Modeling Challenge - Part 3
Mico Ellerich Comia

---

1. Python script for Flask API Endpoint

The following Python script creates a prediction endpoint using the Flask API. The endpoint accepts inputs formatted in a JSON format and returns the predicted output. Prediction can be processed by running the script, then sending the POST method request to the specified server and endpoint (e.g. [localserver]:[port]/predict),

```
1   # Mico Ellerich M. Comia
2   # Reference: https://www.kdnuggets.com/2019/01/build-api-machine-learning-model-using-flask.html
3
4   from flask import Flask, request, redirect, url_for, flash, jsonify
5   import numpy as np
6   import pandas as pd
7   import joblib
8   import json
9
10  app = Flask(__name__)
11
12  @app.route('/predict', methods=['POST'])
13  def predict():
14      data = request.get_json()
15      prediction = svm_model.predict([np.array(list(data.values()))])
16      output = {"Y": int(prediction[0])}
17
18      return jsonify(output)
19
20  if __name__ == '__main__':
21      svm_model = joblib.load("model/svm_0520-1547.sav")
22      app.run(debug=True, host='0.0.0.0')
23
```

The script works by loading the saved prediction model from Part 2 of the challenge using Joblib's load method. Using values from the user, we use the loaded model to output a prediction which we then return to the user. An explanation of the script line by line can be found below:

- **Line 21:** Loading of saved SVM model using joblib.
- **Line 12:** Definition of endpoint for the POST request. In this case, we used "/predict."
- **Line 14:** Getting the payload from the POST request and converting it to a dictionary.
- **Line 15:** Using the loaded model to create a prediction
- **Line 16:** Formatting the output as a dictionary and casting the prediction as int.
- **Line 18:** Returning the prediction to client in JSON format

2. Testing the Endpoint using Postman

To test if the endpoint is working, we use values similar to that of the test set's. Particularly, rows 1 and 2 which should return an output of 1 and 0, respectively. In Figure 1, we can see the values of rows 1 and 2 in X_test.csv and y_test.csv.

X_test

X1	X2	X3	X4	X5
-1.0284867952984800	0.9693519708180400	1.0098315622156500	-0.520666159985288	1.207995363789590
-0.24930341080122300	-1.8822820528243400	0.4919960878828170	2.5443960039753100	-1.6725450571819400

y_t

Y
1
0

Figure 1. Values from X_test.csv and Y_test.csv

Passing values similar to those in FIgure 1 to our endpoint, we can see in Figure 3 and 5 that the endpoint has returned the expected values.

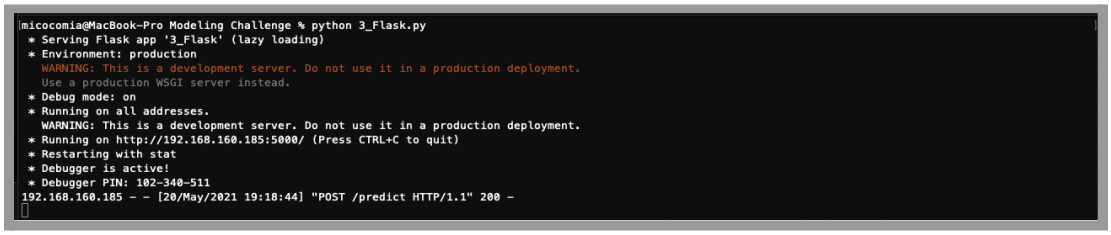

Figure 2. Terminal output for expected output '1' request'

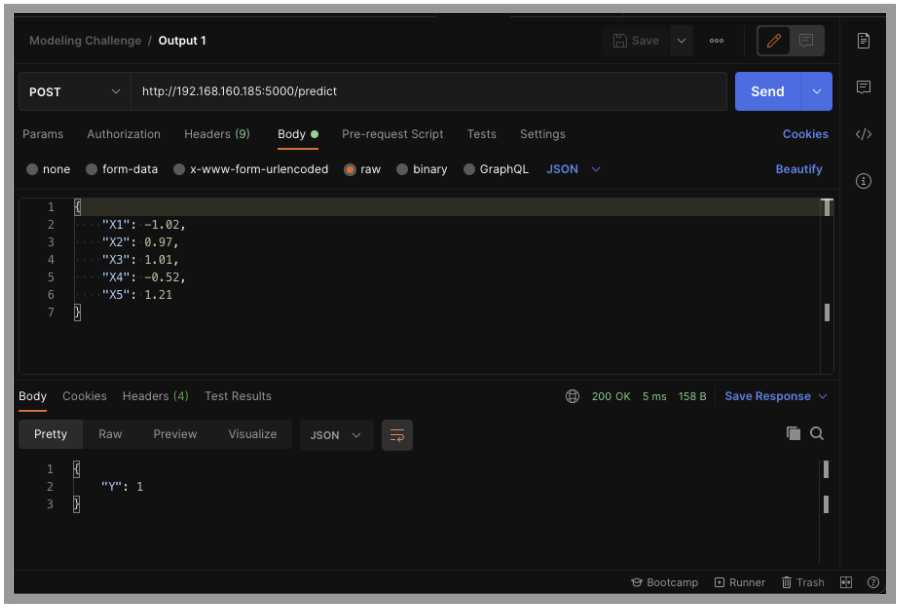

Figure 3. Payload and output for '1' expected output

Figures 2 and 4 show the terminal output once we run the script. It shows us where the Flask app is running (in this case, it's running at http://192.168.160.185:5000/). Additionally, it it outputs whether the received request has been processed successfully or the app has encountered an error.

Figure 4. Terminal output for expected output '0' request

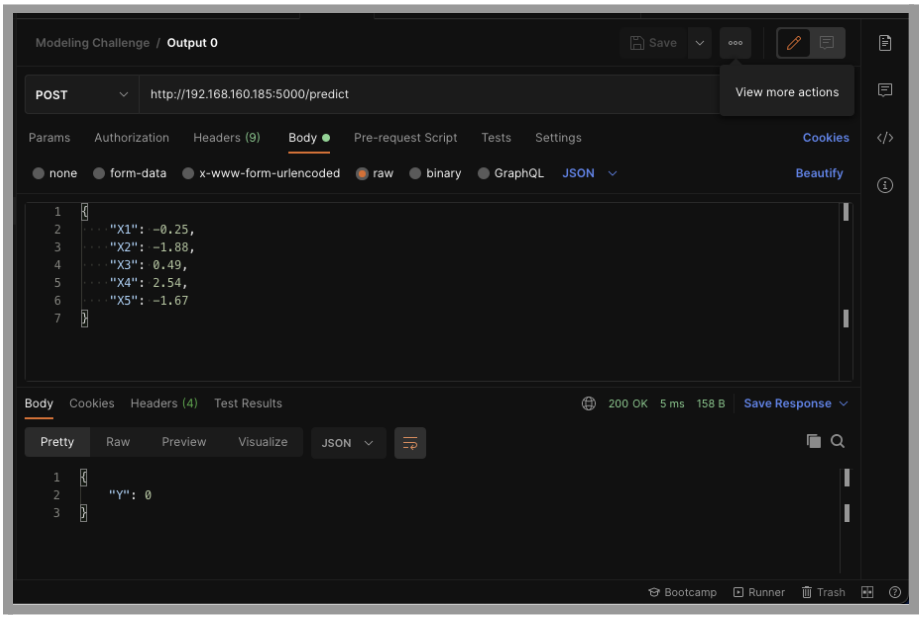

Figure 3. Payload and output for '0' expected output