Data Science Intern at Data Glacier

Deployment on Flask

Name: Yanjun Lin

Batch Code: LISUM11

Date: 2022/07/18

Submitted to: Data Glacier

Table of Contents:

1. Introduction

This project is going to demonstrate the deployment process of FanaticFi – NBA Draft Rank Prediction empowered by Support Vector Machines (SVM) machine learning model using the Flask application. This user-friendly web application is going to predict whether NBA rookie's draft rank is within the top 15 or below based on given player's statistics.

The process includes building a machine learning model, save the model, then create an API for the model, develop HTTP templates with css styles. Finally, utilize predictive capabilities of the machine learning model through HTTP requests using Flask on a local website.

2. Data Information

The historical NBA draft data came from https://www.sports-reference.com (*See Complete Dataset Overview*). Only a few selected most important aspects were used towards Machine Learning model development (*See Final Selection of Data Frame*).

Complete Dataset Overview

	DraftYr	Rk	Pk	Tm	Pos	Player	College	Contract Worth	Yrs	G	...	WS	WS/48	BPM	VORP	MPG	PPG	RPG	APG	playerurl	DraftYear
0	1990	1	1	BRK	NaN	Derrick Coleman	Syracuse	nan	15.00	781.00	...	64.30	0.12	1.40	22.30	33.20	16.50	9.30	2.50	https://www.sports-reference.com/cbb/players/d...	1990
1	1990	2	2	OKC	NaN	Gary Payton	Oregon State	nan	17.00	1335.00	...	145.50	0.15	3.30	62.50	35.30	16.30	3.90	6.70	https://www.sports-reference.com/cbb/players/g...	1990
2	1990	3	3	DEN	NaN	Mahmoud Abdul-Rauf	LSU	nan	9.00	586.00	...	25.20	0.08	-0.80	4.50	26.70	14.60	1.90	3.50	https://www.sports-reference.com/cbb/players/m...	1990
3	1990	4	4	ORL	NaN	Dennis Scott	Georgia Tech	nan	10.00	629.00	...	33.40	0.09	0.20	9.90	28.60	12.90	2.80	2.10	https://www.sports-reference.com/cbb/players/d...	1990
4	1990	5	5	CHA	NaN	Kendall Gill	Illinois	nan	15.00	966.00	...	47.80	0.08	0.10	15.80	30.50	13.40	4.10	3.00	https://www.sports-reference.com/cbb/players/k...	1990

5 rows × 27 columns

Final Selection of Data Frame

	Rk	MPG	PPG	RPG	APG
0	1	33.2	16.5	9.3	2.5
1	2	35.3	16.3	3.9	6.7
2	3	26.7	14.6	1.9	3.5
3	4	28.6	12.9	2.8	2.1
4	5	30.5	13.4	4.1	3.0

2.1 Attributes Information

The selected collections of player statistics aspects are considered the most influential attributes when it comes to drafting NBA rookie players. Thus, are used to build the model.

Attributes	Explanations
RK	Draft rank from historical data
MPG	Average munities per game played
PPG	Average points scored per game
RPG	Average rebounds per game
APG	Average assists per game

3. Building SVM Model

3.1 Import Required Libraries and Dependencies

```python
# Imports
import pandas as pd
from pathlib import Path
from sklearn import svm
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
```

3.2 Data Import and Preprocessing

A simplified version of dataset was imported from a csv formatted file. *Group* column was created to serve as the Target for the SVM model. The prediction is simply a binary classification case since only two categories are defied within the Target (1: Rank Top 15, 0: Rank 16 and below).

```python
# Upload cc_default.csv to Colab
# from google.colab import files

# csv_file = files.upload()

# Load data
df = pd.read_csv(Path('./NBA_20_yr.csv'))
df.head()
```

	Rk	MPG	PPG	RPG	APG
0	1	33.2	16.5	9.3	2.5
1	2	35.3	16.3	3.9	6.7
2	3	26.7	14.6	1.9	3.5
3	4	28.6	12.9	2.8	2.1
4	5	30.5	13.4	4.1	3.0

```python
# Create a column to hold Group data
df.loc[:,'Group'] = 0
df.fillna(0, inplace = True)
df
```

