

Week 4 Assignment

Deployment on Flask

Name: Omar Pablo Jazouli Martínez

Batch code: LICAN01

Submission date: 18/03/2021

Submitted to: Data Glacier

In order to deploy a Machine Learning model we have taken the following four steps.

1 Create and save model

As creating a good and useful model goes beyond the scope of this assignment, we have selected the toy dataset of the Titanic passengers. The aim is to predict whether a passenger survived, taking into account their ticket class, gender and age. A decision tree was used to do so and then the model is saved in disk.

```
data = pd.read_csv('https://raw.githubusercontent.com/laxmimerit/All-CSV-ML-Data-Files-Download/master/titanic.csv')
data.dropna(inplace=True)
use_cols = ["Pclass", "Sex", "Age"]
X_train, X_test, y_train, y_test = train_test_split(
    data[use_cols],
    data['Survived'],
    test_size=0.3,
    random_state=0)
X_train.shape, X_test.shape
# call the model
from sklearn import tree
clf = tree.DecisionTreeClassifier(max_depth=4)
# train the model
model = clf.fit(X_train[["Pclass", "Sex", "Age"]].values, y_train)
# make predictions on train and test set
pred_train = model.predict_proba(X_train[["Pclass", "Sex", "Age"]].values)
pred_test = model.predict_proba(X_test[["Pclass", "Sex", "Age"]].values)
print('Train set')
print('DecisionTree roc-auc: {}'.format(roc_auc_score(y_train, pred_train[:,1])))
print('Test set')
print('DecisionTree roc-auc: {}'.format(roc_auc_score(y_test, pred_test[:,1])))
```

```
Train set
DecisionTree roc-auc: 0.8846861045442606
Test set
DecisionTree roc-auc: 0.8424908424908426
```

```
# save the model to disk
filename = 'titanic_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

2 Flask app

Once we have created and saved the model, it is time to build a flask app to deploy it later on.

```
from flask import Flask,request,render_template
import pickle
import numpy as np

app=Flask(__name__)
model=pickle.load(open("titanic_model.sav", 'rb'))

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

@app.route('/predict',methods=["POST"])
def predict():
    int_features=[int(x) for x in request.form.values()]
    final_features=np.array(int_features).reshape(1,-1)
    prediction=model.predict(final_features)[0]
    if prediction==1:
        output="The passenger survived"
    else:
        output="The passenger did not survived"
    return render_template("index.html",prediction_text=output)

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

This file is called "app.py". In it, we call the file "index.html", which contains the html code of the web application.

3 HTML and CSS templates

The html file has the form to input the data that our model will use to make its prediction (ticket class, gender and age).

```
<!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='style.css')}}">
  </head>
  <body>
    <div class="login">
      <h2>Predict whether a Titanic passenger survived</h2>

      <form action="{{url_for('predict')}}" method="post">
        <label for="fname">Ticket Class (1,2,3):</label><br>
        <input type="number" id="fname" name="fname" min="1" max="3"><br><br>
        <label for="gender">Gender (Male:0, Female:1):</label><br>
        <input type="number" id="gender" name="gender" min="0" max="1"><br><br>
        <label for="age">Age:</label><br>
        <input type="number" id="age" name="age"><br><br>
        <input type="submit" value="PREDICT">

      </form>

      <h3>
        {{prediction_text}}
      </h3>
    </div>
  </body>
</html>
```

It is connected to a css file ("style.css") in order to have a fancy appearance. This file is the one used in the webinar but with the following changes:

```
.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing:1px; text-align:center; }
.login h2 { color: white; text-shadow: 0 0 10px rgba(0,0,0,0); letter-spacing:1px; text-align:center; }
.login form { color: white; text-shadow: 0 0 10px rgba(0,0,0,0); letter-spacing:1px; text-align:left; }
.login h3 { color: white; text-shadow: 0 0 10px rgba(0,0,0,0); letter-spacing:1px; text-align:center; }
```

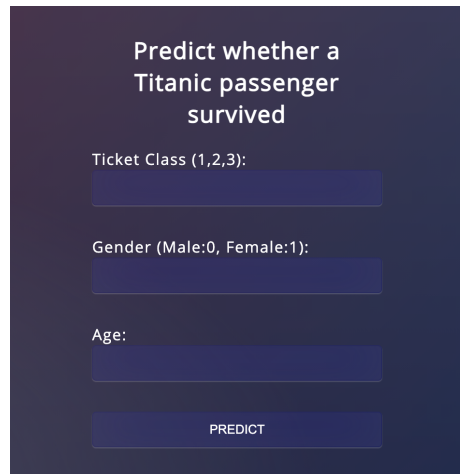
"Index.html" is inside a folder called "templates" and the folder containing "style.css" is called "static".

4 Deploy model

Finally, we are able to deploy our model on a web application. We run the "app.py" file with the command shell.

```
(base) MacBook-Air-de-Omar:Desktop omarjazoulimartinez$ export FLASK_APP=app.py
(base) MacBook-Air-de-Omar:Desktop omarjazoulimartinez$ flask run
* Serving Flask app "app.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

After using the above direction on our browser we will be redirected to the following page:



Predict whether a
Titanic passenger
survived

Ticket Class (1,2,3):

Gender (Male:0, Female:1):

Age:

PREDICT

One can input the ticket class, the gender and the age of the target passenger. The model will return whether they survived or not after clicking on the "PREDICT" button.

Predict whether a Titanic passenger survived

Ticket Class (1,2,3):
1

Gender (Male:0, Female:1):
1

Age:
18

PREDICT

The passenger survived

Figure 1: Survived

Predict whether a Titanic passenger survived

Ticket Class (1,2,3):
1

Gender (Male:0, Female:1):
0

Age:
76

PREDICT

The passenger did not survived

Figure 2: Did not survived