Model Deployment using Flask

Name: Mohsin Raza

Batch code: LISUM01

Submission date: 03072021

Submitted to: Data Glacier

Overview

- Deploying your basic machine learning model
- Learn how to use Flask to deploy a machine learning model into production
- Model deployment is a core topic in data scientist interviews – so start learning!

Abstract

This project has been written for the beginners of model deployment. With a simple linear regression example, a model was created on Spyder using Flask.

Table of Contents:

What is Model Deployment?

Deployment is the method by which you integrate a machine learning model into an existing production environment to make practical business decisions based on data. In this way, we turn the model we have created into a product. At the same time, we offer the product to the user side.

What is Flask?

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies, and several common framework-related tools. The only feature that distinguishes Flask from other frameworks is that it is very easy to use.

Installing Flask on your Machine

Installing Flask is simple and straightforward. I generally use pip installed.

```
# If you are using pip
$ pip install flask

# For Linux
$ sudo apt-get install python3-flask
```

If you want to work with the latest Flask code before it’s released, install or update the code from the master branch:

```
# Living on the edge
$ pip install -U https://github.com/pallets/flask/archive/master.tar.gz
```

That's it. We are ready to deploying your machine learning model.

Setting up the Project WorkFlow

1. Model Building
2. Save the model and setup app
3. Webpage Template
4. Predict class and send results

Build Machine Learning Model

I prefer to work on Jupyter Notebook. — Our dataset has 25 rows and 2 columns. Let's take a look at what our dataset actually looks like. To do this, use the head() method:

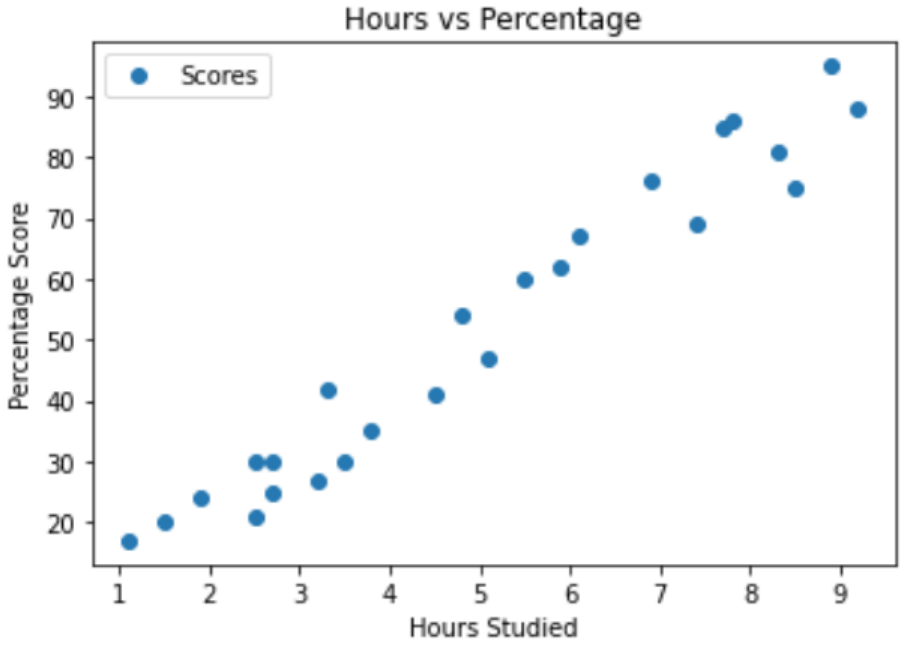
```
In [3]: scores.head()
```

Out[3]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

Let's plot our data points on 2-D graph to eyeball our dataset and see if we can manually find any relationship between the data.

```
In [4]: scores.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



Now that we have our attributes and labels, the next step is to split this data into training and test sets. We'll do this by using Scikit-Learn's built-in train_test_split() method:

```
In [6]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

In [7]: from sklearn.linear_model import LinearRegression
        regressor = LinearRegression()
        regressor.fit(X_train, y_train)

Out[7]: LinearRegression()
```

To retrieve the intercept and For retrieving the slope (coefficient of x):

```
In [8]: print(regressor.intercept_)

2.018160041434683

In [9]: print(regressor.coef_)

[9.91065648]
```

Making Predictions: Now that we have trained our algorithm, it's time to make some predictions.