	Rk	MPG	PPG	RPG	APG	Group
0	1	33.2	16.5	9.3	2.5	0
1	2	35.3	16.3	3.9	6.7	0
2	3	26.7	14.6	1.9	3.5	0
3	4	28.6	12.9	2.8	2.1	0
4	5	30.5	13.4	4.1	3.0	0

```python
# Create a column to hold Group data
df.loc[:,'Group'] = 0
df.fillna(0, inplace = True)
df
```

	Rk	MPG	PPG	RPG	APG	Group
0	1	33.2	16.5	9.3	2.5	0
1	2	35.3	16.3	3.9	6.7	0
2	3	26.7	14.6	1.9	3.5	0
3	4	28.6	12.9	2.8	2.1	0
4	5	30.5	13.4	4.1	3.0	0
...	...	...	...	...	...	...
1863	56	3.5	0.5	0.0	0.5	0
1864	57	0.0	0.0	0.0	0.0	0
1865	58	6.5	1.1	1.5	0.1	0
1866	59	0.0	0.0	0.0	0.0	0
1867	60	5.3	1.8	0.9	0.0	0

1868 rows × 6 columns

```python
# Create groups based on ranks
for index, row in df.iterrows():
    if row['Rk'] <= 15:
        df.at[index,'Group'] = 1
    else:
        df.at[index,'Group'] = 0
df
```

	Rk	MPG	PPG	RPG	APG	Group
0	1	33.2	16.5	9.3	2.5	1
1	2	35.3	16.3	3.9	6.7	1
2	3	26.7	14.6	1.9	3.5	1
3	4	28.6	12.9	2.8	2.1	1
4	5	30.5	13.4	4.1	3.0	1
...	...	...	...	...	...	...
1863	56	3.5	0.5	0.0	0.5	0
1864	57	0.0	0.0	0.0	0.0	0
1865	58	6.5	1.1	1.5	0.1	0
1866	59	0.0	0.0	0.0	0.0	0
1867	60	5.3	1.8	0.9	0.0	0

1868 rows × 6 columns

The final dataset then was divided into features (X) and target (y) group, as features contains MPG, PPG, RPG, and APG columns, while as the target contains the Group column.

```python
# Set Feature and Target
X = df.drop(columns=['Rk','Group'])
y = df['Group']
y.value_counts()
```

```
0    1388
1     480
Name: Group, dtype: int64
```

3.3 Data Scaling

After the data frame preprocessing, it was further split into training and testing groups in the default rate of 0.75 : 0.25 using the standard scaler library. Then it was saved as a pickle file for later use.

```python
# Split dataset into train, test datasets
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 1)
# Initiate the scaler
X_scaler = StandardScaler()
# Fit the scaler to the features dataset
X_scaler = X_scaler.fit(X)
# Scale train, test datasets
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

```python
X_test_scaled[0]

array([ 0.59432799,  0.52259442, -0.25369002, -0.0878169 ])
```

```python
import pickle
```

```python
# save the scaler
Standard_Scaler = open("scaler.pkl","wb")
pickle.dump(X_scaler,Standard_Scaler)
Standard_Scaler.close()
# Load the scaler
model = open("scaler.pkl","rb")
new_scaler = pickle.load(model)
new_scaler
```

```python
StandardScaler()
```

3.4 Build and Test Model

Machine learning model then was created with scaled data to be trained to classify rookie draft ranks implementing Support Vector Machine (SVM) under scikit-learn library. After initializing the SVM model, we fit it into the training dataset, and make predictions with testing set. Finally, evaluate the model performance with the classification report.

```python
# From SVM, instantiate SVC classifier model instance
svm_model = svm.SVC(kernel = 'linear', random_state = 0)

# Fit the model to the data using the training data
svm_model = svm_model.fit(X_train_scaled, y_train)

# Use the testing data to make the model predictions
svm_pred = svm_model.predict(X_test_scaled)
```

```python
# Use a classification report to evaluate the model using the predictions and testing data
svm_testing_report = classification_report(y_test, svm_pred)

# Print the classification report
print(svm_testing_report)
```

```
              precision    recall  f1-score   support

           0       0.83      0.94      0.88       357
           1       0.65      0.38      0.48       110

    accuracy                           0.81       467
   macro avg       0.74      0.66      0.68       467
weighted avg       0.79      0.81      0.79       467
```

3.5 Save the Model

Last but not least: save the model as a pickle file for later use.

