Model deployment on Heroku

Nikola Andrejić

Intern at Data Glacier

LISUM10

Contents

1 CREATING AND TRAINING THE MODEL

1.1 BASIC PYTHON LIBRARIES

We start with importing numpy and pandas as the basic libraries for handling our dataset.

In [1]:
```
#Importing the libraries
import numpy as np
import pandas as pd
```

1.2 THE DATASET

The dataset we use for training our model is the IMDB Dataset of 50K Movie Reviews. This dataset contains the review and the label "sentiment" that is either "positive" or "negative".

In [2]:
```
#Importing the dataset
df = pd.read_csv("IMDB Dataset.csv")
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

In [4]:
```
df.head()
```

Out[4]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

1.3 FEATURE PREPARATION

We notice that the review contains some html tags.

In [5]:
```
#We need to convert html to text
df["review"][1]
```

Out[5]:
```
'A wonderful little production. <br /><br />The filming technique is
very unassuming- very old-time-BBC fashion and gives a comforting, a
nd sometimes discomforting, sense of realism to the entire piece. <b
r /><br />The actors are extremely well chosen- Michael Sheen not on
ly "has got all the polari" but he has all the voices down pat too!
You can truly see the seamless editing guided by the references to W
illiams\' diary entries, not only is it well worth the watching but
it is a terrificly written and performed piece. A masterful producti
on about one of the great master\'s of comedy and his life. <br /><b
r />The realism really comes home with the little things: the fantas
y of the guard which, rather than use the traditional \'dream\' tech
niques remains solid then disappears. It plays on our knowledge and
our senses, particularly with the scenes concerning Orton and Halliw
ell and the sets (particularly of their flat with Halliwell\'s mural
s decorating every surface) are terribly well done.'
```

To deal with this, we convert this column to text using the html2text function from the html2text module.

In [6]:
```
from html2text import html2text
```

In [7]:
```
#Converting html to text
html2text(df.loc[1,"review"])
```

Out[7]:
```
'A wonderful little production.  \n  \nThe filming technique is very
unassuming- very old-time-BBC fashion and gives\na comforting, and s
ometimes discomforting, sense of realism to the entire\npiece.  \n
\nThe actors are extremely well chosen- Michael Sheen not only "has
got all the\npolari" but he has all the voices down pat too! You can
truly see the seamless\nediting guided by the references to Williams
\' diary entries, not only is it\nwell worth the watching but it is
a terrificly written and performed piece. A\nmasterful production ab
out one of the great master\'s of comedy and his life.  \n  \nThe re
alism really comes home with the little things: the fantasy of the g
uard\nwhich, rather than use the traditional \'dream\' techniques re
mains solid then\ndisappears. It plays on our knowledge and our sens
es, particularly with the\nscenes concerning Orton and Halliwell and
the sets (particularly of their flat\nwith Halliwell\'s murals decor
ating every surface) are terribly well done.\n\n'
```

In [8]:
```
#Now for the whole dataset
df["review"]=df["review"].apply(lambda rev: html2text(rev))
```

1.4 CREATING THE MODEL

We start with noticing that the dataset is perfectly balanced in terms of labels.

In [9]:
```
#This dataset is perfectly balanced
df["sentiment"].value_counts()
```

Out[9]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

Next we separate the features (the review) and the label (the sentiment) and then also divide the dataset into train and test subsets.

In [10]:
```
#Separate features (which is a review) from a label (sentiment)
X=df["review"]
y=df["sentiment"]
```

In [11]:
```
#Perform a train test split with 20% test size
from sklearn.model_selection import train_test_split
```

In [12]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,
                                                 random_state=101)
```

We create our model as a pipeline that has two steps. The first step performs the tifidf vectorization with english stop words and the second step applies the Naive Bayes itself. We train the model on the training set and make predictions on the test set.

In [13]:
```
#Create a model with tfidf vectorization and Naive Bayes classifier.
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
steps=[("tfidf",TfidfVectorizer(stop_words="english")),
       ("nb",MultinomialNB())]
pipe=Pipeline(steps)
```

In [14]:
```
#Train the model
pipe.fit(X_train,y_train)
```

Out[14]:
```
Pipeline(steps=[('tfidf', TfidfVectorizer(stop_words='english')),
                ('nb', MultinomialNB())])
```

In [15]:
```
#Predict the labels for the test set
y_predict = pipe.predict(X_test)
```

To evaluate our model we print the classification report and the confusion matrix.

