Data Analyst Internship

WEEK 4: DEPLOYMENT ON FLASK

Name: Ramya Hariharan

Batch Code: LISUM 12

Date: 27-Aug-2022

Submitted to: Data Glacier

Table of Contents:

1. Introduction

In this project, we use the Flask Framework to deploy the machine learning model Random Forest (RF) classifier. As an example, our approach helps to predict the species of flowers. Utilizing Flask, the Python micro-framework for developing web applications, establish an API for the model. Through HTTP queries, this API enables us to make use of predictive capabilities.

2. Data Information

Iris is a sample dataset that was acquired from Kaggle and stored as a CSV file. The independent and dependent variables of the dataset are listed in the table below. Class is the dependent variable, whereas Sepal Length, Sepal Width, Petal Length, and Petal Width are independent variables.

Id	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa

Table 2.1: Dataset Information

3. Building a Model

- Initially, We import the required libraries.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pickle
```

- Load the Dataset

```
#load the csv file

df = pd.read_csv("Iris.csv")
```

- Dataset Details

```
df.head()

#Dataset Details
print(df.head(5))
print(df.size)
print(df.shape)
print(df.keys())
```

```
   Id  Sepal_Length  Sepal_Width  Petal_Length  Petal_Width      Species
0   1           5.1          3.5           1.4          0.2  Iris-setosa
1   2           4.9          3.0           1.4          0.2  Iris-setosa
2   3           4.7          3.2           1.3          0.2  Iris-setosa
3   4           4.6          3.1           1.5          0.2  Iris-setosa
4   5           5.0          3.6           1.4          0.2  Iris-setosa
900
(150, 6)
Index(['Id', 'Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width',
       'Species'],
      dtype='object')
```

- Select Independent and dependent variable

```
#Select independent and dependent variable
x = df[["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]]
y = df["Species"]
```

3.1. Build Model

We use a machine learning model to classify the different types of flowers. We use scikit-learn to implement Random Forest (RF) for this purpose. Fit the Random Forest model onto the training dataset after importing and initialising it.

```
#Split the dataset into train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=50)

#Feature scaling
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

#Instantiate the model
classifier = RandomForestClassifier()

#Fit the model
classifier.fit(x_train, y_train)
```

3.2. Save the model

After that, we use Pickle to save the model.

```
#Make pickle file of our model
pickle.dump(classifier, open("model.pkl", "wb"))
```

4. Turning Model into Web Application

We create a simple online page with fields for Sepal Length, Sepal Width, Petal Length, and Petal Width that allows us to enter the length as a web application. If we click "predict" after filling out the length, it will reveal the species type.

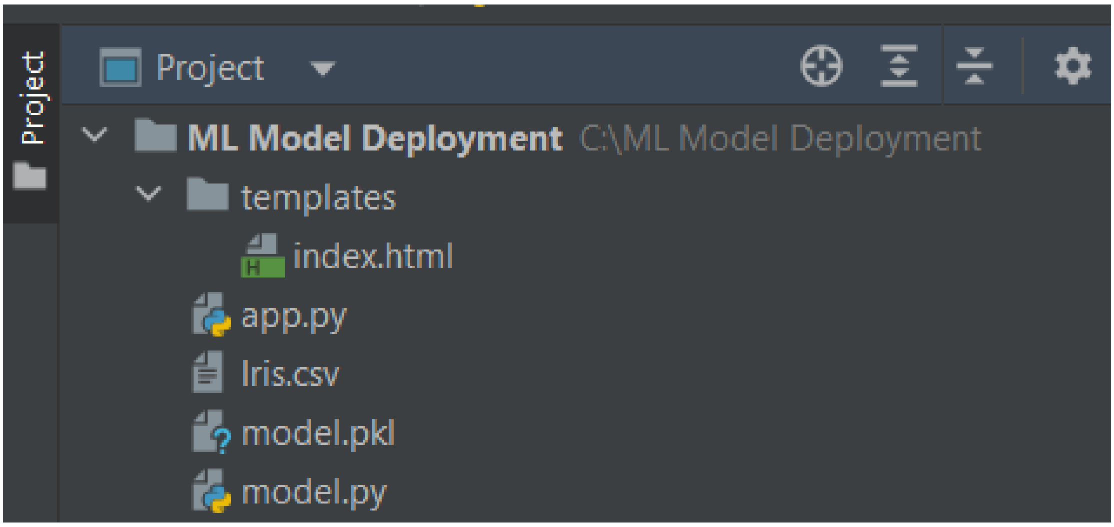

Table 4: Application Folder File Directory

For this project, we first create a folder called ML Model Deployment. The folder's file tree is represented by this. Each file's description is provided in the sections below. In our application, there is only one HTML file called index.html in the sub-directory templates, which is where Flask would seek static HTML files to render in the web browser.

4.1. App.py

The essential code for running the Flask web application, including the machine learning (ML) code for classification, is contained in the app.py file.

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

#Create flask app
app = Flask(__name__)

#Load the 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(debug=True)
```

Figure 4.1: App.py

- In order to inform Flask that it may locate the HTML template folder (templates) in the same directory where it is located, we initialized a new Flask instance with the argument __name__. This is because we ran our application as a single module.
- The URL that should cause the home function to run was then specified using the route decorator (@app.route('/')).
- The form data was sent to the server in the message body using the POST method. We further activated Flask's debugger by setting the debug=True option inside the app.run method.
- Finally, we utilised the run function to only launch the server-side application when the Python interpreter performed this script directly, which we verified using the if statement with __name__ == "__main__".

4.2. Index.html

The following are the contents of the index.html file that will render a text form where a user can enter the length of the flower.

```html
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
  <meta charset="UTF-8">
  <title>ML API</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
</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 4.2: index.html

4.3. Running Procedure

Once we have completed everything above, we can launch the API by double clicking app.py or by using the following command in the terminal:

```
"C:\Users\BALAJI RAMYA\anaconda3\python.exe" "C:/ML Model Deployment/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
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 753-831-909
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 4.3: Command Execution

We could now launch a web browser and go to http://127.0.0.1:5000/ to see a straightforward webpage with the content shown below.

Figure 4.4: Flower Class Prediction Website Page

Now we enter the length in each field.

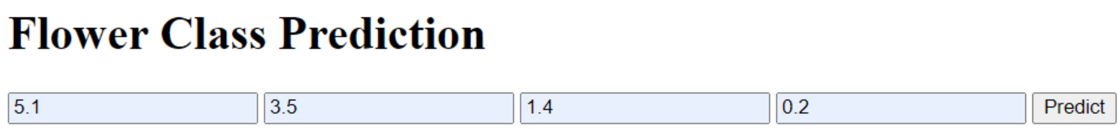

Figure 4.5: Input in the form

Clicking the "predict" button after entering the inputs allows us to see the outcome of our input.

Figure 4.6: Result of the given input