Deployment on Flask

Name: Pargat Singh
Batch Code: LISUM39
Submission Date: 28/11/2024
Submitted To: Data Glacier

1. Introduction

Project Overview:
This project focuses on predicting the species of Iris flowers based on their physical measurements (sepal length, sepal width, petal length, and petal width). We used the **Iris dataset**, a famous dataset in machine learning, to train a **Random Forest Classifier** model. After training the model, we deployed it on a **Flask web application**, allowing users to input flower measurements and receive a prediction for the species.

Dataset:
The **Iris dataset** contains 150 rows of data, each representing a flower's measurements across four features:

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width

There are three possible species:

- **Iris-setosa**
- **Iris-versicolor**
- **Iris-virginica**

The goal of this project is to build a model that can predict the species of a flower based on these features.

2. Data Preprocessing

Steps Taken:

1. **Loading the Data**:
   The Iris dataset was loaded from a CSV file. After loading the data, we explored the first few rows to understand its structure.
2. **Feature Selection**:
   The features used to predict the species are:
   - Sepal Length
   - Sepal Width
   - Petal Length
   - Petal Width

   The target variable (what we want to predict) is the species of the flower.

3. **Data Splitting**:
   We split the dataset into two parts:
   - **Training Set** (70% of the data)
   - **Testing Set** (30% of the data)
     This split helps evaluate the model on unseen data to check its performance.
4. **Feature Scaling**:
   Since the features have different scales (e.g., petal length and sepal width), we used **StandardScaler** to standardize the data. This ensures the model performs optimally by making all features contribute equally.

```
WEEK4_DEPLOYMENT_ON_FLASK
  templates
    index.html
  app.py
  iris_model.pkl
  iris.data.csv
  train_model.py
```

```python
# train_model.py > ...
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import joblib

# Load the dataset from a CSV file
df = pd.read_csv("C:/Users/pargat/Desktop/B Data Science Internship/Data Glacier/Week 4/Week4_Deployment_on_Flask/iris.data.csv")

# Display the first few rows of the dataset to understand its structure
print(df.head())

# Select independent variables (features) and the dependent variable (target)
X = df[["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]]  # Features
y = df["Species"]  # Target variable (species)

# Split the dataset into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=50)

# Feature scaling: Standardizing the features (mean=0, std=1) for better model performance
sc = StandardScaler()
X_train = sc.fit_transform(X_train)  # Fit on training data and transform it
X_test = sc.transform(X_test)  # Transform the test data based on the training data scaling
```

3. Model Training

Model Choice:
We chose the **Random Forest Classifier** for this project due to its robustness and ability to handle a wide variety of data types. Random Forest creates multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.

Training the Model:
The model was trained on the training set using the `RandomForestClassifier` from **scikit-learn**. We then saved the trained model using **joblib** to avoid retraining it every time we run the application.

Model Saving:
After training, we saved the model in a file named `iris_model.pkl`, which is later loaded in the Flask app for prediction.

```
# Instantiate the model
model = RandomForestClassifier()

# Train the model on the training data
model.fit(X_train, y_train)

# Save the trained model to a file using joblib for later use
joblib.dump(model, 'iris_model.pkl')

# Print confirmation message
print("Model saved as iris_model.pkl")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\pargat\Desktop\B Data Science Internship\Data Glacier\Week 4\Week4_Deployment_on_Flask> python train_model.py
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width      Species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
Model saved as iris_model.pkl
```

4. Flask Web Application

Flask Setup:
We built a simple **Flask web application** to deploy the trained model. This web app allows users to input flower measurements (sepal length, sepal width, petal length, and petal width) and receive a predicted species in return.

App Structure:

1. **Home Route**:
   The `/` route renders an HTML form where users can enter the flower's measurements.
2. **Prediction Route**:
   The `/predict` route handles the form submission. It receives the user input, scales the features, and makes a prediction using the pre-trained model. The predicted species is then displayed on the same page.

Model Integration:
The saved model (`iris_model.pkl`) is loaded into the Flask app using **joblib**, and predictions are made with the `model.predict()` method.

app.py > ...

```python
from flask import Flask, request, render_template, jsonify
import joblib
import numpy as np
from sklearn.preprocessing import StandardScaler

# Initialize the Flask app
app = Flask(__name__)

# Load the trained model and the scaler used for feature scaling
model = joblib.load('iris_model.pkl')  # Load the pre-trained RandomForest model
scaler = StandardScaler()  # Create a StandardScaler instance for feature scaling

# Route to render the form where users can input feature values
@app.route('/')
def home():
    return render_template('index.html')  # Render the HTML form (index.html) to the user

# Route to handle form submission and return the prediction
@app.route('/predict', methods=['POST'])
def predict():
    # Retrieve the input feature values from the form
    sepal_length = float(request.form['sepal_length'])  # Convert to float for model input
    sepal_width = float(request.form['sepal_width'])    # Convert to float for model input
    petal_length = float(request.form['petal_length'])  # Convert to float for model input
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
4         5.0         3.6          1.4          0.2  Iris-setosa
Model saved as iris_model.pkl
PS C:\Users\pargat\Desktop\B Data Science Internship\Data Glacier\Week 4\Week4_Deployment_on_Flask> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
```

5. Model Prediction via Web Interface

Web Interface (index.html)

The `index.html` file is the front-end of the Flask web application where users input flower measurements (sepal length, sepal width, petal length, petal width) for species prediction.