In [16]:
```
#Evaluate the model by observing the classification report
#and the confusion matrix
from sklearn.metrics import (plot_confusion_matrix,
                             classification_report)
```

In [17]:
```
print(classification_report(y_test,y_predict))
```

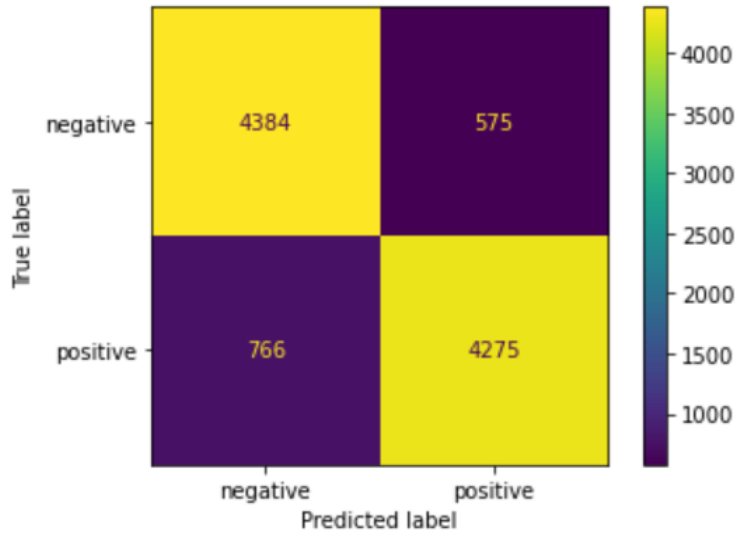
```
              precision    recall  f1-score   support

    negative       0.85      0.88      0.87      4959
    positive       0.88      0.85      0.86      5041

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

In [18]:
```
plot_confusion_matrix(pipe,X_test,y_test)
```

Out[18]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x
26763b905e0>
```



Finally, we save our model with pickle.

