Deployment on Flask - Iris Data Model

Week 4 Assignment

Name: Nazri

Batch code: LISUM34

Submission date: 26/06/2024

Submitted to: Data Glacier

Flask Deployment on Iris Data Model

Steps Followed:

1. Pick Iris Toy Data Set

	A	B	C	D	E
1	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
2	5.1	3.5	1.4	0.2	Setosa
3	4.9	3	1.4	0.2	Setosa
4	4.7	3.2	1.3	0.2	Setosa
5	4.6	3.1	1.5	0.2	Setosa
6	5	3.6	1.4	0.2	Setosa
7	5.4	3.9	1.7	0.4	Setosa
8	4.6	3.4	1.4	0.3	Setosa
9	5	3.4	1.5	0.2	Setosa
10	4.4	2.9	1.4	0.2	Setosa
11	4.9	3.1	1.5	0.1	Setosa
12	5.4	3.7	1.5	0.2	Setosa
13	4.8	3.4	1.6	0.2	Setosa
14	4.8	3	1.4	0.1	Setosa
15	4.3	3	1.1	0.1	Setosa
16	5.8	4	1.2	0.2	Setosa
17	5.7	4.4	1.5	0.4	Setosa
18	5.4	3.9	1.3	0.4	Setosa
19	5.1	3.5	1.4	0.3	Setosa
20	5.7	3.8	1.7	0.3	Setosa
21	5.1	3.8	1.5	0.3	Setosa

2. Import necessary libraries:

Prepare the environment by installing necessary libraries like Scikit-learn and importing them. Also ensure the compatibility of Scikit-learn version with the IDE PyCharm

```
[23] !pip install scikit-learn==1.5.1
```

```
Requirement already satisfied: scikit-learn==1.5.1 in /usr/local/lib/python3.10/dist-packages (1.5.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (3.5.0)
```

```
[2] # Import Libraries
import pandas as pd
import sklearn
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import pickle
import gdown
```

```
print(sklearn.__version__)
```

```
1.5.1
```

3. Downloading the Dataset:

Using ‘gdown’ download the Iris dataset from Google Drive.

4. Loading the Iris Dataset into a Dataframe:

The dataset is read from the CSV file using pandas from the contents folder.

```
[4] #importing the dataset from drive
    gdown.download_folder('https://drive.google.com/drive/folders/1Akoln8Xc14yMxO1AXQw88YddMFyrfEft?', quiet=True)
```

```
['/content/Iris-Dataset/iris.csv']
```

```
[5] #Load the dataset
    iris_data = pd.read_csv('/content/Iris-Dataset/iris.csv')
```

5. Exploratory Data Analysis:

The head, shape, info, and isnull methods are used to inspect the dataset's structure, datatypes, and check for missing values.

```
iris_data.head()
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
[7] iris_data.shape
```

```
(150, 5)
```

```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Sepal_Length  150 non-null    float64
 1   Sepal_Width   150 non-null    float64
 2   Petal_Length  150 non-null    float64
 3   Petal_Width   150 non-null    float64
 4   Class         150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
[9] # check fo null vallues
    iris_data.isnull().sum()
```

	0
Sepal_Length	0
Sepal_Width	0
Petal_Length	0
Petal_Width	0
Class	0

dtype: int64

Check the summary statistics

```
[13] iris_data.describe()
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
count	149.000000	149.000000	149.000000	149.000000
mean	5.843624	3.059732	3.748993	1.194631
std	0.830851	0.436342	1.767791	0.762622
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.300000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Check for Duplicates

```
[10] # Check for duplicates
     num_duplicates = iris_data.duplicated().sum()
     print(f"Number of duplicate rows: {num_duplicates}")
```

```
Number of duplicate rows: 1
```

```
[11] # Identify duplicate records
     duplicates = iris_data[iris_data.duplicated()]