Key Components:

1. **Input Form**: The form includes four fields for numeric input (sepal length, sepal width, petal length, petal width). Each field accepts decimal values.
2. **Submit Button**: Users submit their input to the Flask server via a POST request to the `/predict` route.
3. **Prediction Display**: After submission, the predicted species is displayed below the form using Flask's Jinja templating engine if a prediction is made.

Code Overview:

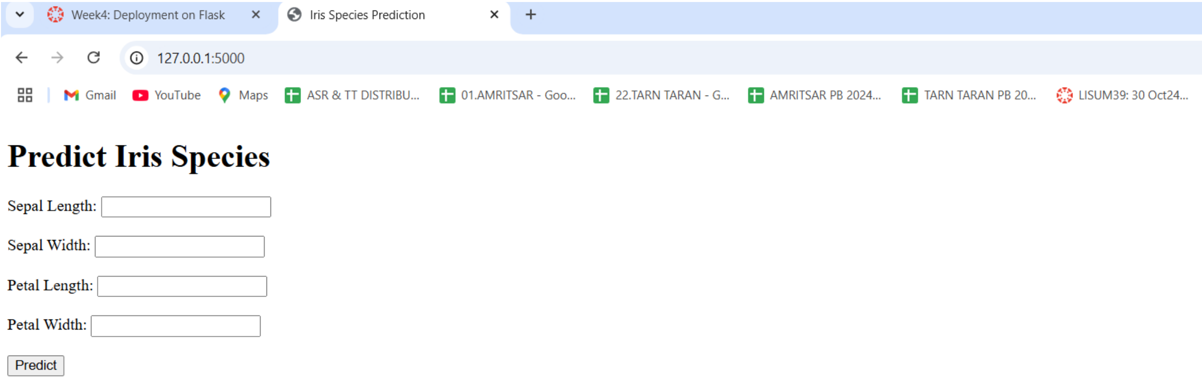
The HTML uses a simple layout with a form for user input and styling to ensure a clean and user-friendly interface. The result is shown dynamically after prediction.

This page interacts with the Flask backend to receive input, send it for prediction, and display the result.

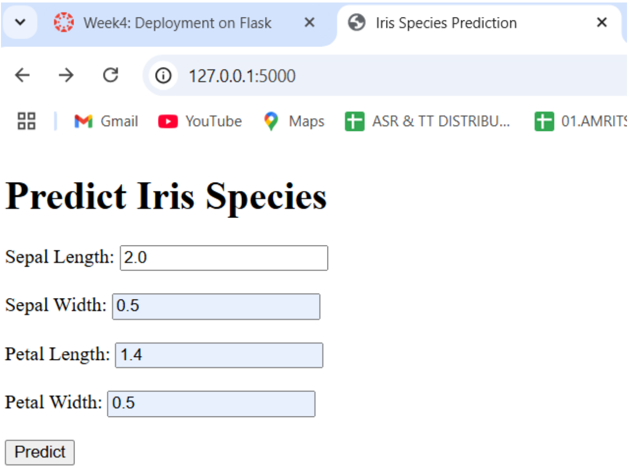
templates > index.html > html > body

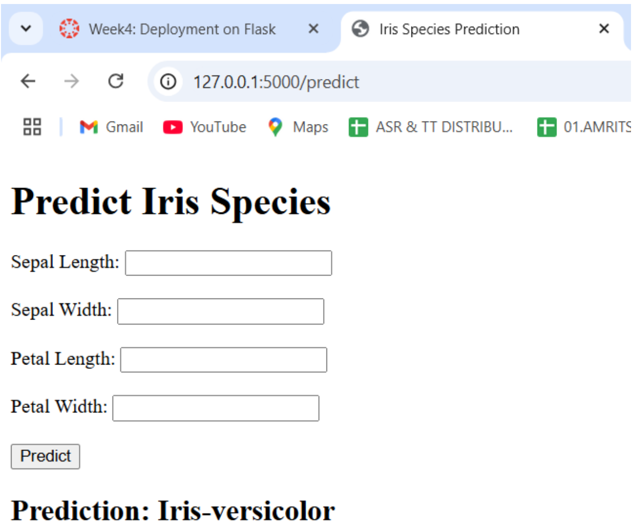
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Iris Species Prediction</title>
</head>
<body>
    <h1>Predict Iris Species</h1>
    <form action="/predict" method="POST">
        <label for="sepal_length">Sepal Length:</label>
        <input type="text" id="sepal_length" name="sepal_length" required><br><br>

        <label for="sepal_width">Sepal Width:</label>
        <input type="text" id="sepal_width" name="sepal_width" required><br><br>

        <label for="petal_length">Petal Length:</label>
        <input type="text" id="petal_length" name="petal_length" required><br><br>

        <label for="petal_width">Petal Width:</label>
        <input type="text" id="petal_width" name="petal_width" required><br><br>

        <input type="submit" value="Predict">
    </form>

    {% if prediction %}
    <h2>Prediction: {{ prediction }}</h2>
    {% endif %}
</body>
</html>
```

- **Open the webpage in your browser (`http://127.0.0.1:5000/`)**



- **Interact with the web form and get predictions**





6. Conclusion

Summary:
This project successfully demonstrated how to build and deploy a machine learning model for classifying Iris flowers. By using the Iris dataset, we trained a Random Forest model and deployed it as a web application with Flask. Users can input flower measurements and receive a prediction of the flower’s species.

Key Learnings:

- Proper data preprocessing, including feature scaling and data splitting, is essential for building a good predictive model.
- Flask is an easy and effective tool for deploying machine learning models as web applications.
- Model deployment allows users to interact with machine learning models through a user-friendly interface.