```python
# save the model
Support_Vector_Machine = open("model.pkl","wb")
pickle.dump(svm_model,Support_Vector_Machine)
Support_Vector_Machine.close()
```

```python
# Load the model
model = open("model.pkl","rb")
new_model = pickle.load(model)
new_model
```

4. Turning Model into Web Application

Develop a user interactive web application that allows the machine learning model to make predictions based on user's input. After a user submitting information about a potential player, the web application will render a result of the player is predicted to be the top 15 picks or not.

Inside the FanaticFi_Flask folder, you will find the file directory as showing below. Each file will be explained in detail in the later part of this document.

FanaticFi_Flask Folder File Directory

app.py
FanaticFi_Flask.ipynb
Model.pkl
Scaler.pkl
NBA_20_yr.csv
templates/   index.html
static/   logo.png
                            styles/   style.css

4.1 App.py

```python
import pickle
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.preprocessing import StandardScaler
from flask import Flask,request,render_template,url_for

app = Flask(__name__)

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

@app.route('/predict',methods=['POST'])
def predict():

    # import pickle file of my model
    model = pickle.load(open('model.pkl', 'rb'))

    # import pickle file of my scaler
    scaler = pickle.load(open('scaler.pkl', 'rb'))

    # For rendering results on HTML GUI
    init_features = [float(x) for x in request.form.values()]
    final_features = np.array([init_features])
    features_scaled = scaler.transform(final_features)
    prediction = model.predict(features_scaled)
    if prediction[0] == 1:
        result="The player you choose will be in the Top 15 picks"
    else:
        result="The player you choose will rank 16 and below"

    return render_template('index.html', prediction_text=result)

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

When run the *app.py* file, then entire Flask web application, includes the machine learning model, HTML file and its style file will all be executed by Python.

- Initialize a new Flask instance with the argument __*name*__, the application will run as a single module to let Flask know that it can find the HTML template folder (*templates*) in the directory
- The route decorator *(@app.route('/'))* is used to specify the URL that should trigger the execution of the home function
- The *home* function renders the *index.html* file, located in the *templates* folder
- Inside the *predict* function, both previously saved model and scaler pickle files are loaded to scale user input data and make predictions
- Use the *POST* method to transport data to server
- Setting __*name*__ == '__*main*__', to make sure we only run the application on the server when this script is directly executed by the Python interpreter
- Setting *debug=True* to activated Flask's debugger when run *app.py*

4.2 Index.html

The following are the contents of the *index.html* file that will render input areas where a user can enter required statistic of a player. In the header section, the *styles.css* file is loaded to render the design of HTML file.

```html
<!DOCTYPE html>
<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='styles/style.css') }}">

</head>

<body>
 <div class="login">
  <h1>Predict NBA Draft Rank</h1>

     <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
        <input type="text" name="MPG" placeholder="Minutes Played Per Game" required="required" />
        <input type="text" name="PPG" placeholder="Points Scored Per Game" required="required" />
        <input type="text" name="RPG" placeholder="Rebounds Per Game" required="required" />
        <input type="text" name="APG" placeholder="Assists Per Game" required="required" />

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

   <br>
   <br>
   {{ prediction_text }}

 </div>
 <img src="{{ url_for('static', filename='logo.png') }}" style="width: 100px;position: absolute;top: 10px;left: 10px;" alt="Logo"/>

</body>
</html>
```

4.3 Style.css

The *styles.css* file determines the look and design of the web application, and it must be saved in a sub-directory called *static*, which is the default directory where Flask looks for static files. Below does not include the entire content of the css file, but an example of how it formats the body section, input format of the user interface.

```css
body {
    width: 100%;
    height:100%;
    font-family: 'Open Sans', sans-serif;
    color: rgb(226, 82, 82);
    font-size: 18px;
    text-align:center;
    letter-spacing:1.2px;
    background: #6c4e4e !important;
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1 );
}
.login {
    position: absolute;
    top: 40%;
    left: 50%;
    margin: -150px 0 0 -150px;
    width:400px;
    height:400px;
}

.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing:1px; text-align:center; }

input {
    width: 100%;
    margin-bottom: 10px;
    background: rgba(0,0,0,0.3);
    border: none;
    outline: none;
    padding: 10px;
    font-size: 13px;
    color: #fff;
    text-shadow: 1px 1px 1px rgba(0,0,0,0.3);
    border: 1px solid rgba(0,0,0,0.3);
    border-radius: 4px;
    box-shadow: inset 0 -5px 45px rgba(100,100,100,0.2), 0 1px 1px rgba(255,255,255,0.2);
    -webkit-transition: box-shadow .5s ease;
```

