Data Glacier

Data Science Intern at Data Glacier

Week 4: Deployment on Flask

Name: Jelson Lino

Batch Code: LISUM12

Date: 06 April 2022

Submitted to: Data Glacier

Table of Contents:

1. Introduction 3
2. Data Information 3
3. Model Building 4
4. Turning Model into Flask Framework 6
   - 4.1. App.py 7
   - 4.2. Index.html 8
   - 4.3. Style.css 9
   - 4.4. Running Procedure 9

1. Introduction

This project involves the deployment of a linear regression model using the Flask framework. The model presented here predicts the performance of a robot called EinsBot. The model uses four features to determine EinsBot output. Because the robot is still under construction and being developed for a client, in this study, the features are named Displacement, Input Parameter A, Input Parameter B, and Input Parameter C, while the target variable is called Output Parameter. In this project, we first built a machine learning model to determine the output of Einsbot and then create 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.

2. Data Information

The data used in this project are readings collected by the robot. The readings were previously cleaned and saved in a single CSV file, and the features were renamed for the purpose of this study. This project does not involve data cleaning. This report simply covers model building and deployment. Table 1 and Figure 1 show data organization, data type, features, and the number of readings.

Figure 1: Dataset information

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Displacement      400 non-null    float64
 1   Input Parameter A 400 non-null    float64
 2   Input Parameter B 400 non-null    float64
 3   Output Parameter  400 non-null    float64
 4   Input Parameter C 400 non-null    float64
dtypes: float64(5)
memory usage: 15.8 KB
```

Table 1: Data organization

Displacement	Input Parameter A	Input Parameter B	Output Parameter	Input Parameter C
-20.0	-58.0	-45.0	20.0	13.00
-19.9	-57.7	-44.8	19.9	12.95
-19.8	-57.4	-44.6	19.8	12.90
-19.7	-57.1	-44.4	19.7	12.85
-19.6	-56.8	-44.2	19.6	12.80

3. Model Building

Import Libraries and Dataset

This part of the report shows the libraries and dataset used in the study.

In [1]:
```
import pandas as pd
import numpy as np
import pickle
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_absolute_error
```

In [1]:
```
import pandas as pd
import numpy as np
import pickle
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_absolute_error
```

Data Preprocessing

The dataset was vertically split into the target and features. The output parameter was selected as the variable while the remaining variables were selected as features. In addition, the data was horizontally split, with 80% of the data being used for the training of the model and the remaining 20% being used for the testing of the model.

In [5]:
```
# Define target and features
target = "Output Parameter"
features = ["Displacement", "Input Parameter A", "Input Parameter B", "Input Parameter C"]
y = readings[target]
X = readings[features]
X.head()
```

Out[5]:

	Displacement	Input Parameter A	Input Parameter B	Input Parameter C
0	-20.0	-58.0	-45.0	13.00
1	-19.9	-57.7	-44.8	12.95
2	-19.8	-57.4	-44.6	12.90
3	-19.7	-57.1	-44.4	12.85
4	-19.6	-56.8	-44.2	12.80

2.2 Horizontal Split

In [6]:
```
# Split the data into train and test
cutoff = int(len(X) * 0.8)
X_train, y_train = X.iloc[ :cutoff], y.iloc[:cutoff]
X_test, y_test = X.iloc[cutoff: ], y.iloc[cutoff: ]
```

Model Building and Evaluation

After the data was preprocessed, a linear regression model was built and the preprocessed data was used to train and evaluate the model as shown below. Mean absolute error was used to evaluate the model. As shown below, the model shows a positive behavior when exposed to the test data.

3.1 Setting the model MAE baseline

In [7]:
```
y_mean = y_train.mean()
y_pred_baseline = [y_mean]*len(y_train)
print("Mean Output Parameter value:", round(y_mean, 2))
print("Baseline MAE:", mean_absolute_error(y_train, y_pred_baseline))
```

```
Mean Output Parameter value: 4.05
Baseline MAE: 8.0
```

3.2 Model Iteration

In [8]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[8]:
```
LinearRegression()
```

3.3 Model Evaluation

In [9]:
```
training_mae = mean_absolute_error(y_train, model.predict(X_train))
test_mae = mean_absolute_error(y_test, model.predict(X_test))
print("Training MAE:", round(training_mae, 2))
print("Test MAE:", round(test_mae, 2))
```

```
Training MAE: 0.0
Test MAE: 0.0
```

Model Saving

After evaluation, the model was saved using pickle.

In [10]:
```
filename = 'model.pkl'
pickle.dump(model, open(filename, 'wb'))
```

4. Turning Model into Web Application

A web application consisting of a simple web page with a form field that let the web visitors enter the values of the features was developed. After submitting the values of the features, the web application will return the output parameter.

A folder was created for this project called Week 4. The following table is the directory tree inside the folder.

Table 2: Application folder directory

app.py
templates/
index.html
static/
style.css
model.pkl
EinsBotReadings.csv

The sub-directory templates and static are the directories in which Flask looks for HTML and css files, respectively.

App.py

The `app.py` file contains the main code that will be executed by the Python interpreter to run the Flask web application.

Figure 2: App.py

