Model Deployment Report

Deployment of Random Forest Model with Flask

Name: Nonye Nweke

Batch Code: LISUM36

Submission Date: 2024-08-28

Submitted to: [Instructor Name]

Model Deployment Report

Step 1: Data Loading

```
from sklearn.datasets import load_iris
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pickle

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

# Convert to Pandas DataFrame
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Add the target column
iris_df['species'] = iris.target

# Get the shape of the data
print("Shape of the data (features):", iris.data.shape)
print("Shape of the target (labels):", iris.target.shape)

# View the first few rows of the dataset
print(iris_df.head())
```

```
Shape of the data (features): (150, 4)
Shape of the target (labels): (150,)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
```

Loading the Iris dataset using sklearn.

This step involves loading the Iris dataset, which contains 150 samples of iris flowers with 4 features each.

Model Deployment Report

Step 2: Model Training

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Training a Random Forest Classifier on the Iris dataset.

In this step, we train a Random Forest model using the training data. The model is trained with 100 decision trees.

Model Deployment Report

Step 3: Model Saving

```
# Save the model
with open('iris_model.pkl', 'wb') as file:
    pickle.dump(model, file)
```

Saving the trained model using pickle.

The trained Random Forest model is saved to a file named iris_model.pkl using Python's pickle module.

Model Deployment Report

Step 4: Flask Deployment

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

app = Flask(__name__)

# Load the saved model
with open('iris_model.pkl', 'rb') as file:
    model = pickle.load(file)

@app.route('/')
def home():
    return "Iris Classifier API"

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json['data']
    prediction = model.predict([np.array(data)])
    return jsonify({'prediction': int(prediction[0])})

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

```
 * Serving Flask app '__main__'
 * Debug mode: on
```

Creating a Flask web app to deploy the model.

Model Deployment Report

Step 5: Flask Deployment

```python
#creating app.py
from flask import Flask, request, jsonify, render_template
import pickle
import numpy as np

# Initialize the Flask app
app = Flask(__name__)

# Load the model
with open('iris_model.pkl', 'rb') as file:
    model = pickle.load(file)

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

@app.route('/predict', methods=['POST'])
def predict():
    # Get the data from the form
    features = [float(x) for x in request.form.values()]
    final_features = [np.array(features)]

    # Make prediction
    prediction = model.predict(final_features)
    output = prediction[0]

    # Return the result
    return render_template('index.html', prediction_text=f'The predicted class is: {output}')

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

Creating a Flask web app to deploy the model.

A simple flask web page is set up to load the model and provide predictions based on user inputs.

Model Deployment Report

Step 6: Web form

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Iris Prediction</title>
</head>
<body>
    <h1>Iris Flower Prediction</h1>
    <form action="{{ url_for('predict') }}" method="post">
        <label>Sepal Length:</label>
        <input type="text" name="sepal_length"><br>
        <label>Sepal Width:</label>
        <input type="text" name="sepal_width"><br>
        <label>Petal Length:</label>
        <input type="text" name="petal_length"><br>
        <label>Petal Width:</label>
        <input type="text" name="petal_width"><br>
        <button type="submit">Predict</button>
    </form>
    <h2>{{ prediction_text }}</h2>
</body>
</html>
```

Creating a simple web form.

An index.html file in a folder named templates is created in the same directory as the app.py.

Model Deployment Report

Step 7: Web form

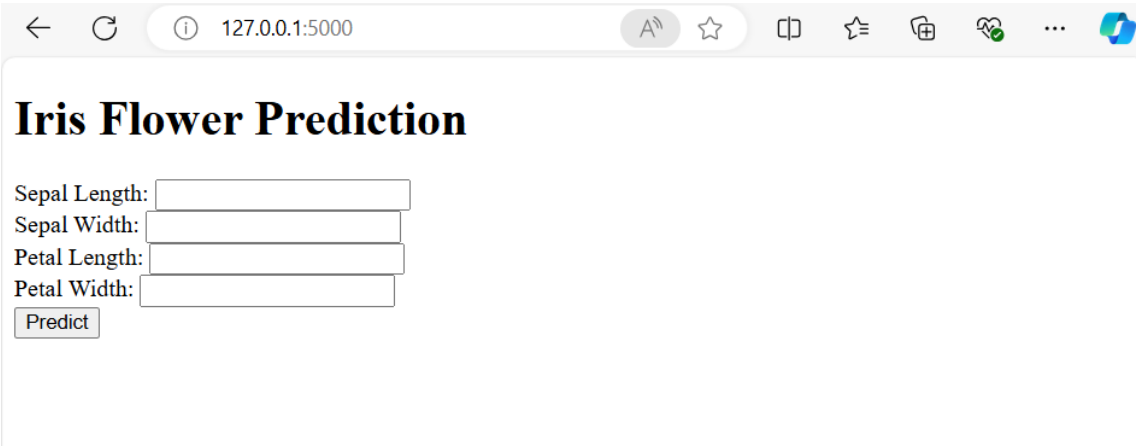
```python
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get the data from the form
        sepal_length = float(request.form['sepal_length'])
        sepal_width = float(request.form['sepal_width'])
        petal_length = float(request.form['petal_length'])
        petal_width = float(request.form['petal_width'])

        # Make a prediction
        prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])
        output = prediction[0]

        return render_template('index.html', prediction_text=f'The predicted iris species is {output}')

    except Exception as e:
        return render_template('index.html', prediction_text=f'Error: {e}')
```

Creating a simple web form.

an index.html file in a folder named templates is created in the same directory as the app.py.

Model Deployment Report

Step 8: Web form



viewing the web form.

the web form will be used to enter values for each variable and predict the flower specie.