Practice activity: Preparing a model for deployment

coursera.org/learn/foundations-of-ai-and-machine-learning/supplement/ChkxN/practice-activity-preparing-a-model-for-deployment

Introduction

In this activity, you will apply the skills and knowledge you've gained to prepare an ML model for deployment. This exercise will give you hands-on experience with the steps involved in packaging, containerizing, and deploying a model in a production environment.

By the end of this lab, you will be able to:

- Take a trained model, prepare it for deployment, and ensure it's ready to serve predictions in a real-world setting.

Objective

- Package an ML model along with its dependencies.
- Containerize the model using Docker.
- Deploy the model locally, and test its performance.
- Optionally, deploy the model to a cloud environment for wider accessibility.

Prerequisites

Before starting the activity, ensure you have the following tools installed on your machine:

- **Python 3.6 or later**: for running scripts and packaging the model
- **Docker**: for containerizing the model
- **Git**: (optional) for version control

- **An IDE**: such as VS Code, PyCharm, or Jupyter Notebook

1. Packaging the model

Step-by-step guide:

Step 1: Save the trained model

1. Train a simple model

If you don’t have a trained model, you can use the following example to train a simple logistic regression model on the Iris dataset:

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import joblib

# Load the dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)

# Train the model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Save the model to a file
joblib.dump(model, 'iris_model.pkl')
```

2. Ensure the model is saved

Verify that the model file iris_model.pkl is saved in your working directory.

Step 2: Create a requirements.txt file

1. List the dependencies

Create a requirements.txt file to list all the Python packages your model depends on. This ensures that the same environment can be recreated in production.

Example requirements.txt:

4

flask==2.0.2

2. Save the file

Save the requirements.txt file in the same directory as your model.

Step 3: Create a Python script for serving the model

1. Create a serve_model.py script

This script will use Flask to serve the model as a RESTful API, allowing it to make predictions when called.

```
from flask import Flask, request, jsonify
import joblib
import numpy as np

app = Flask(__name__)

# Load the model
```

```
model = joblib.load('iris_model.pkl')


@app.route('/predict', methods=['POST'])

def predict():

    data = request.get_json(force=True)

    prediction = model.predict(np.array(data['input']).reshape(1, -1))

    return jsonify({'prediction': int(prediction[0])})


if __name__ == '__main__':

    app.run(host='0.0.0.0', port=80)
```

2. Test the script

Run the script using python serve_model.py, and use tools such as Postman or cURL to test the API locally by sending a POST request to http://127.0.0.1:80/predict with a JSON body, such as:

```
{
    “Input” : [5.1, 3.5, 1.4, 0.2]
}
```

2. Containerizing the model with Docker

Step-by-step guide

Step 1: Create a Dockerfile

1. Write a Dockerfile

The Dockerfile defines the environment in which your model will run, including the base image, dependencies, and startup command.

Example Dockerfile:

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
```

```
COPY . /app


# Install any needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt


# Make port 80 available to the world outside this container

EXPOSE 80


# Run serve_model.py when the container launches

CMD [python, serve_model.py]
```

2. Save the Dockerfile

Save it in the same directory as your model and scripts.

Step 2: Build the Docker image

1. Build the image

Open a terminal, navigate to your project directory, and build the Docker image using the following command:

```
docker build -t iris_model_image .
```

2. Verify the image

Once the build is complete, verify that the image was created by running docker images.

Step 3: Run the Docker container

1. Run the container

Run the Docker container from the image you just created:

```
docker run -d -p 80:80 iris_model_image
```

2. Test the deployment

Test the deployed model by sending a POST request to http://localhost:80/predict using the same JSON payload you used earlier.

3. Optional: Deploying to a cloud environment

If you want to deploy your Docker container to a cloud environment (such as Microsoft Azure, Amazon Web Services, or Google Cloud), follow the platform-specific instructions for deploying Docker containers. Here's a brief overview for deploying to Azure:

Step-by-step guide:

Step 1: Push the Docker image to Azure Container Registry

1. Log in to Azure

Log in to your Azure account using the Azure command-line interface:

2. Create a resource group and container registry

Create a resource group and container registry to store your Docker image:

```
az group create --name myResourceGroup --location eastus
az acr create --resource-group myResourceGroup --name myContainerRegistry --sku Basic
```

3. Push the image

Tag your Docker image, and push it to the Azure Container Registry:

```
docker tag iris_model_image myContainerRegistry.azurecr.io/iris_model_image:v1
docker push myContainerRegistry.azurecr.io/iris_model_image:v1
```

Step 2: Deploy the container to Azure Container instances

1. Deploy the container

- Deploy the container directly from your Azure Container Registry:

```
az container create --resource-group myResourceGroup --name irisModelContainer --image myContainerRegistry.azurecr.io/iris_model_image:v1 --dns-name-label irismodel --ports 80
```

2. Test the deployed model

Once deployed, you can test the model using the public IP address or DNS label provided by Azure.

4. Deliverables

Upon completing this activity, you should have your packaged model, Dockerfile, and any scripts you used. If you deployed the model to the cloud, provide the URL to access it.

Reflection: Write a brief reflection on the challenges you encountered and how you overcame them during the lab.

Conclusion

This activity provides you with hands-on experience in preparing an ML model for deployment. By packaging, containerizing, and deploying the model, you gain a deeper understanding of the steps required to transition from model development to production. These skills are crucial for ensuring that your models can be effectively deployed and scaled in real-world environments.

Hi, Rodrigo!

How can I help?