

# 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

train_model.py > ...
1  import pandas as pd
2  from sklearn.preprocessing import StandardScaler
3  from sklearn.ensemble import RandomForestClassifier
4  from sklearn.model_selection import train_test_split
5  import joblib
6
7  # Load the dataset from a CSV file
8  df = pd.read_csv("C:/Users/pargat/Desktop/B Data Science Internship/Data Glacier/Week 4/Week4_Deployment_on_Flask/iris.data.csv")
9
10 # Display the first few rows of the dataset to understand its structure
11 print(df.head())
12
13 # Select independent variables (features) and the dependent variable (target)
14 X = df[["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]] # Features
15 y = df["Species"] # Target variable (species)
16
17 # Split the dataset into training and testing sets (70% train, 30% test)
18 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=50)
19
20 # Feature scaling: Standardizing the features (mean=0, std=1) for better model performance
21 sc = StandardScaler()
22 X_train = sc.fit_transform(X_train) # Fit on training data and transform it
23 X_test = sc.transform(X_test) # Transform the test data based on the training data scaling
24
  
```

## 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.

```
24
25 # Instantiate the model
26 model = RandomForestClassifier()
27
28 # Train the model on the training data
29 model.fit(x_train, y_train)
30
31 # Save the trained model to a file using joblib for later use
32 joblib.dump(model, 'iris_model.pkl')
33
34 # Print confirmation message
35 print("Model saved as iris_model.pkl")
36
```

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 > ...
1  from flask import Flask, request, render_template, jsonify
2  import joblib
3  import numpy as np
4  from sklearn.preprocessing import StandardScaler
5
6  # Initialize the Flask app
7  app = Flask(__name__)
8
9  # Load the trained model and the scaler used for feature scaling
10 model = joblib.load('iris_model.pkl') # Load the pre-trained RandomForest model
11 scaler = StandardScaler() # Create a StandardScaler instance for feature scaling
12
13 # Route to render the form where users can input feature values
14 @app.route('/')
15 def home():
16     return render_template('index.html') # Render the HTML form (index.html) to the user
17
18 # Route to handle form submission and return the prediction
19 @app.route('/predict', methods=['POST'])
20 def predict():
21     # Retrieve the input feature values from the form
22     sepal_length = float(request.form['sepal_length']) # Convert to float for model input
23     sepal_width = float(request.form['sepal_width']) # Convert to float for model input
24     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.

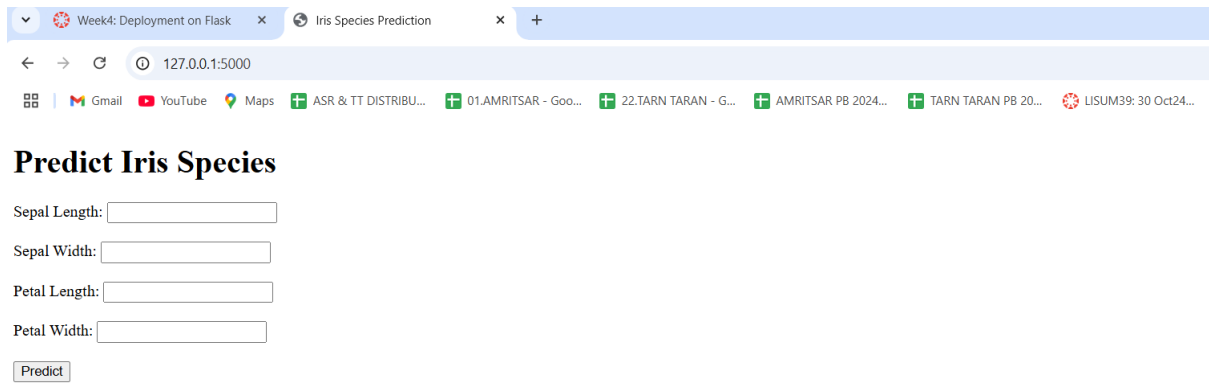
```

templates > index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Iris Species Prediction</title>
7  </head>
8  <body>
9      <h1>Predict Iris Species</h1>
10     <form action="/predict" method="POST">
11         <label for="sepal_length">Sepal Length:</label>
12         <input type="text" id="sepal_length" name="sepal_length" required><br><br>
13
14         <label for="sepal_width">Sepal Width:</label>
15         <input type="text" id="sepal_width" name="sepal_width" required><br><br>
16
17         <label for="petal_length">Petal Length:</label>
18         <input type="text" id="petal_length" name="petal_length" required><br><br>
19
20         <label for="petal_width">Petal Width:</label>
21         <input type="text" id="petal_width" name="petal_width" required><br><br>
22
23         <input type="submit" value="Predict">
24     </form>
25
26     {% if prediction %}
27     <h2>Prediction: {{ prediction }}</h2>
28     {% endif %}
29 </body>
30 </html>
31

```



Open the webpage in your browser (<http://127.0.0.1:5000/>)



Week4: Deployment on Flask Iris Species Prediction

127.0.0.1:5000

Gmail YouTube Maps ASR & TT DISTRIBU... 01.AMRITSAR - Goo... 22.TARN TARAN - G... AMRITSAR PB 2024... TARN TARAN PB 20... LISUM39: 30 Oct24...

## Predict Iris Species

Sepal Length:

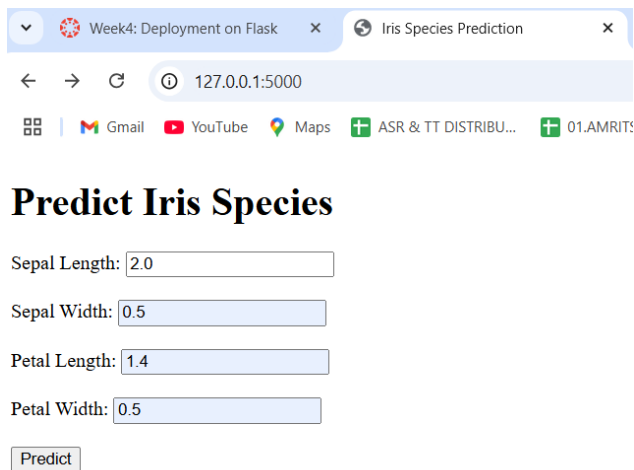
Sepal Width:

Petal Length:

Petal Width:



Interact with the web form and get predictions



Week4: Deployment on Flask Iris Species Prediction

127.0.0.1:5000

Gmail YouTube Maps ASR & TT DISTRIBU... 01.AMRITS

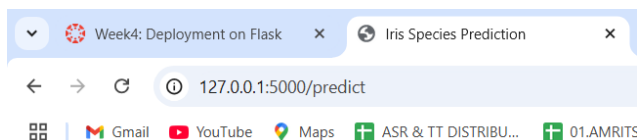
## Predict Iris Species

Sepal Length:

Sepal Width:

Petal Length:

Petal Width:



Week4: Deployment on Flask Iris Species Prediction

127.0.0.1:5000/predict

Gmail YouTube Maps ASR & TT DISTRIBU... 01.AMRITS

## Predict Iris Species

Sepal Length:

Sepal Width:

Petal Length:

Petal Width:

**Prediction: Iris-versicolor**

## 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.