

Model Deployment using AWS

Parth Shah
LISUM01

The Dataset:

The dataset used for this deliverable can be found at the URL

<https://www.kaggle.com/mohansacharya/graduate-admissions>.

The Model:

Below is the code used to train and serialize the model into a pickle file. Pickle was used as it is quite simple to use and understand.

```
model.py
1  import pandas as pd
2  import numpy as np
3  from sklearn.linear_model import LinearRegression
4  from sklearn.model_selection import train_test_split
5  import pickle
6
7  data = pd.read_csv('dataset.csv')
8  X = np.array(data.iloc[:,0:3])
9  y = np.array(data.iloc[:,3])
10
11  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
12
13  lm = LinearRegression()
14  lm.fit(X_train, y_train)
15
16  filename = 'FlaskAPI/model.pkl'
17  pickle.dump(lm, open(filename, 'wb'))
```

Model Deployment:

Once we have our model stored in the pickle file, we can now go on and deploy the model using Flask. It is in this step where we de-serialize the model back into a python object so that we can send some unseen data into our model, through our webpage, and predict an output.

The python script for the deployment of our model is shown below.

```
# importing the necessary dependencies
from flask import Flask, render_template,
request, jsonify from flask_cors import CORS, cross_origin
import pickle

app = Flask(__name__) # initializing a flask app
@app.route('/', methods=['GET']) # route to display the home page
@cross_origin() def homePage():
    return render_template("index.html")

@app.route('/predict', methods=['POST', 'GET']) # route to show the
predictions in a web UI
@cross_origin() def
    index():
        if request.method == 'POST':
            try:
                # reading the inputs given by the user
                gre_score=float(request.form['gre_score'])
                toefl_score = float(request.form['toefl_score'])
                university_rating =
float(request.form['university_rating'])
                sop =
float(request.form['sop'])
                lor =
float(request.form['lor'])
                cgpa =
float(request.form['cgpa'])
                is_research =
request.form['research']
                if(is_research=='yes'):
                    research=1
                else:
                    research=0
                filename = 'finalized_model.pickle'
                loaded_model = pickle.load(open(filename, 'rb')) # loading the model
                file from the storage
                # predictions using the loaded model file

prediction=loaded_model.predict([[gre_score,toefl_score,university_rati
ng,sop,lor,cgpa,research]])
                print('prediction is', prediction) #
                showing the prediction results in a UI
                return
render_template('results.html',prediction=round(100*prediction[0]))
except Exception as e:
    print('The Exception message is:
    ',e)
    return 'something is wrong.' # return
    render_template('results.html')
    else:
        return
        render_template('index.html')

if __name__ == "__main__":
    #app.run(host='127.0.0.1', port=8081, debug=True)
    app.run(debug=True) # running the app
```

HTML Webpage:

Now that our Flask app is ready, we needed an interface to interact with the user and get the data needed to perform predictions. For this, an HTML was used. Below is the HTML code written to generate the page.

Landing Page Source Code:

```
• {% extends 'base.html' %}

{% block head %}

<title>Search Page</title>
<link rel="stylesheet" href="{{ url_for('static',
filename='css/style.css') }}">
{% endblock %}

{% block body %}
<div class="content">
    <h1 style="text-align: center">Predict Your chances for
    Admission</h1>

    <div class="form">
        <form action="/predict" method="POST">
            <input type="number" name="gre_score" id="gre_score"
placeholder="GRE Score">
            <input type="number" name="toefl_score" id="toefl_score"
placeholder="TOEFL Score">
            <input type="number" name="university rating"
```

Output Page Source Code:

```
. <!DOCTYPE html>
  <html lang="en" >

  <head>
    <meta charset="UTF-8">
    <title>Review Page</title>

    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/normalize/5.0.0/normali
ze.min.css">

    <link rel="stylesheet" href="./style.css">
    <link rel="stylesheet" href="{{ url for('static',
filename='css/style.css') }}">

  </head>

  <body>

    <div class="table-users">
      <div class="header">Prediction</div>

      <p>Your chance for admission is {{prediction}} percent</p>
    </div>

  </body>

</html>
```

Deployment to AWS:

- The python application file should be named application.py
- Create a requirements.txt using pip freeze > requirements.txt from the project folder
- Create a folder '.ebextensions' and create a file 'python.config' inside it. Make sure to populate the content of python.config, as shown above.
- Create the zip file from the project folder itself.

.ebextensions	12/17/2019 7:38 PM	File folder	
.elasticbeanstalk	12/17/2019 7:18 PM	File folder	
.git	12/17/2019 7:25 PM	File folder	
.idea	12/17/2019 8:09 PM	File folder	
static	12/17/2019 7:18 PM	File folder	
templates	12/17/2019 7:18 PM	File folder	
application	12/17/2019 7:20 PM	Python File	2 KB
finalized_model.pickle	12/16/2019 7:14 PM	PICKLE File	1 KB
MLforAWS	12/17/2019 7:46 PM	WinRAR ZIP archive	10,467 KB
requirements	12/16/2019 10:30 PM	Text Document	1 KB
train	12/12/2019 9:46 PM	Python File	1 KB

Deployment Process


- Go to <https://aws.amazon.com/> and create an account if already don't have one.
- Go to the console and go to the 'Build a web app' section and click it.

Build a solution

Get started with simple wizards and automated workflows.


Launch a virtual machine

With EC2
2-3 minutes




Build a web app

With Elastic Beanstalk
6 minutes




Build using virtual servers

With Lightsail
1-2 minutes




Register a domain

With Route 53
3 minutes




Connect an IoT device

With AWS IoT
5 minutes



Start migrating to AWS

With CloudEndure Migration
1-2 minutes



[► See more](#)

Give the name of the application, give platform as python, and select the option to upload your code.

Create a web app

Create a new application and environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

Application information

Application name

Up to 100 Unicode characters, not including forward slash (/).

☐ Application tags

Base configuration

Platform

Choose [Configure more options](#) for more platform configuration options.

Application code

☐ Sample application

Get started right away with sample code.

☒ Upload your code

Upload a source bundle from your computer or copy one from Amazon S3.

ZIP or WAR

☐ Application code tags

Final Result:

Predict Your chances for Admission

GRE Score	TOEFL Score	University Rating	
SOP Score	LOR Score	CGPA	Yes ▾
<input type="button" value="Predict"/>			