```
In [10]: y_pred = regressor.predict(X_test)

In [11]: df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
         df
```

Out[11]:

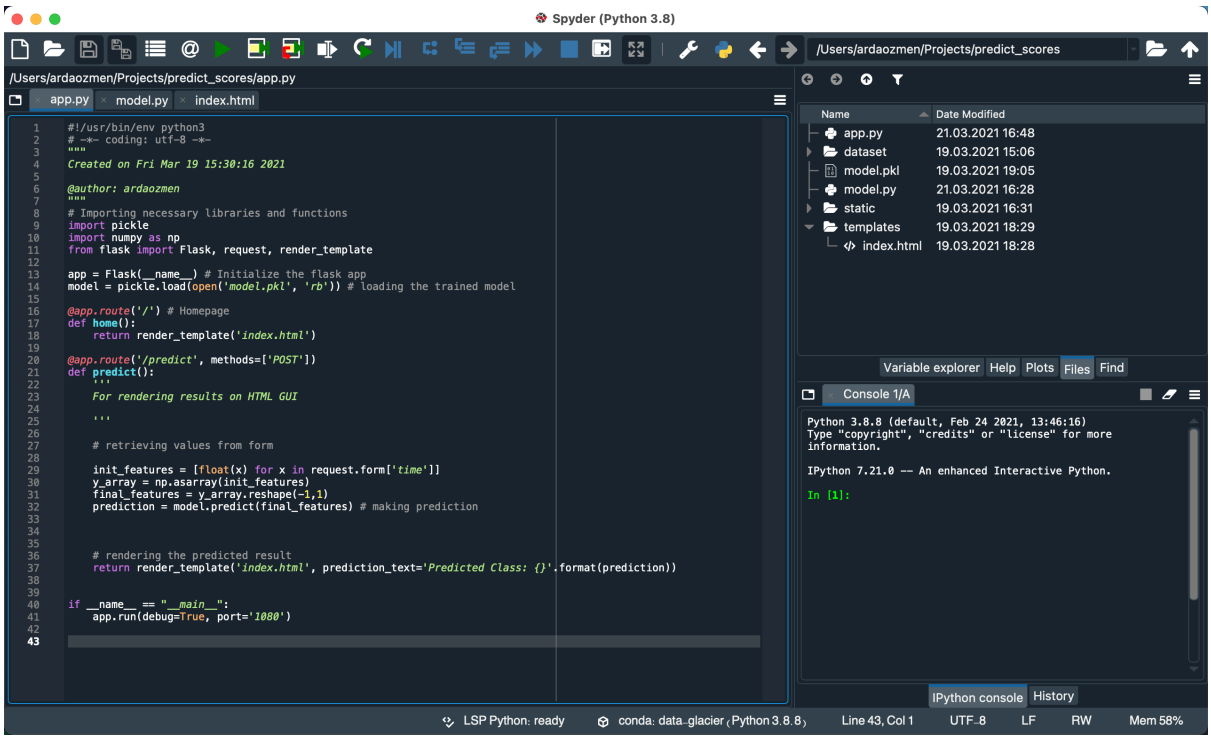
	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

Testing and Proofing:

```
In [13]: my_score = 5

In [14]: y_array = np.asarray(my_score)

In [15]: regressor.predict(y_array.reshape(-1,1))
Out[15]: array([51.57144244])