5. Run Application

After completing all previous steps, we can now run the App using the Terminal command line interface as shown below (make sure you are running from the correct directory and under the correct virtual environment where all dependencies were installed).
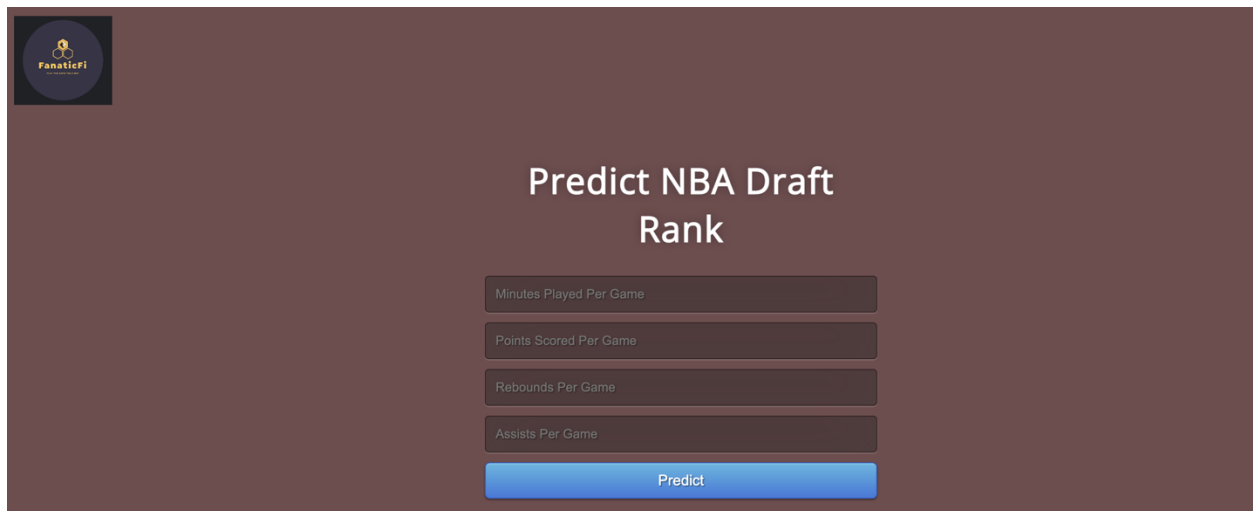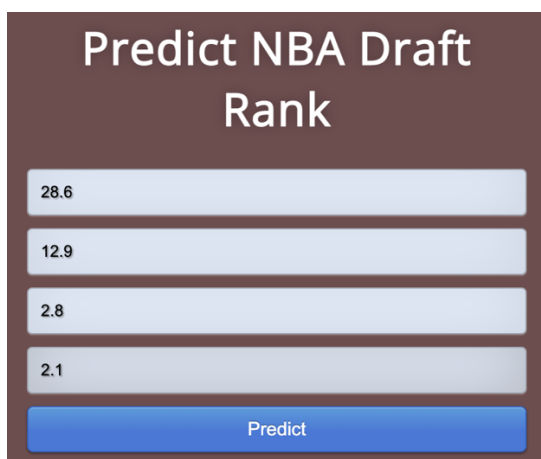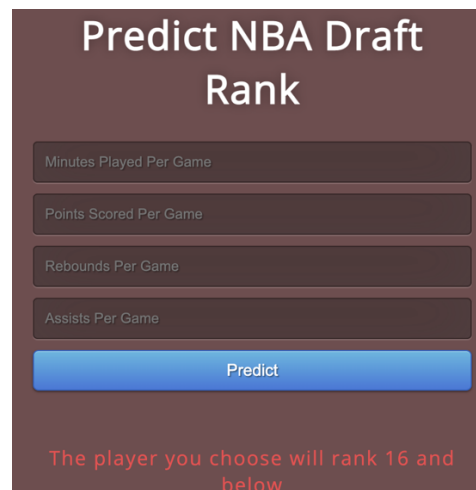
```
(base) yjun_ln@Yjun-Lns-MacBook-Pro FanaticFi_Flask % conda activate dev
(dev) yjun_ln@Yjun-Lns-MacBook-Pro FanaticFi_Flask % python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 688-972-434
127.0.0.1 - - [18/Jul/2022 15:58:26] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jul/2022 15:58:26] "GET /static/logo.png HTTP/1.1" 200
 -
127.0.0.1 - - [18/Jul/2022 15:58:26] "GET /static/styles/style.css HTTP/
1.1" 200 -
```