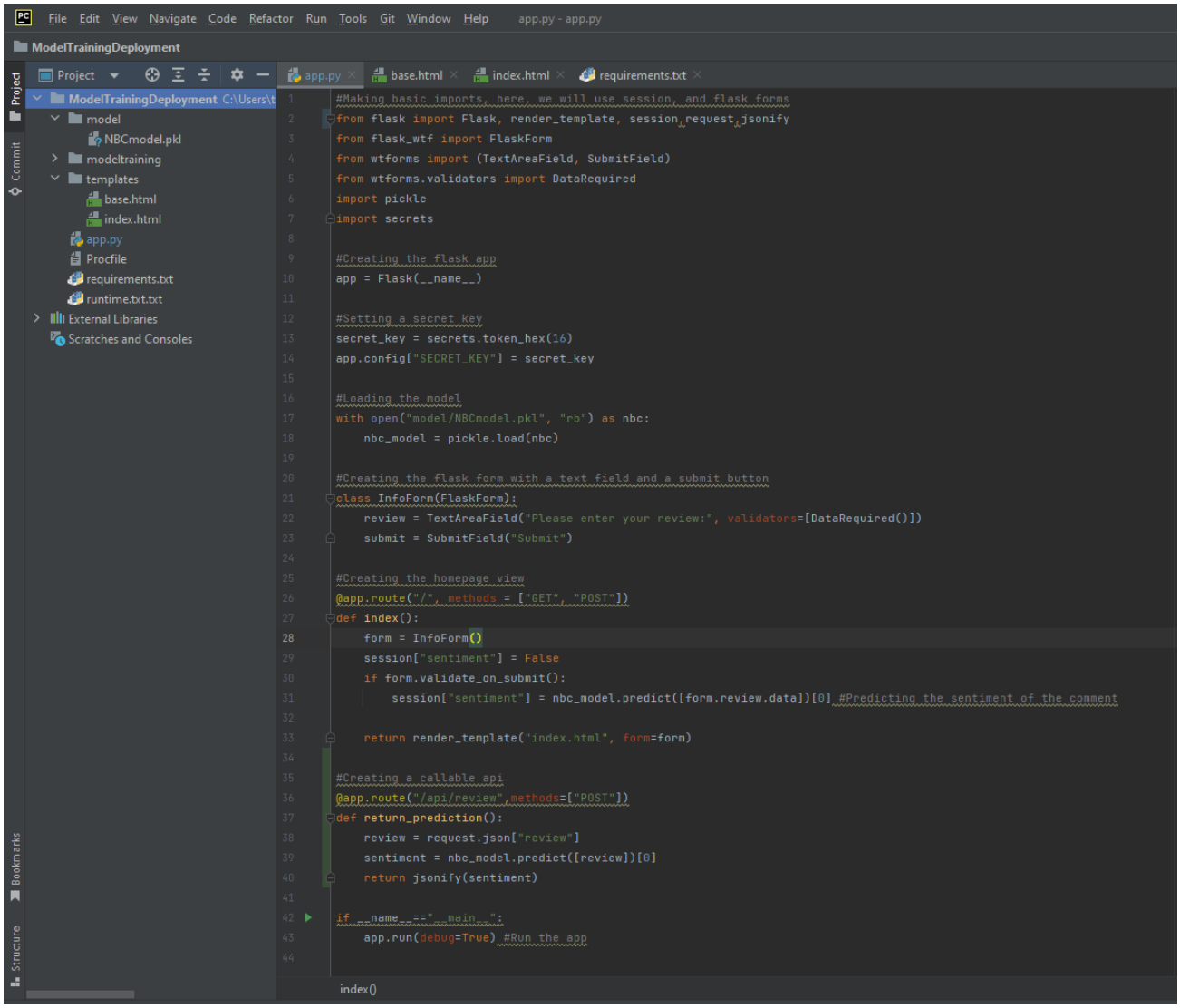
In [19]:
```
import pickle
```

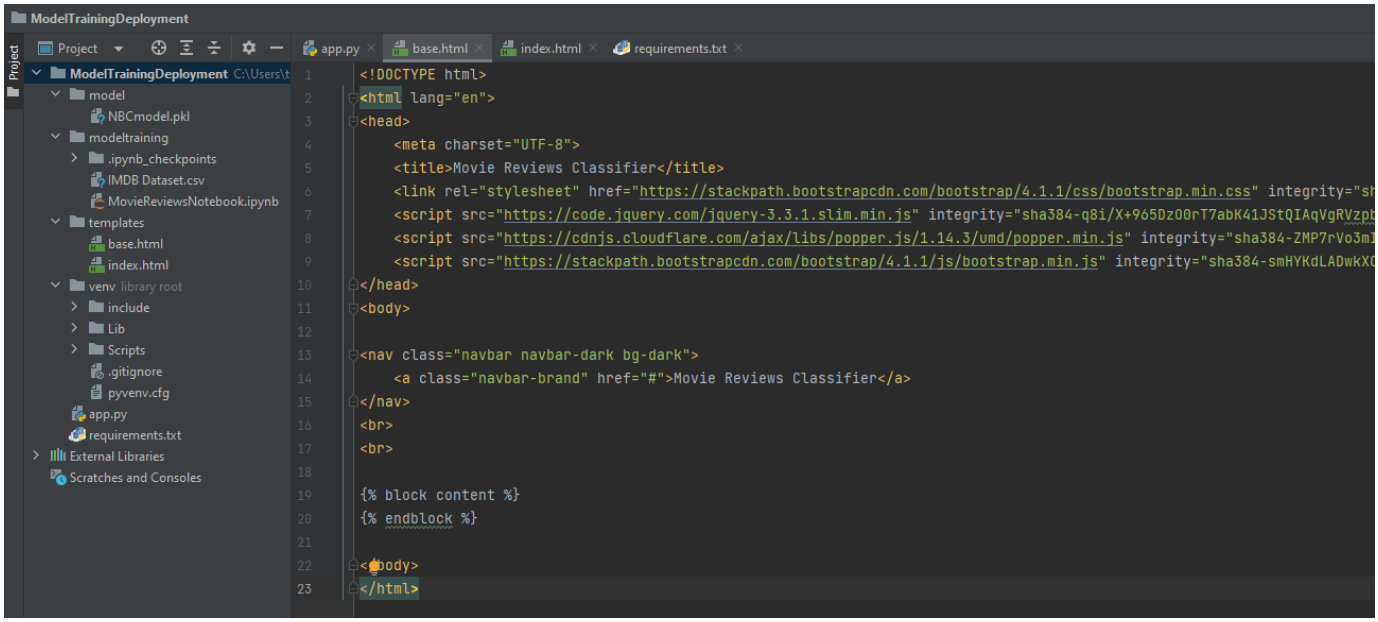
In [20]:
```
#Saving the model with pickle in a .pkl file
with open("..\\model\\NBCmodel.pkl","wb") as nbc:
    pickle.dump(pipe,nbc)
```

