Week 4 Flask Deployment

Title: Deployment Documentation

Name: Sanjana Naidu Gedela

Batch Code: LISUM30

Submission Date:2/28/2024

Table of Contents:

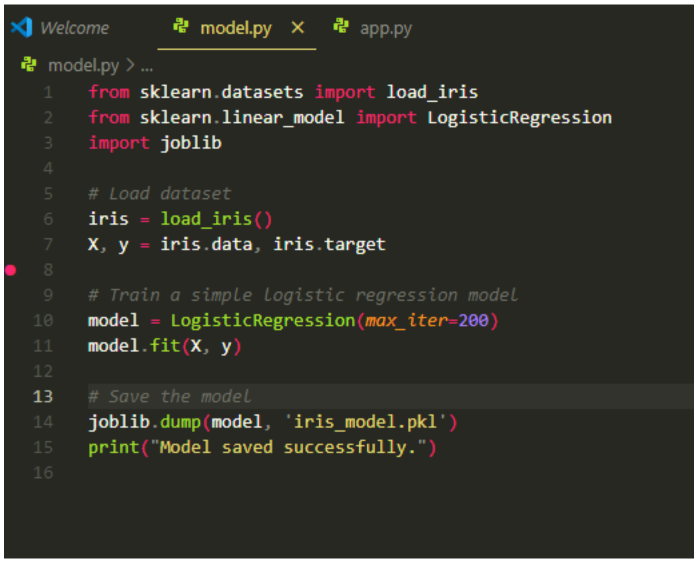
Introduction:

In this document, we detail the process of deploying a machine learning model using the Flask web framework. We outline the steps taken to select a dataset, train a logistic regression model, serialize the model for use in a web application, and create a Flask app to serve predictions from the model. We also provide evidence of testing the Flask application.

Model Selection and Training

We chose the Iris dataset for its simplicity and because it is a well-known dataset within the machine learning community. The logistic regression model was selected due to its effectiveness in handling multi-class classification problems.

The following Python code was used to train the model:


Welcome model.py app.py

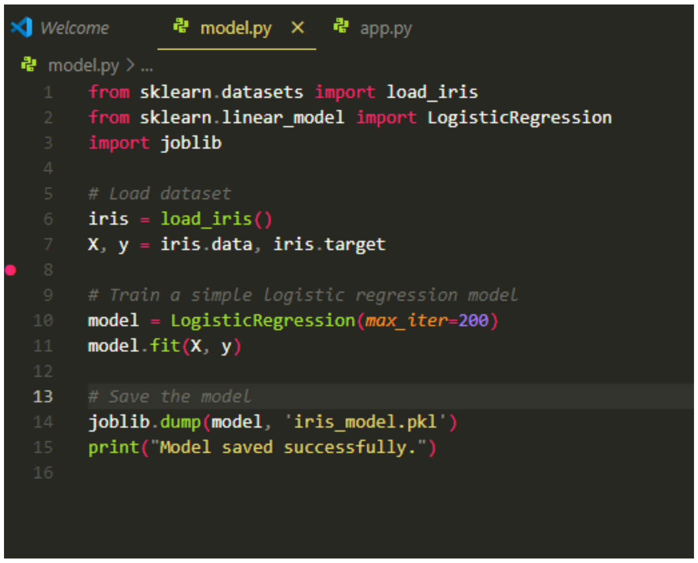
```python
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import joblib

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Train a simple logistic regression model
model = LogisticRegression(max_iter=200)
model.fit(X, y)

# Save the model
joblib.dump(model, 'iris_model.pkl')
print("Model saved successfully.")
```

Model Serialization:

The following Python code was used to train the model:

After training the model, we used joblib to serialize and save the model to disk. This allows us to load the trained model into our Flask application without retraining.

The code for serialization is as follows:


Welcome model.py app.py

```python
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import joblib

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Train a simple logistic regression model
model = LogisticRegression(max_iter=200)
model.fit(X, y)

# Save the model
joblib.dump(model, 'iris_model.pkl')
print("Model saved successfully.")
```

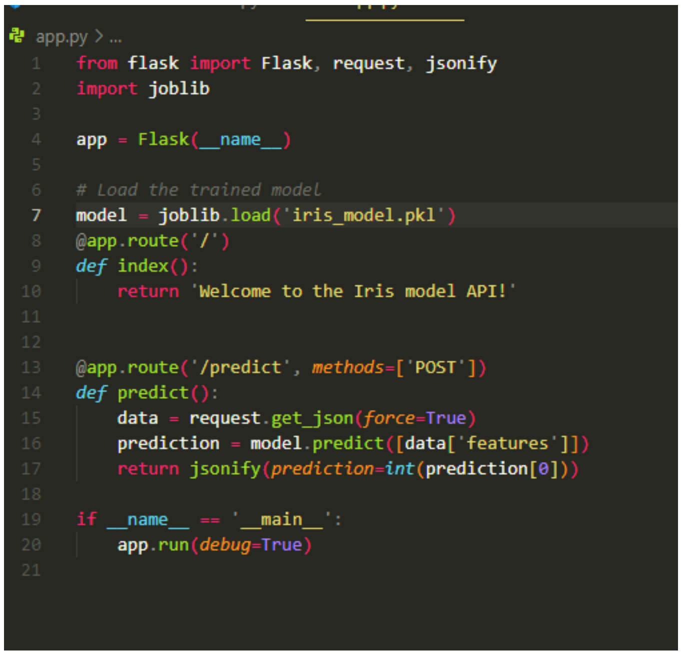
Flask Web Application Setup:

Flask is a micro web framework for Python, which is very suitable for small to medium web applications and APIs. We created a Flask application with the following features:

A route for predicting based on the model.

The ability to handle POST requests with input features in JSON format.

Here is the core Flask application code:



app.py > ...

```python
from flask import Flask, request, jsonify
import joblib

app = Flask(__name__)

# Load the trained model
model = joblib.load('iris_model.pkl')
@app.route('/')
def index():
    return 'Welcome to the Iris model API!'


@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    prediction = model.predict([data['features']])
    return jsonify(prediction=int(prediction[0]))

if __name__ == '__main__':
    app.run(debug=True)
```

Testing the Flask Application

To test our Flask application, we used the following curl command to simulate a client making a POST request:

```
curl -X POST -H "Content-Type: application/json" -d '{"features":[5.1, 3.5, 1.4, 0.2]}'
http://127.0.0.1:5000/predict
```

```
{
  "prediction": 0
}
```

Conclusion:

The deployment of our Flask application was successful. The application is capable of receiving input features via POST requests and responding with a prediction. This process demonstrates the viability of using Flask to deploy a machine learning model as a web service.