Then, copy and paste the given web address http://127.0.0.1:5000/ to browser and checkout the result of the API.

Input required statistics of a chosen player and get prediction result.

6. Deploy model on Heroku

To deploy existing API on Heroku, we need to construct 2 additional files: Procfile and requests.txt to help Heroku configure the server that is hosting the application and dependencies to run the application. But before everything, we need to setup a brand-new virtual environment just for this project using $ **conda create --name fanaticfi python=3.7** and activate it with $ **conda activate env_name**. This environment should be activated when working in this project directory/repository for application installations and running commands. Since this is a new environment, all the dependencies will need to be installed via $ **pip install** before they can be imported. Use $ **conda deactivate** to deactivate the virtual environment.

To create the **Procfile** file, we need to first $ **pip install gunicorn** to include it as a dependency, then run $ **echo "web: gunicorn app_basic:app" > Procfile** in the project directory.
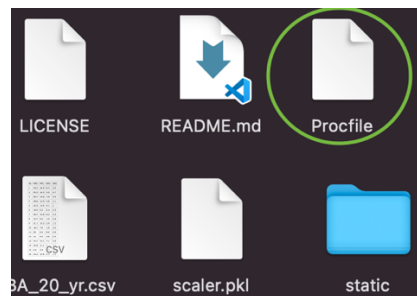
To create the **requirement.txt** file, run the $ **pip freeze > requirements.txt** command in the project directory. This file writes a list of all libraries installed in the active environment to run this project.

6.1 Steps for Model Deployment Using Heroku

First, uploaded all files include the **Procfile** file and **requirements.txt** file to the GitHub repository, then link it with Heroku follow the steps below:

1. Sign up for a **Heroku** account then click on **Create new app** at either location.
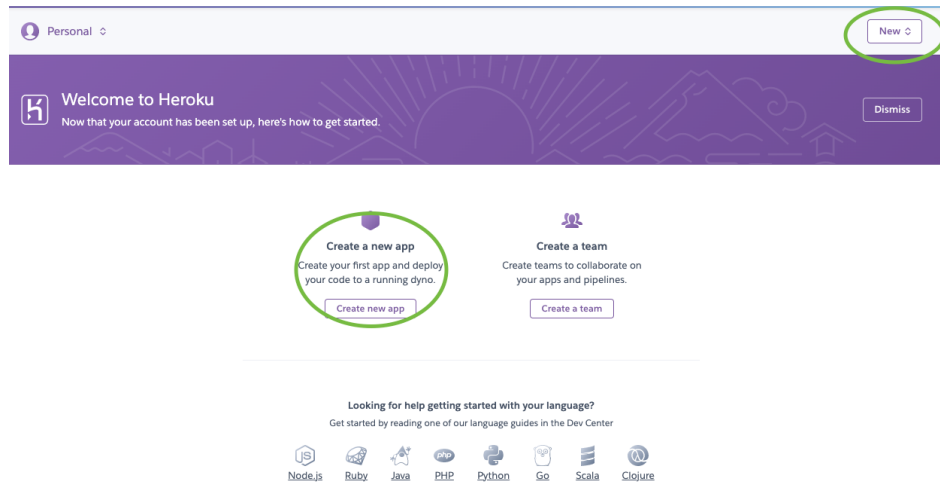
2. Enter **App name and region** then click **Create app**

3. Connect to GitHub repository

4. Deploy branch

This process usually takes 5-10 minutes.

5. View application after deploying the branch

The app is published at https:fanaticfi-api.herokuapp.com/