Data Glacier
Week 4: Deployment on Flask

Name: Loh Wan Teng
Batch Code: LISUM11
Submission Date: 28 July 2022
Submitted to: Data Glacier

Table of Content

1. Introduction

The project uses Random Forest to classify the Iris species based on the users' input values.

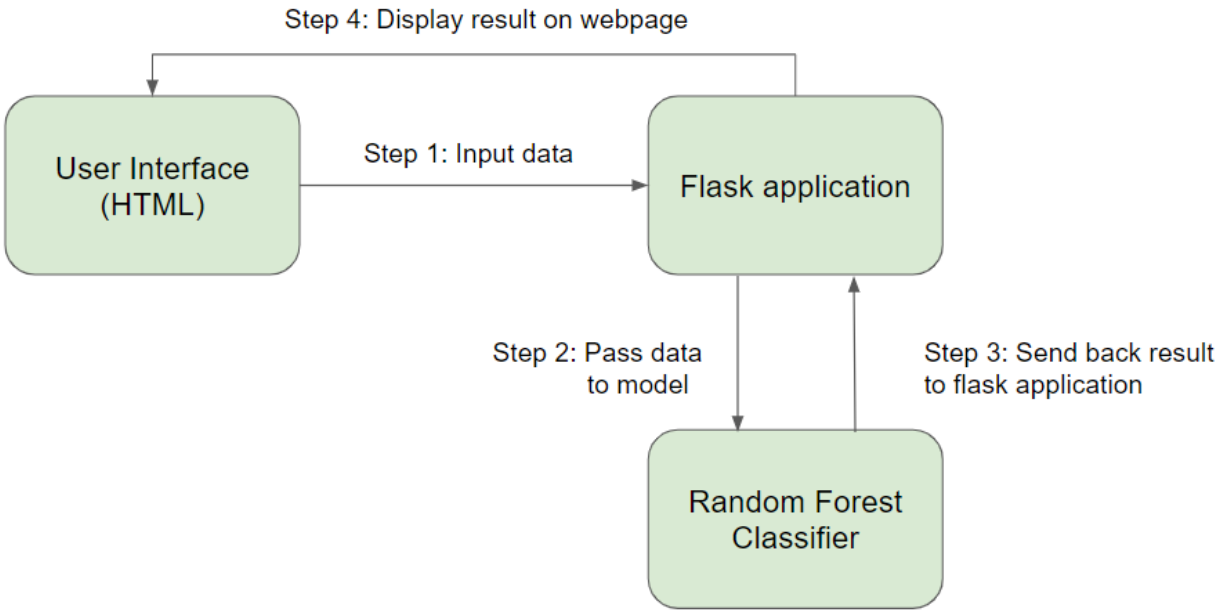


Figure 1: Application Workflow

The user interface allows users to input the Iris information. The input values are then passed to the Random Forest model. The result generated by the classifier is then sent back to the flask application to be displayed on the webpage.

2. Data Information

The dataset is downloaded from Kaggle. The dataset consists of 150 rows with 6 columns, recording the information about Iris. The table below shows the attribute information of the dataset.

Attributes	Descriptions
Id	Input ID of the data.
SepalLengthCm	Sepal length of the plant recorded in “cm”.

SepalWidthCm	Sepal width of the plant recorded in “cm”.
PetalLengthCm	Petal length of the plant recorded in “cm”.
PetalWidthCm	Petal width of the plant recorded in “cm”.
Species	Species of the plant.

Table 2: Attribute Information

3. Machine Learning Model

3.1. Import libraries and dataset

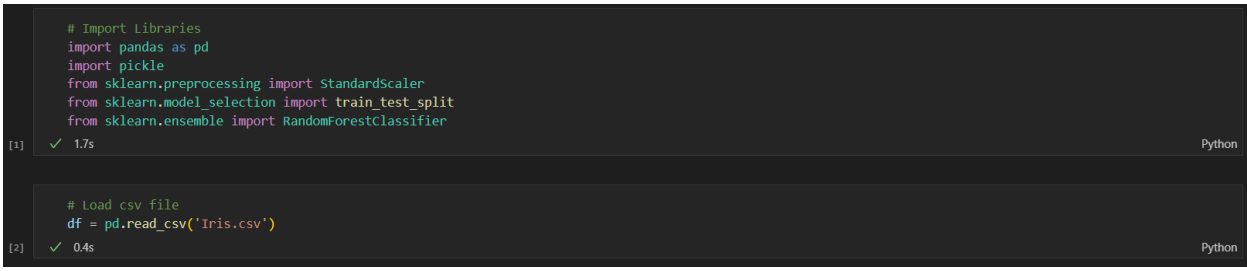
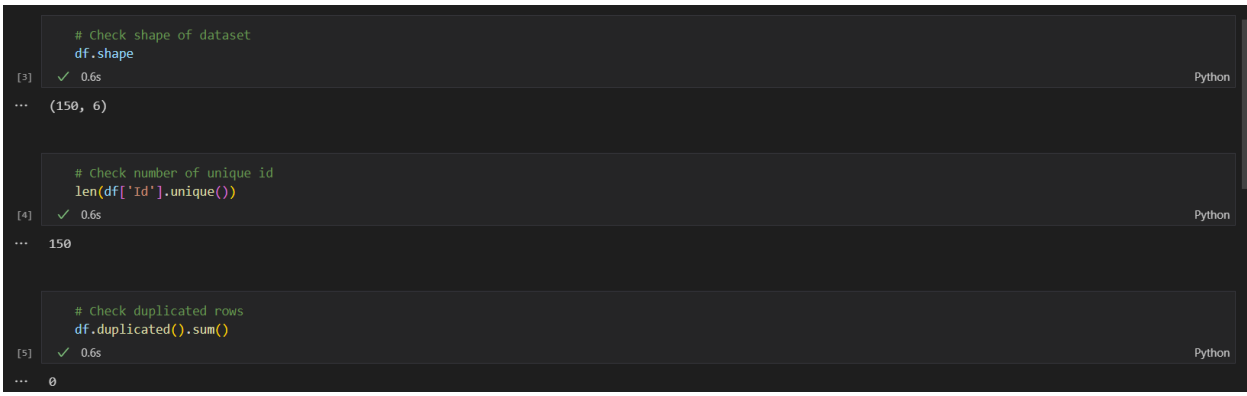