2 DEPLOYMENT ON HEROKU

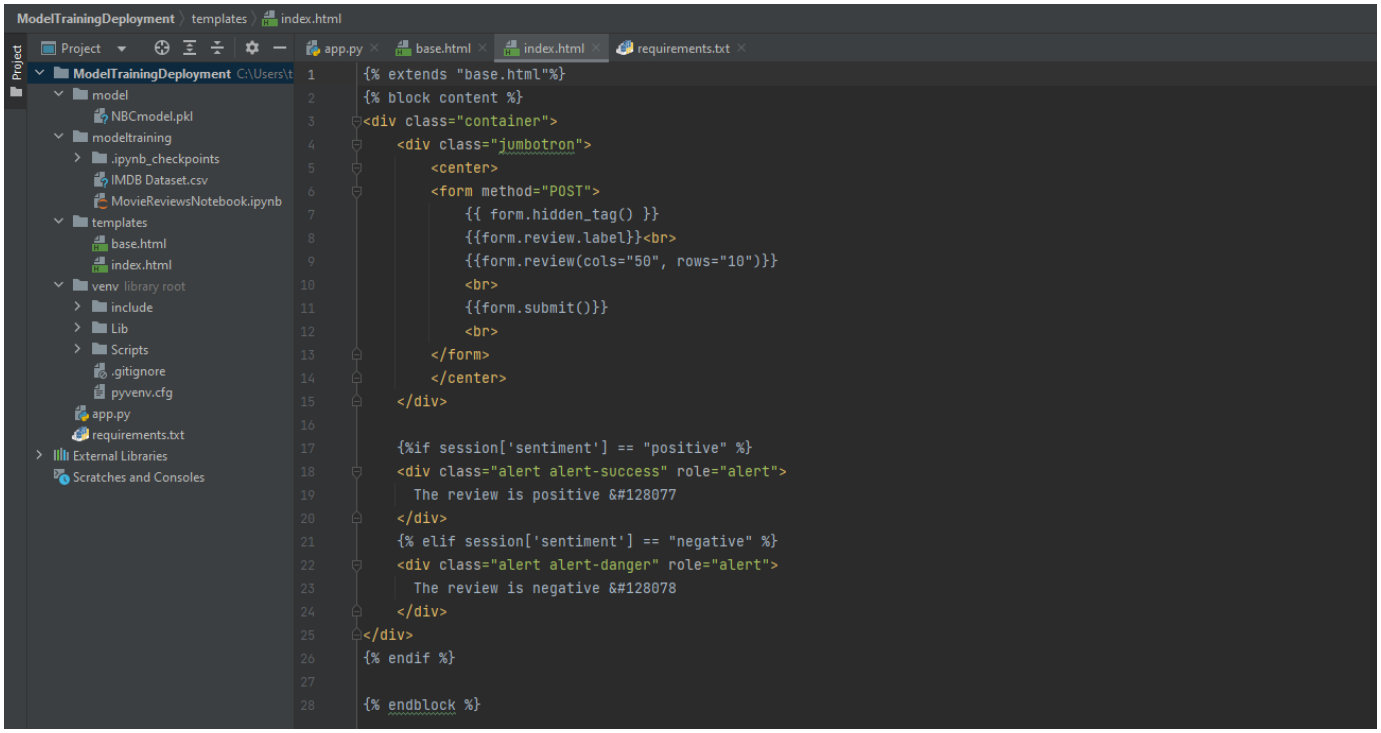
2.1 CREATING THE FLASK APP

To deploy on flask we create a python project in pycharm where easily set the virtual environment from the requirements.txt file. We will use bootstrap 4 to style our pages and also we will interact with jinja templates via Flask forms and sessions. In the following figures we provide the snapshots of the app.py file, our html files and the requirements.txt file. Notice that we load the model that we previously saved with pickle in lines 17 and 18 of app.py and we apply it for predicting the sentiment in line 31. The callable api is created through the function ”return_prediction”in line 37.




