Week 4: Deployment on Flask

Report date: 28-May-2024

Internship Batch: LISUM33

Data intake by: Gunjan Varyani

Data intake reviewer : Data Glacier

1.Introduction:

In this project, we are going to deploy machine learning model using the Flask Framework. As a demonstration, our model helps to predict the salary based on the years of experience.

We will focus on both: building a machine learning model for Predicting the Salary expectations, then creating an API for the model, using Flask, the Python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests.

The table below lists the datasets, number of years of experience and based on the salary expectation is provided.

Years of Experience	Salary
1.1	39343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4	55794
4	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431
9	105582
9.5	116969
9.6	112635
10.3	122391
10.5	121872

2. Building a model :

Importing required libraries and dataset:

In this part, we import libraires and datasets which contain the information of the dataset.

```
#In this file, we will use the flask web framework to handle the POST requests that we
will get from the request.py and from HTML file


# Simple Linear Regression

This model predicts the salary of the employ based on experience using a simple linear
regression model.

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle

# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

2.1 Next we will Build a model:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3,
random_state = 0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

2.2 After that we will save the model and run the model.

```
# Saving model using pickle
pickle.dump(regressor, open('model.pkl','wb'))

# Loading model to compare the results
model = pickle.load( open('model.pkl','rb'))
print(model.predict([[1.8]]))
```

3. Turning model into web application:

We developed a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of Salary predictions.

3.1 App.py

The *app.py* file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying SD.

```
#In this file, we will use the flask web framework to handle the POST requests that we
will get from the request.py and from HTML file

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

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

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

@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text='Salary is
{}'.format(output))


@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls trought request
    '''
```

```
    data = request.get_json(force=True)
    prediction = model.predict([np.array(list(data.values()))])

    output = prediction[0]
    return jsonify(output)

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

1) First, we ran our application as a single module; thus, we initialized a new Flask instance with the argument *__name__* to let Flask know that it can find the HTML template folder (*templates*) in the same directory where it is located.
2) Next, we used the route decorator *(@app.route('/'))* to specify the URL that should trigger the execution of the home function.
3) Our *render_template* function simply rendered the *index.html* HTML file, which is in the *templates* folder.

4) Inside the *predict* function, we access the salary data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
5) We used the *POST* method to transport the form data to the server in the message body. Finally, by setting the *debug=True* argument inside the app.run method, we further activated Flask's debugger.
6) Lastly, we used the *run* function to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the *if* statement with *__name__ == '__main__'*.

4. Index.html

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

we create a `index.html` file that will be rendered via the `render_template` line return inside the `predict` function, which we defined in the `app.py` script to display the text that a user-submitted via the text field.

```
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
```

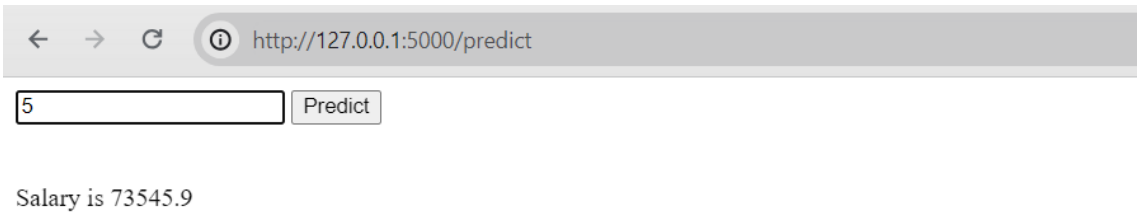
```
  <meta charset="UTF-8">
  <title>ML API</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet'
type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet'
type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet'
type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

<body>
 <div class="login">
 <div>

  </div>

     <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
      <input type="text" name="YearsExperience" placeholder="YearsExperience"
required="required" />

        <button type="submit" class="btn btn-primary btn-block btn-
large">Predict</button>
    </form>

   <br>
   <br>
   {{ prediction_text }}

 </div>


</body>
</html>
```

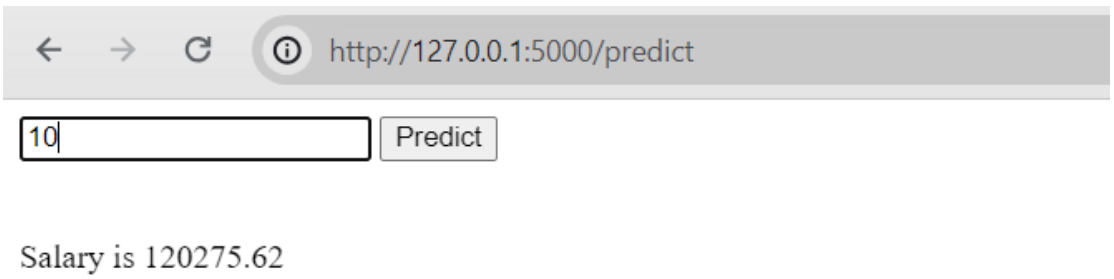
5. Running Procedure

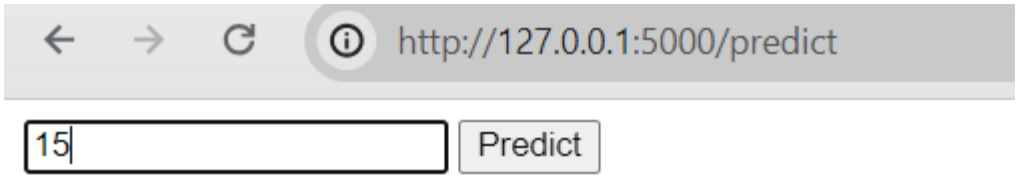
Once we have done all the above steps, we can start running the API by either double click *app.py*, or executing the command from the Terminal:

```
(.venv) D:\Data Glacier Intern\week 4>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
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 685-308-960
```

Now we could open a web browser and navigate to http://127.0.0.1:5000/, we should see a simple website with the content like so






http://127.0.0.1:5000/predict
15
Predict
Salary is 167005.33