     # Print duplicate records
     print("Duplicate records:")
     print(duplicates)
```

```
Duplicate records:
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width      Class
142           5.8          2.7           5.1          1.9  Virginica
```

Note: In the excel sheet, the duplicated data is aligned in the 103 and 144 row.

101	5.7	2.8	4.1	1.3	Versicolor
102	6.3	3.3	6	2.5	Virginica
103	5.8	2.7	5.1	1.9	Virginica
104	7.1	3	5.9	2.1	Virginica
105	6.3	2.9	5.6	1.8	Virginica
142	6.7	3.1	5.6	2.4	Virginica
143	6.9	3.1	5.1	2.3	Virginica
144	5.8	2.7	5.1	1.9	Virginica
145	6.8	3.2	5.9	2.3	Virginica
146	6.7	3.3	5.7	2.5	Virginica

6. Data Preprocessing:

Remove the duplicate rows identified during EDA process.

```
[12] # Remove duplicate rows
     iris_data = iris_data.drop_duplicates()

     # Verify that duplicates are removed
     num_duplicates_after = iris_data.duplicated().sum()
     print(f"Number of duplicate rows after cleaning: {num_duplicates_after}")
```

```
Number of duplicate rows after cleaning: 0
```

7. Splitting the Dataset

Select the features and target variables from the dataset and split the dataset into training and testing sets.

```
[14] # Select independent and dependent variable
     # Split the data into features and target
     X = iris_data[["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]]
     y = iris_data["Class"]
```

```
[15] # Split the dataset into train and test
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

8. Model Selection and Training the model:

Train the machine learning model on the pre-processed data with machine learning algorithms such as random forest classifier and logistic regression. Fit the model on the training data.

9. Model Evaluation:

Evaluate the model using the test set. And check its accuracy, precision, recall, and F1-score.

```
[16] # Train the model with Randon forest classifier
     rfc = RandomForestClassifier()

     # Fit the model
     rfc.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
[17] #Evaluate the model
     y_pred = rfc.predict(X_test)
     accuracy = accuracy_score(y_test, y_pred)
     print('Randon Forest Model Accuracy:', accuracy)
```

```
Randon Forest Model Accuracy: 1.0
```

```
[18] # Train the model with logistic regression
     lg= LogisticRegression(max_iter=200)

     # Fit the model
     lg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=200)
```

```
# Evaluate the model
y_pred = lg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Logistic Regression Model Accuracy:', accuracy)
```

```
Logistic Regression Model Accuracy: 1.0
```

```
from sklearn.metrics import classification_report

# Evaluate Random Forest
y_pred_rf = rfc.predict(X_test)
print('Random Forest Classification Report:\n', classification_report(y_test, y_pred_rf))

# Evaluate Logistic Regression
y_pred_lg = lg.predict(X_test)
print('Logistic Regression Classification Report:\n', classification_report(y_test, y_pred_lg))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        10
  Versicolor       1.00      1.00      1.00         9
   Virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Logistic Regression Classification Report:
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        10
  Versicolor       1.00      1.00      1.00         9
   Virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Summary:

When both models achieve an accuracy of 1 on the test data, it might indicate that the models are overfitting, especially if the dataset is small or lacks complexity. Overfitting occurs when a model learns the training data too well, including noise and outliers, leading to poor generalization on unseen data.

10. Perform Cross-Validation on the models

To ensure that the models are truly generalizing well, we should use cross-validation. This involves splitting the dataset into multiple folds and training/evaluating the model on different folds. This process helps in assessing how the model performs across different subsets of the data.

```
[21] from sklearn.model_selection import cross_val_score

     # Cross-validation for Random Forest
     rf_cv_scores = cross_val_score(rfc, X_train, y_train, cv=5)
     print('Random Forest Cross-Validation Scores:', rf_cv_scores)
     print('Random Forest Mean CV Score:', rf_cv_scores.mean())

     # Cross-validation for Logistic Regression
     lg_cv_scores = cross_val_score(lg, X_train, y_train, cv=5)
     print('Logistic Regression Cross-Validation Scores:', lg_cv_scores)
     print('Logistic Regression Mean CV Score:', lg_cv_scores.mean())
```

