitmentData.csv' file.
- app.py - This contains flask Web/API that receives employee details through API calls, computes the predited value based on our model and returns it.
- request.py - This uses requests module to call APIs already defined in app.py and dispalys the returned value.
- idex.html file is used to design the output of the form.
- style.css file is used to apply aesthetics to the form.
- requirements.txt is the compilation of all the modules and their version used for the project.
- Dockerfile is a text document that contains all the commands used to call on the command line to assemble an image.
- app.yaml is the serialised version of app.py.

The following sections provide a detailed step-by-step guide.

2 Step 1: Select a Toy Dataset

For this deployment, we used a simple toy dataset for regression analysis. Below is an example of the dataset used:

EXPERIENCE	ASSESSMENT_SCORE	INTERVIEW_SCORE	EXPECTED_SALARY
one	8	9	40000
two	8	6	50000
three	6	7	60000
four	10	9	70000
five	9	6	80000
six	7	8	60000
seven	5	7	80000
eight	7	8	75000
nine	6	6	45000
ten	5	7	40000
eleven	5	8	80000
twelve	6	6	75000

3 Step 2: Train and Save the Model

I trained a simple linear regression model and saved it using the Python 'pickle' module. The code snippet below demonstrates how the model was trained and saved:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import pickle

# Importing the libraries
import pandas as pd
import pickle

dataset = pd.read_csv('HR_Recruitment_Data.csv')


X = dataset.iloc[:, :3]

#Converting words to integer values
def convert_to_int(word):
    word_dict = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5, 'six':6,
        'seven':7, 'eight':8,
                'nine':9, 'ten':10, 'eleven':11, 'twelve':12, 'zero':0}
    return word_dict[word]

X['experience'] = X['experience'].apply(lambda x : convert_to_int(x))

y = dataset.iloc[:, -1]

#Splitting Training and Test Set
#Since we have a very small dataset, we will train our model with all
   availabe data.

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

#Fitting model with trainig data
regressor.fit(X, y)

# Saving model to disk
pickle.dump(regressor, open('model.pkl','wb'))

# Loading model to compare the results
HR_Recruitment = pickle.load(open('model.pkl','rb'))
```

4 Step 3: Deploy the Model on Google Cloud Platform

Next, I created a simple Flask application to deploy the model on GCP. The code snippet below shows the main structure of the Flask app:

```
from flask import Flask, request, jsonify, render_template
import numpy as np
import pickle
```

```
app = Flask(__name__)

# Load pre-trained model
HR_Recruitment = pickle.load(open('model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = HR_Recruitment.predict(final_features)

    output = round(prediction[0], 2)
    return render_template('index.html', prediction_text=f'Employee␣
       Starting␣Salary␣Forecast:␣$␣{output}')

@app.route('/predict_api', methods=['POST'])
def predict_api():
    data = request.get_json(force=True)
    prediction = HR_Recruitment.predict([np.array(list(data.values()))])

    output = prediction[0]
    return jsonify(output)

# Required for Google Cloud Functions
def predict_http(request):
    with app.test_request_context():
        return app.full_dispatch_request()

if __name__ == "__main__":
    app.run(host ='0.0.0.0', port = 5000, debug=True)
```

5 Step 4: Containerizing the Flask App

I created a Dockerfile in the root directory of my Flask app. Then define the base image and copy my application code into the container. I install dependencies and expose the port(5000) on which my Flask app runs. The code snippet below demonstrates how the form was designed:

```
# Use the official Python image as a base image
FROM python:3.12-slim

# Install development tools including GCC
RUN apt-get update && \
    apt-get install -y build-essential

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt .
```

```
# Install the dependencies
RUN pip install -r requirements.txt

# Copy the rest of the application code into the container
COPY . .

# Specify the port number the container should expose
EXPOSE 5000

# Command to run the application
CMD ["python", "app.py"]
```

6 Step 5: Building and Testing the Docker Image

I built the Docker image using the docker build command.
docker build -t my-flask-app .
Then tested the Docker image locally to ensure it runs as expected.
docker run -p 8080:5000 my-flask-app

7 Step 6: Pushing the Docker Image to Google Container Registry (GCR)

I tag the Docker image with my GCR repository's URL.
docker tag my-flask-app gcr.io/salary-prediction-422622/my-flask-app
I authenticated Docker to push images to GCR.
gcloud auth configure-docker
I then pushed the Docker image to GCR.
docker push gcr.io/salary-prediction-422622/my-flask-app

8 Step 7: Deploying the Flask App on Google Cloud Run

I navigated to the Google Cloud Console and open the Cloud Run service. Then clicked on Create Service and specify the details like the region, container image, and other configurations.

Click on "Deploy" to deploy my Flask app to Cloud Run. Once deployed, I received a URL where my Flask app is accessible

I then navigated to URL https://my-flask-app-owftpovrwq-ew.a.run.app to view the homepage.
Enter valid numerical values in all 3 input boxes and hit Predict.