```
import numpy as np
from flask import Flask, request, 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
    '''
    features = [float(x) for x in request.form.values()]
    final_features = [np.array(features)]
    prediction = model.predict(final_features)

    output = round(prediction[0], 2)

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

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

- The application runs as a single module; thus, a new Flask instance is initialized 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.
- Next, the route decorator *(@app.route('/'))* was used to specify the URL that should trigger the execution of the home function.
- The *home* function simply rendered the *index.html* file, which is located in the *templates* folder.

- Inside the *predict* function, the data is accessed and converted to float data type, and the saved model is used to predict the target variable. The result is rounded to two significant figures before being displayed.
- By setting the *debug=True* argument inside the *app.run* method, Flask's debugger was activated.
- Lastly, the *run* function was used to run the application on the server when this script is directly executed by the Python interpreter, which was ensured by using *if* statement with *_name_ == '_main_'*.

Index.html

The following are the contents of the `index.html` file that renders a text form where a user can enter the values of the features.

Figure 3: Index.html

```
<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">
  <h1>Predict EisnBot Output</h1>

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

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

  <br>
  <br>
  {{ prediction_text }}

 </div>


</body>
</html>
```

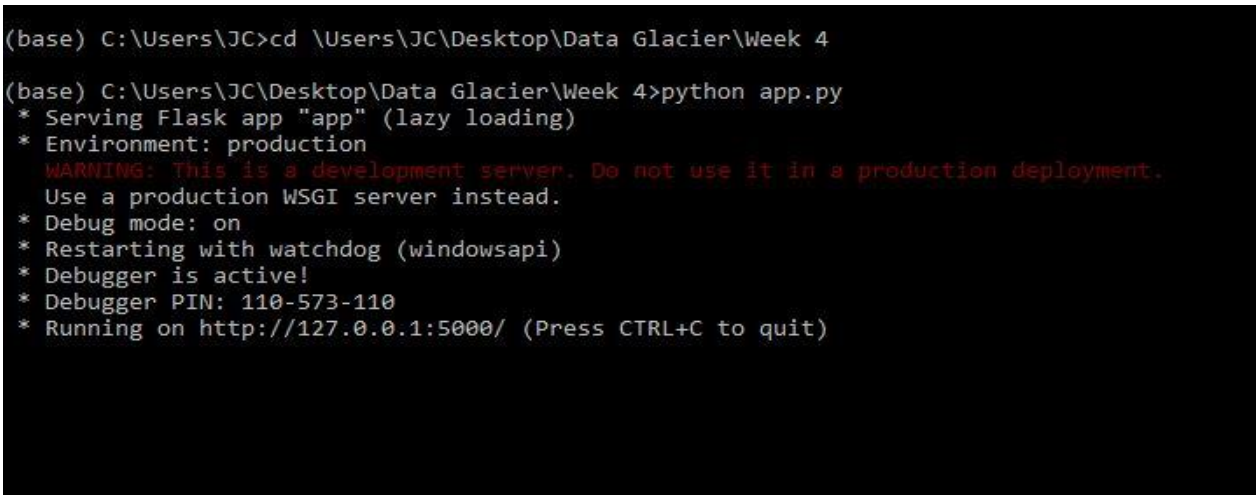
Style.css

In the header section of *index.html*, the *styles.css* file is loaded. CSS determines the look of HTML documents. *styles.css* is saved in a sub-directory called *static*, which is the default directory where Flask looks for static files.

Running Procedure

Once the above files are prepared, one can run the API by executing the command from the terminal.

Figure 4: Command execution

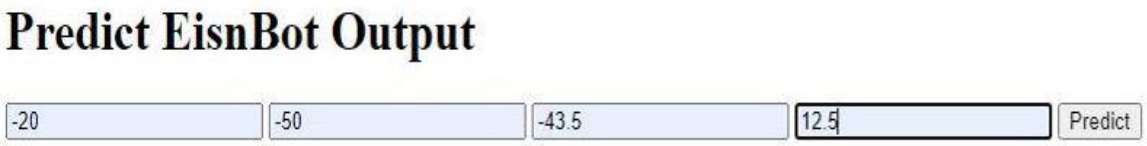


Now, a user can open a web browser and navigate to http://127.0.0.1:5000/ and see the following simple website.

Figure 5: EinsBot output prediction website page

The user can enter the values of each feature in the form.

Figure 6: Feature values



After entering the values, the user can click the predict button and the website returns the predicted value as shown below.

Figure 7: Output parameter prediction