ModelTrainingDeployment
Project
app.py
base.html
index.html
requirements.txt
ModelTrainingDeployment C:\Users\t
model
NBCmodel.pkl
modeltraining
.ipynb_checkpoints
IMDB Dataset.csv
MovieReviewsNotebook.ipynb
templates
base.html
index.html
venv library root
include
Lib
Scripts
.gitignore
pyvenv.cfg
app.py
requirements.txt
External Libraries
Scratches and Consoles
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Movie Reviews Classifier</title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" integrity="sh
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpb
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-ZMP7rVo3mI
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js" integrity="sha384-smHYKdLADwkXO
</head>
<body>
<nav class="navbar navbar-dark bg-dark">
Movie Reviews Classifier
</nav>

{% block content %}
{% endblock %}
<body>
</html>


ModelTrainingDeployment templates index.html
Project
app.py
base.html
index.html
requirements.txt
ModelTrainingDeployment C:\Users\t
model
NBCmodel.pkl
modeltraining
.ipynb_checkpoints
IMDB Dataset.csv
MovieReviewsNotebook.ipynb
templates
base.html
index.html
venv library root
include
Lib
Scripts
.gitignore
pyvenv.cfg
app.py
requirements.txt
External Libraries
Scratches and Consoles
{% extends "base.html"%}
{% block content %}

<form method="POST">
{{ form.hidden_tag() }}
{{form.review.label}}
{{form.review(cols="50", rows="10")}}

{{form.submit()}}

</form>

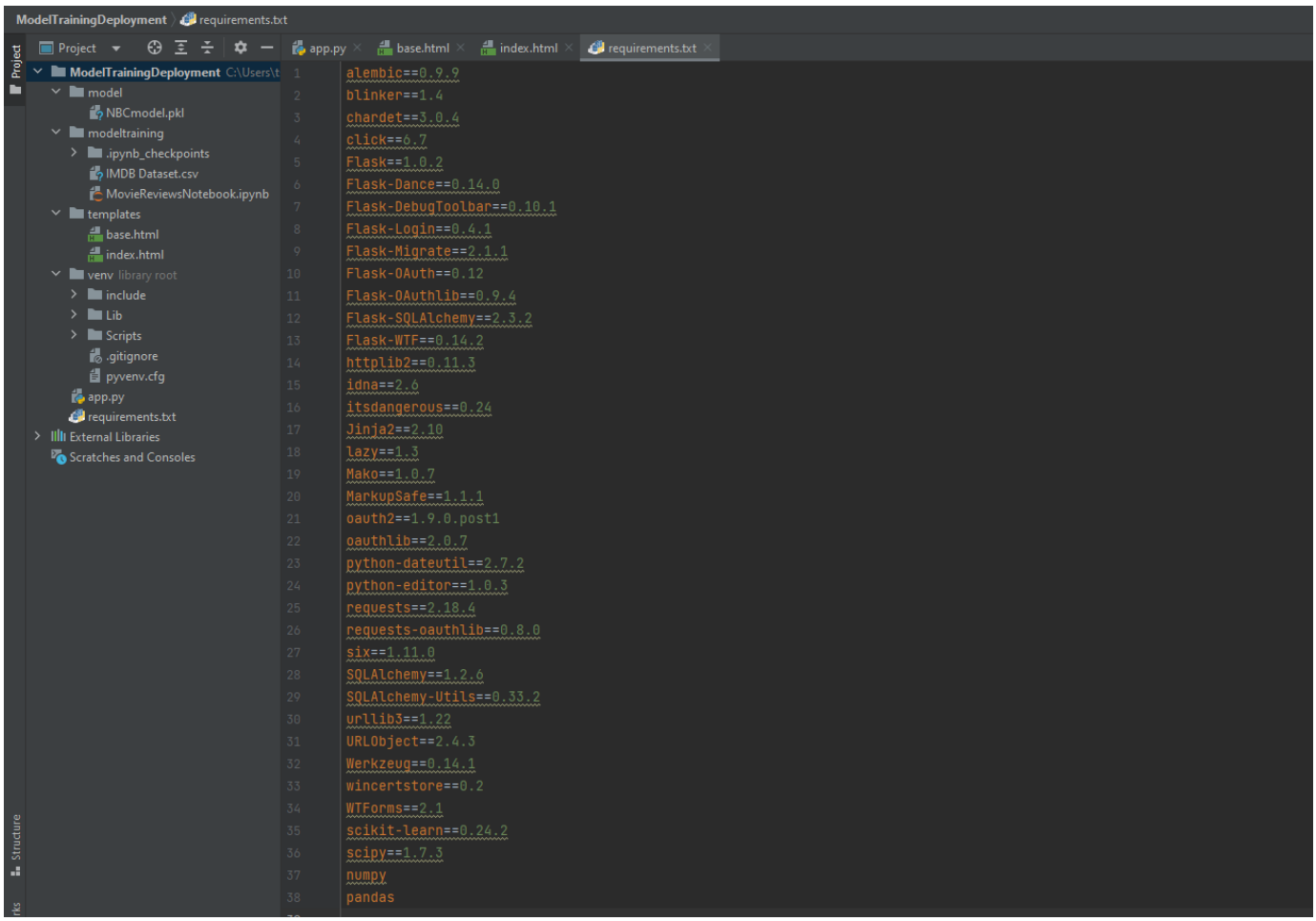
{%if session['sentiment'] == "positive" %}