Figure 3: Import Libraries

3.2. Data Preprocessing

Dataset is split into 80% training data and 20% testing data.



```
# Check NA columns
df.isna().sum()
```
[6] ✓ 0.8s Python

```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
Species         0
dtype: int64
```

```
# Check null columns
df.isnull().sum()
```
[7] ✓ 0.5s Python

```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
Species         0
dtype: int64
```

```
# Seperate independent and dependent variable
X = df.iloc[:, 1:-1] # eliminate 'Id' column
y = df['Species']
```
[8] ✓ 0.6s Python

```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```
[9] ✓ 0.5s Python

```
# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```
[10] ✓ 0.6s Python

Figure 4, 5, 6: Data Preprocessing

3.3. Build Model

```
# Instantiate the model
classifier = RandomForestClassifier()
```
[11] ✓ 0.5s Python

```
# Fit model
classifier.fit(X_train, y_train)
```
[12] ✓ 0.2s Python

```
▾ RandomForestClassifier
RandomForestClassifier()
```

Figure 7: Build Model

3.4. Save Model

A “model.pkl” file is generated using pickle.

```
# Make pickle file
pickle.dump(classifier, open("model.pkl", "wb"))
```
[13] ✓ 0.5s Python

Figure 8: Save Model

4. Model Deployment on Flask

The file directory of the project is shown below. The **templates** folder contains the webpage interface which all

templates/
index.html
app.py
Iris.csv
model.ipynb
model.pkl

Table 9: File Directory

4.1. app.py

The **app.py** contains the main code to be executed by the Python interpreter to run the Flask web application.

```python
import numpy as np
import pickle
from flask import Flask, request, render_template

# Create flask app
app = Flask(__name__)

# Load pickle model
model = pickle.load(open("model.pkl", "rb"))

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

@app.route("/predict", methods = ["POST"])
def predict():
    float_features = [float(x) for x in request.form.values()]
    features = [np.array(float_features)]
    prediction = model.predict(features)
    return render_template("index.html", prediction_text = "The flower species is {}".format(prediction))

if __name__ == "__main__":
    app.run(port = 5000, debug = True)
```

Figure 10: app.py

- A Flask application with argument **__name__** is initialised.
- The Random Forest model is loaded by opening the **model.pkl** with read binary file access (**rb**).
- The route decorator (**@app.route("/")**) is used to specify the URL that should trigger the execution of the home function.

- The **home()** function is to reder the **index.html** file located in the **templates** folder.
- The **predict()** function send the form data using **POST** method. The data is then converted to a list of float numbers and is then used for the model prediction. Finally, the prediction result is rendered on the **index.html**.
- The **run** function is executed if the script is directly executed by the Python interpreter(**__main__**) to debug and run the Flask application on the sever with port 5000.

4.2. templates/index.html

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>Flower Class Prediction API</title>
</head>

<body>
  <div class="login">
    <h1>Flower Class Prediction</h1>
    <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
      <input type="text" name="Sepal_Length" placeholder="Sepal_Length" required="required" />
      <input type="text" name="Sepal_Width" placeholder="Sepal_Width" required="required" />
      <input type="text" name="Petal_Length" placeholder="Petal_Length" required="required" />
      <input type="text" name="Petal_Width" placeholder="Petal_Width" required="required" />
      <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>
    <br>
    <br>
    {{ prediction_text }}
  </div>
</body>
</html>
```

Figure 11: templates/index.html

The users are required to input all information about the flower in order to predict the species of the flower by clicking the “Predict” button.

4.3. Flask Overview

```
PS C:\Users\WT\OneDrive\Desktop\GitHub\12-weeks-data-science\Week4> python app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 307-633-402
127.0.0.1 - - [12/Aug/2022 17:46:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Aug/2022 17:46:46] "GET /predict HTTP/1.1" 405 -
```

Figure 12: Running Flask application

Figure 13: User Interface

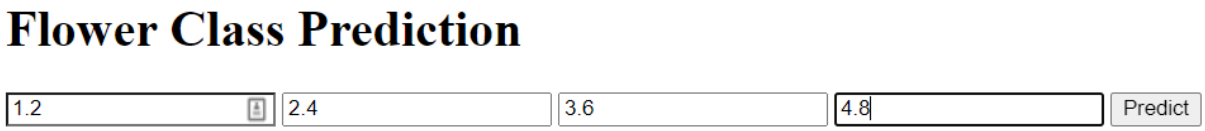


Figure 14: Input Value by User

Figure 15: Prediction