In [16]: (5 * 9.91065648) + 2.018160041434683
Out[16]: 51.571442441434684
```

Project Snapshots

Spyder usage



Save the Model (model.py)

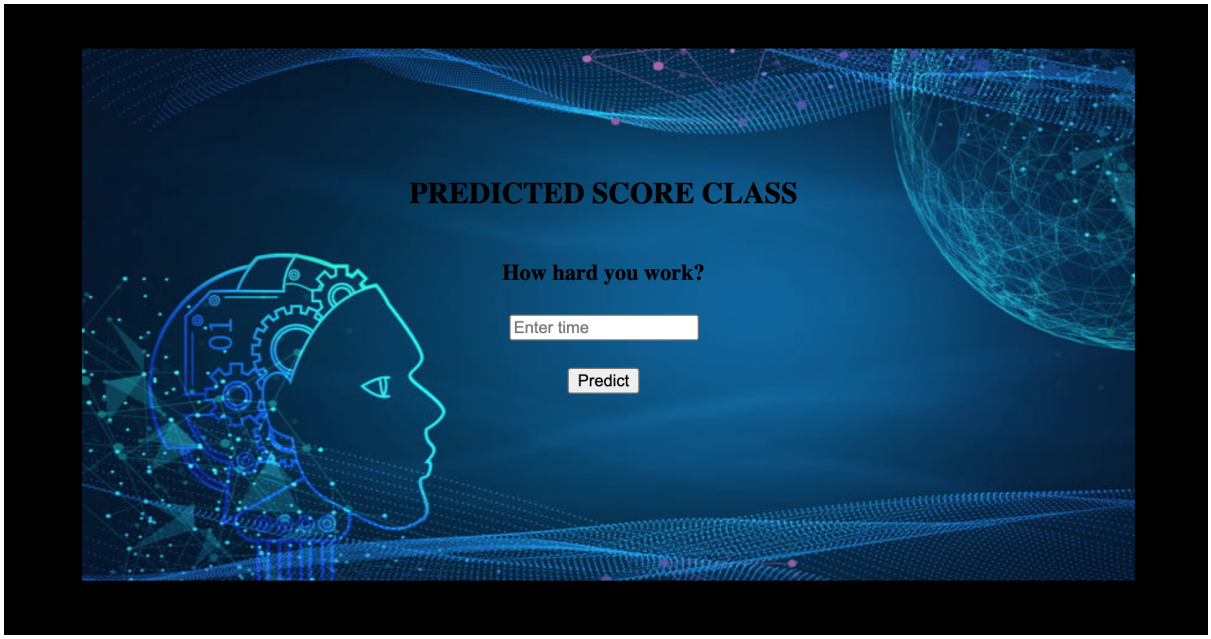
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 19 14:33:31 2021

@author: ardaozmen
"""
# Importing necessary libraries
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Reading the data
scores = pd.read_csv('dataset/student_scores.csv')
X = scores.iloc[:, :-1].values
y = scores.iloc[:, :1].values

# Training Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)

pickle.dump(regressor, open('model.pkl', 'wb'))

```

Connect the Webpage with the Model (app.py)

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 19 15:30:16 2021

@author: ardaozmen
"""
# Importing necessary libraries and functions
import pickle
import numpy as np
from flask import Flask, request, render_template

app = Flask(__name__) # Initialize the flask app
model = pickle.load(open('model.pkl', 'rb')) # loading the trained model

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

@app.route('/predict', methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI

    '''

    # retrieving values from form

    init_features = [float(x) for x in request.form['time']]
    y_array = np.asarray(init_features)
    final_features = y_array.reshape(-1,1)
    prediction = model.predict(final_features) # making prediction



    # rendering the predicted result
    return render_template('index.html', prediction_text='Predicted Class: {}'.format(prediction))


if __name__ == "__main__":
    app.run(debug=True, port='1080')

```

Working of the Deployed Model

We have successfully started the Flask server! Open your browser and go to this address – http://127.0.0.1:1080/. You will see that the Flask server has rendered the default template.



References:

- https://towardsdatascience.com/how-to-easily-deploy-machine-learning-models-using-flask-b95af8fe34d4
- https://medium.datadriveninvestor.com/deploy-your-machine-learning-model-using-flask-made-easy-now-635d2f12c50c