The review is positive &#128077

{% elif session['sentiment'] == "negative" %}

The review is negative &#128078

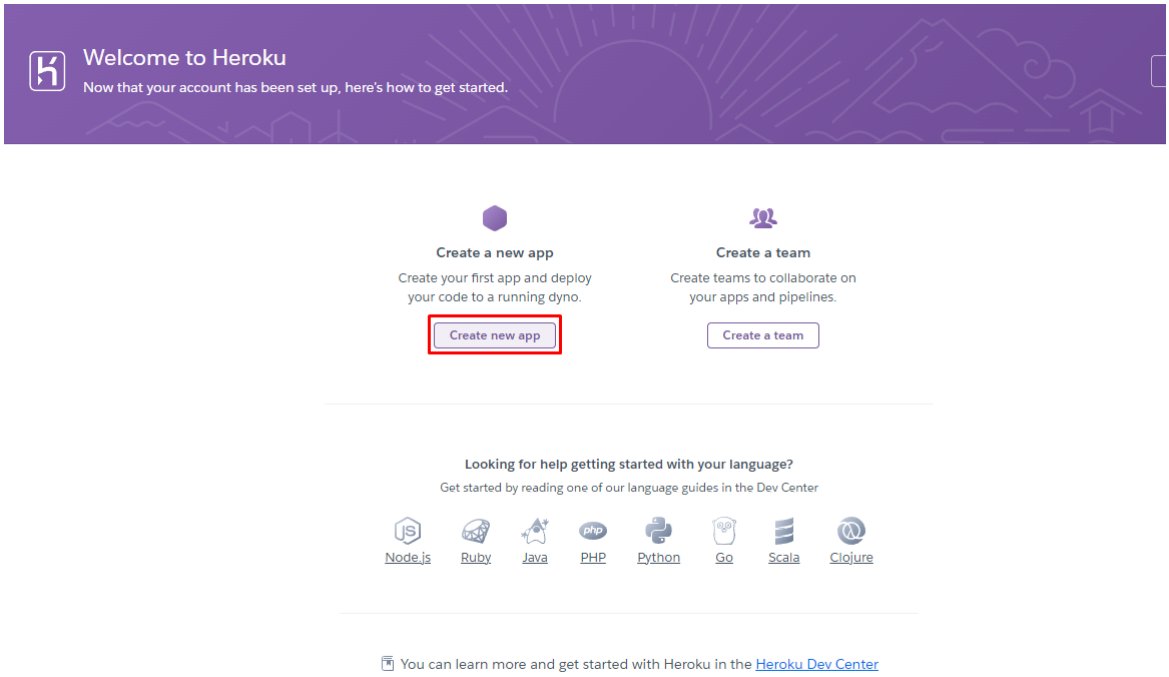
{% endif %}
{% endblock %}

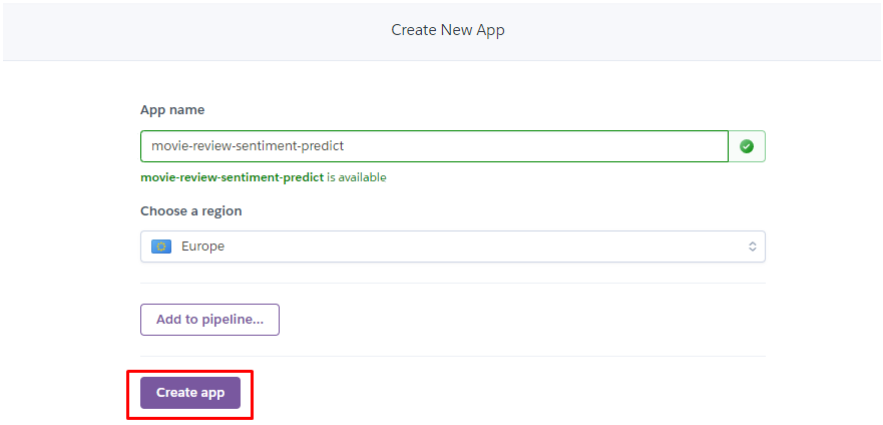


In the figure below we provide the log of running our application.