```
Random Forest Cross-Validation Scores: [1.         0.91666667 0.875      1.         0.95652174]
Random Forest Mean CV Score: 0.9496376811594203
Logistic Regression Cross-Validation Scores: [1.         0.91666667 0.875      1.         0.95652174]
Logistic Regression Mean CV Score: 0.9496376811594203
```

11. Choosing the Best Model Random Forest:

Choose the model Random Forest Classifier considering the following features

- Handle Non-linearity
- Robustness to Outliers
- Handle large datasets and Complex Patterns

12. Save the trained model using pickle

```
[22] # Choosing the model
     # Make pickle file of our model
     pickle.dump(rfc, open("model.pkl", "wb"))
```

13. Setting up Flask Application

Create a Flask application (app.py).

- Load the saved model in the Flask app.
- Define routes for prediction, such as /predict.
- Use request to get input from the user and return predictions.

Create a HTML Template (index.html)

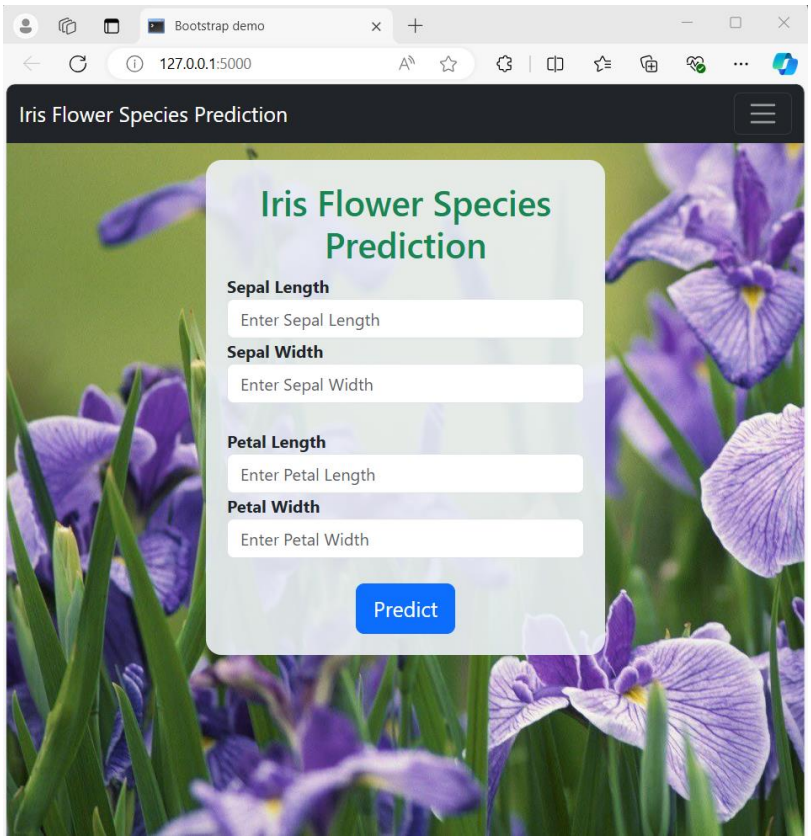
- In a templates directory, create index.html for user input

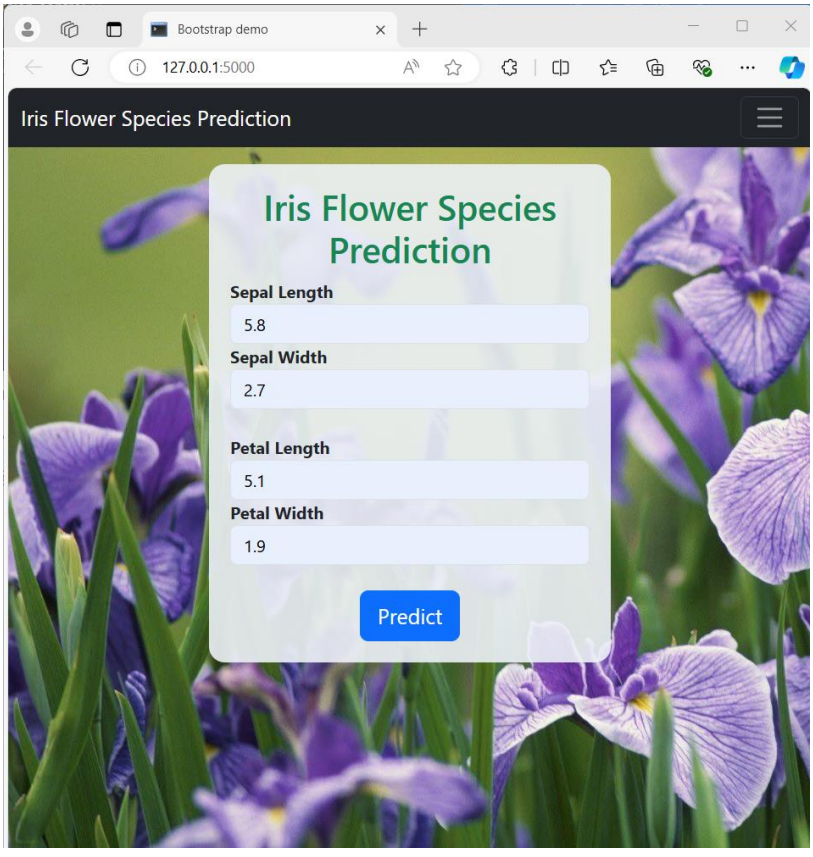
14. Running the Flask App:

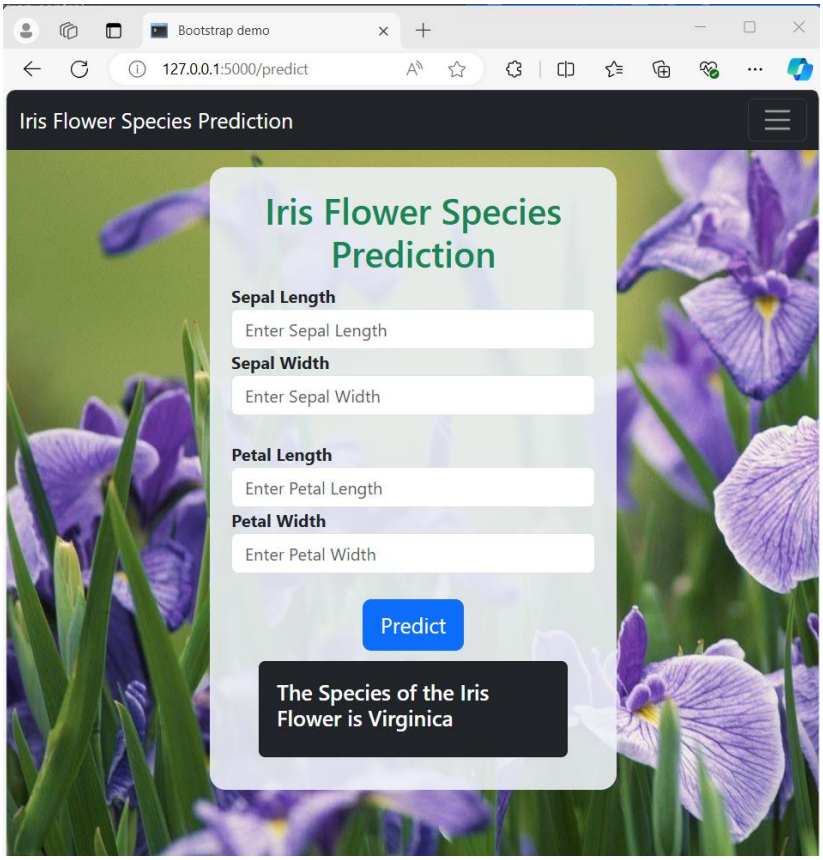
- Run the Flask app locally.

