Week 5 Assignment

Cloud and API deployment

Name: Omar Pablo Jazouli Martínez
Batch code: LICAN01
Submission date: 25/03/2021
Submitted to: Data Glacier

The Machine Learning model (Titanic) used in the previous assignment has been deployed on cloud (Heroku) following these 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.

```python
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).

```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='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>
```

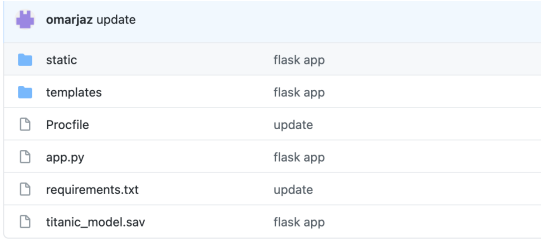
It is connected to a css file ("style.css") in order to have a fancy appearance. This file is the one used in the webinar but with the following changes:

```css
.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 Submit the app to GitHub

We create a repository (titanic_app) containing the flask app, the model, the html code (templates) and the css file (static).



In the root directory, two new files have been added:

1. Procfile: It will tell Heroku how to run the app.

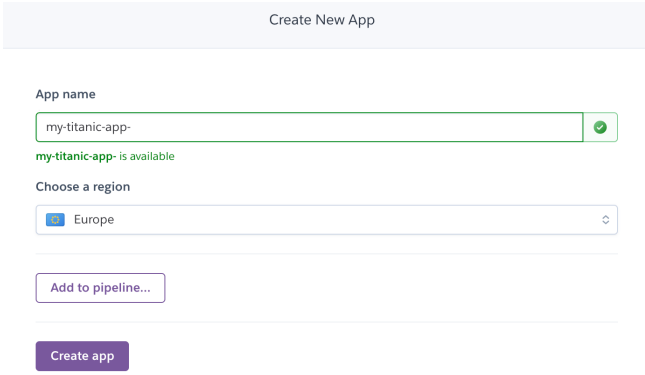
```
web: gunicorn app:app
```

2. Requirements.txt: It will tell Heroku that the project needs all these libraries to the run the app correctly.

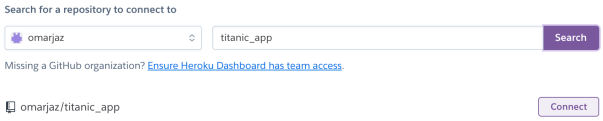
```
flask
numpy
sklearn
gunicorn
joblib
```

5 Deployment on Heroku

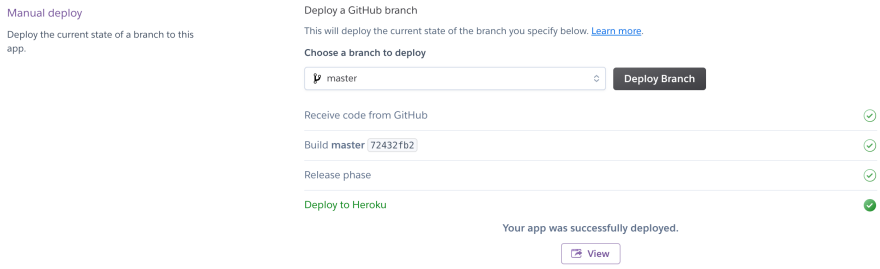
Firstly, a new app was created on Heroku:



Now, it is time to connect it to our GitHub repository:

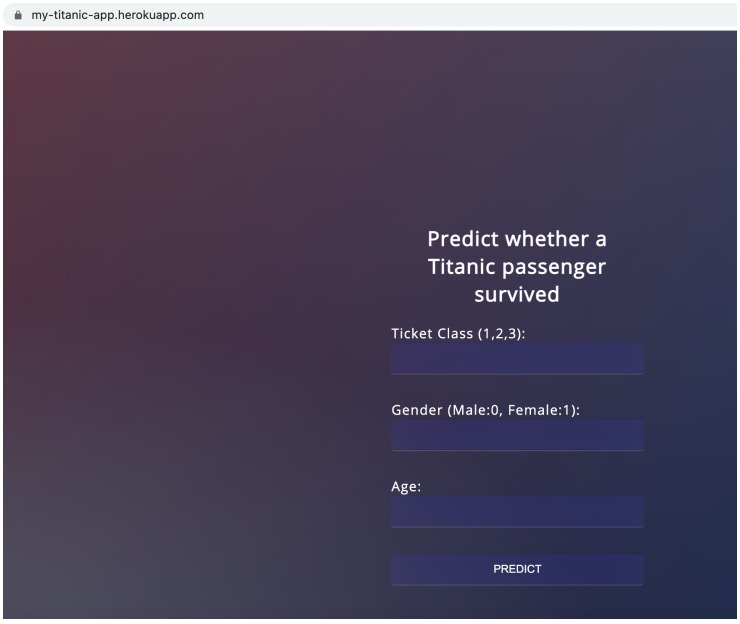


We choose the manual deploy option:



6 Use the deployed model

Finally, one can check that the model has been deployed correctly on the Heroku app:



We can now 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.

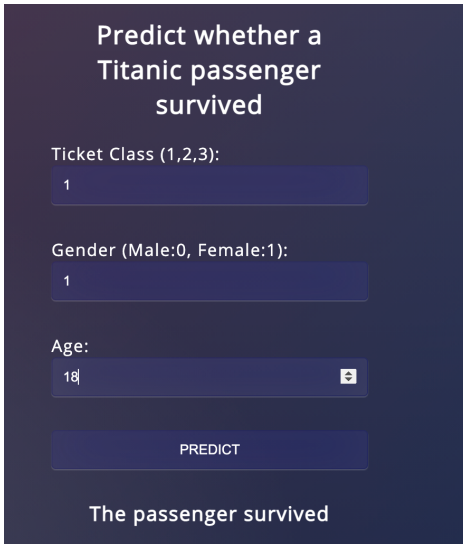


Figure 1: Survived

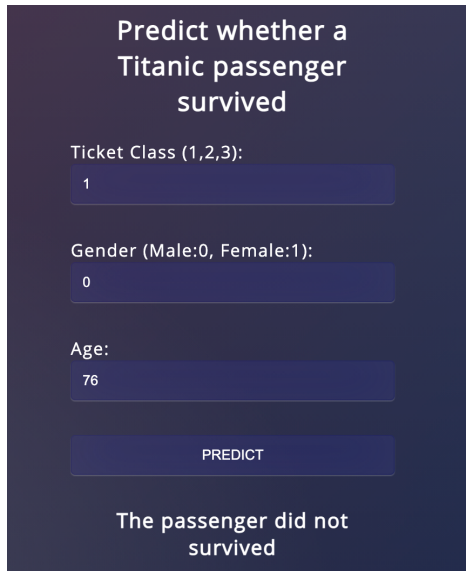


Figure 2: Did not survived