Deploy Machine Learning Models using FASTAPI: A step by step walkthrough | by Nirmalya Misra
Deploying ML Models as API using FastAPI - GeeksforGeeks
Creating a Machine Learning App using FastAPI and Deploying it Using Kubernetes | Engineering Education (EngEd) Program | Section

FastAPI and FLASK

Introduction

These are basically used to deploy ML models. Using FLASK, we create a web app which can host our ML model and make predictions. FLASK is simplistic, minimal and good for smaller projects. Fastapi primarily focuses on performance.

FLASK code breakdown

```
from flask import Flask, render_template, request
import pickle
import numpy as np
from tensorflow.keras.models import load_model

#model = pickle.load(open('model.pkl', 'rb'))
model = load_model('model.h5')
```

The following libraries were imported. Load_model from keras was used to import the model instead of pickle. The neural network model was breaking when pickled.

```
app = Flask(__name__)
```

This is used to initialize the app.

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

@app.route('/predict', methods=['POST'])
def home():
    data1 = request.form['a']
    data2 = request.form['b']
```

```
    arr = np.array([[int(data1), int(data2)]])
    pred = model.predict(arr)
    return render_template('after_1.html', data=pred)
```

@app.route decorator is used to define the routes or URLs that the application responds to. It maps a URL endpoint to a Python function, which will be executed when a user accesses that specific URL.
@app.route('/') specifies that the function man() should be executed when the root URL (i.e., '/') is accessed. The render_template() function is called to render the HTML template 'home.html', which will be returned as the response to the user's request.
@app.route('/predict', methods=['POST']) indicates that the function home() should be executed when the URL '/predict' is accessed with the HTTP method POST. This corresponds to the form submission action in the HTML template.
Inside the home() function, the values submitted via the form are retrieved using request.form. These values are then used to create a NumPy array and pass it to the machine learning model for prediction. Finally, the prediction result is sent to the 'after_1.html' template for rendering.

To run the code,

```
python app.py
```

How to Use

Homepage

- Open the web application in your browser.
- You will see a form titled "SWITCH DETECTION" with two input fields: "Value for Force" and "Value for Voltage".
- Enter the corresponding values for force and voltage in the input fields.
- Click the "Predict!" button to submit the values.

Prediction Result

- After clicking the "Predict!" button, the web application will process the input values using the machine learning model.
- The predicted result will be displayed on a new page titled "Prediction Result".
- The prediction result represents the classification output based on the input values.

Fastapi code breakdown

```
# Import FastAPI and other modules
from fastapi import FastAPI, Request
from fastapi.templating import Jinja2Templates
from fastapi.responses import HTMLResponse
import tensorflow as tf
import numpy as np

# Create an app instance
app = FastAPI()
```

Following libraries are imported and an app instance is created as shown.

```
# Create a Jinja2 template instance
templates = Jinja2Templates(directory="templates")
# Load the ML model
model = tf.keras.models.load_model("model.h5")
```

The model is loaded using the keras library and jinja2templates is used to access the html template directory for frontend.

```
# Define the home page route
@app.get("/", response_class=HTMLResponse)
def home(request: Request):
    # Render the home page template
    return templates.TemplateResponse("home.html", {"request": request})
```

This codeblock is used to define the homepage route using @app.get decorator. This resembles app.route(“/”) in FLASK. It further specifies that the response should be HTML.

The function inside this decorator is executed when this route is accessed. The home() function renders the "home.html" template using TemplateResponse from the templates instance.

```
# Define the predict route
@app.post("/predict")
async def predict(request: Request):
    # Get the input values from the form
    form_data = await request.form()
    a = float(form_data["a"])
```

```
    b = float(form_data["b"])

    # Reshape the input values into a numpy array
    input_data = np.array([[a, b]])

    # Make a prediction using the ML model
    prediction = model.predict(input_data)
    data = prediction

    # Render the result page template with the prediction value
    return templates.TemplateResponse("after_1.html", {"request": request,
"data": data})
```

@app.post("/predict") indicates that this function should be executed when the "/predict" URL is accessed with a POST request. Inside the predict() function, the input values are retrieved using using **await request.form()**. The input values are reshaped into a NumPy array and used for prediction using the ML model.

The "after_1.html" template renders the result page with the prediction value.

To run the app

```
uvicorn fastapiapp:app
```

How to Use

Homepage

- Open the web application in your browser.
- You will see a form titled "SWITCH DETECTION" with two input fields: "Value for Force" and "Value for Voltage".
- Enter the corresponding values for force and voltage in the input fields.
- Click the "Predict!" button to submit the values.

Prediction Result

- After clicking the "Predict!" button, the web application will process the input values using the machine learning model.
- The predicted result will be displayed on a new page titled "Prediction Result".
- The prediction result represents the classification output based on the input values.