```
"C:\Users\nazri_c98ckep\PycharmProject\Flask Deployment-Iris Data\venv\Scripts\python.exe" "C:\Users\nazri_c98ckep\PycharmProject\Flask Deployment-Iris Data\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
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 420-096-619
```

Output







app.py

```
from flask import Flask, request, render_template
import numpy as np
import pickle
import sklearn
import pandas as pd

# importing model
model = pickle.load(open('model.pkl', 'rb'))

# creating flask app
app = Flask(__name__)

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

@app.route("/predict", methods=['POST'])
def predict():
    # Collecting input features from the form
    float_features = [float(x) for x in request.form.values()]
    # print("Received input features:", float_features)  # Debugging

    # Defining the feature names as used during the model training
    feature_names = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']

    # Creating a DataFrame with the feature names
```

```
    features = pd.DataFrame([float_features], columns=feature_names)
    # print("DataFrame created:", features)  # Debugging

    # Making predictions
    prediction = model.predict(features)
    # print("Prediction:", prediction)  # Debugging

    # Rendering the template with the prediction result
    return render_template("index.html", prediction_text="The Species of the Iris Flower is
{}".format(prediction[0]))

# Main function to run the Flask app
if __name__ == "__main__":
    app.run(debug=True)
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"
crossorigin="anonymous">
  </head>
  <style>
    body {
      background-image: url('{{ url_for('static', filename='img.jpeg') }}');
      background-size: cover;
      background-repeat: no-repeat;
      background-attachment: fixed;
    }
    h1 {
      color: #BE2ED6;
      text-align: center;
    }
    .warning {
      color: red;
      font-weight: bold;
      text-align: center;
    }
    .card {
      margin: 10px auto;
      color: white;
    }
    .container {
      background: rgba(237, 242, 247, 0.9); /* Semi-transparent background */
      font-weight: bold;
```

```
    padding: 20px; /* Increased padding for better spacing */
    border-radius: 15px;
    width: 50%; /* Set width to 50% of the viewport */
    max-width: 600px; /* Maximum width */
    margin: 0 auto; /* Center the container horizontally */
   }
  </style>

  <body>
   <!--
==========navbar==========
====-->
   <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
     <a class="navbar-brand" href="/">Iris Flower Species Prediction</a>
     <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
     </button>
     <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
       <li class="nav-item">
        <a class="nav-link active" aria-current="page" href="#">Home</a>
       </li>
       <li class="nav-item">
        <a class="nav-link" href="#">Contact</a>
       </li>
       <li class="nav-item">
        <a class="nav-link" href="#">About</a>
       </li>
      </ul>
      <form class="d-flex" role="search">
       <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
       <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
     </div>
    </div>
   </nav>

   <!--
==========
==========-->
   <div class="container my-3 mt-3">
    <h1 class="text-success">Iris Flower Species Prediction<span class="text-success"></span></h1>

    <!-- adding form -->
    <form action="/predict" method="POST">
     <div class="row">
      <div class="col-md-6">
       <label for="Sepal Length">Sepal Length</label>
```

```
          <input type="text" id="Sepal_Length" name="Sepal Length" placeholder="Enter Sepal Length"
class="form-control" required="required">
        </div>
        <div class="col-md-6">
          <label for="Sepal Width">Sepal Width</label>
          <input type="text" id="Sepal_Width" name="Sepal Width" placeholder="Enter Sepal Width"
class="form-control" required="required">
        </div>
       </div>

       <div class="row mt-4">
        <div class="col-md-6">
          <label for="Petal Length">Petal Length</label>
          <input type="text" id="Petal_Length" name="Petal Length" placeholder="Enter Petal Length"
class="form-control" required="required">
        </div>
        <div class="col-md-6">
          <label for="Petal Width">Petal Width</label>
          <input type="text" id="Petal_Width" name="Petal Width" placeholder="Enter Petal Width"
class="form-control" required="required">
        </div>
       </div>

       <div class="row mt-4">
        <div class="col-md-12 text-center">
          <button type="submit" class="btn btn-primary btn-lg">Predict</button>
        </div>
       </div>
     </form>

     {% if prediction_text %}
     <div class="card bg-dark" style="width: 18rem;">
      <div class="card-body">
        <h5 class="card-title">{{ prediction_text }}</h5>
      </div>
     </div>
     {% endif %}
   </div>

   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4ddQ3GYNS7NTKfAdVQSZe"
crossorigin="anonymous"></script>
  </body>
</html>
```