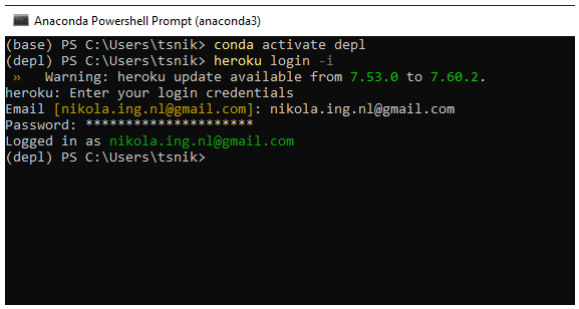
2.2 UPLOADING TO HEROKU

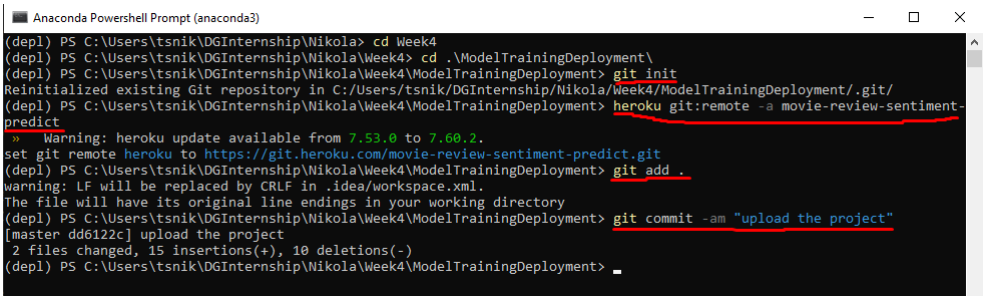
We log in to our Heroku account, create the app and follow the steps for uploading. The snapshots are shown below.

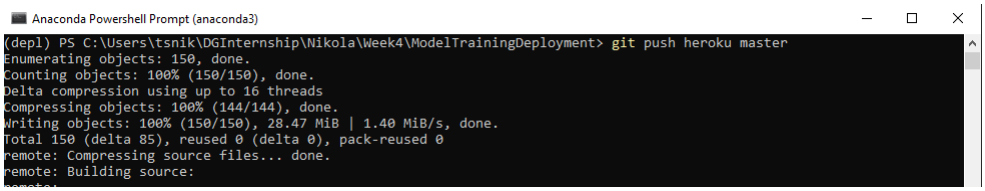


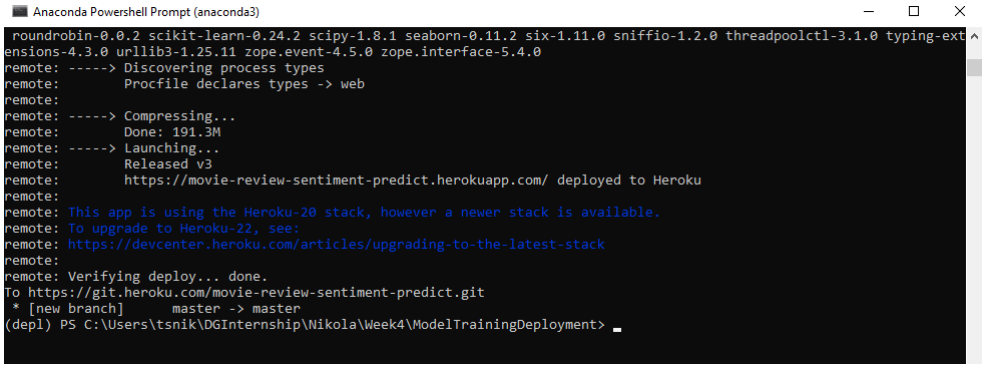


Now we open our anaconda prompt and use git and heroku commands to upload the application. These are shown below.



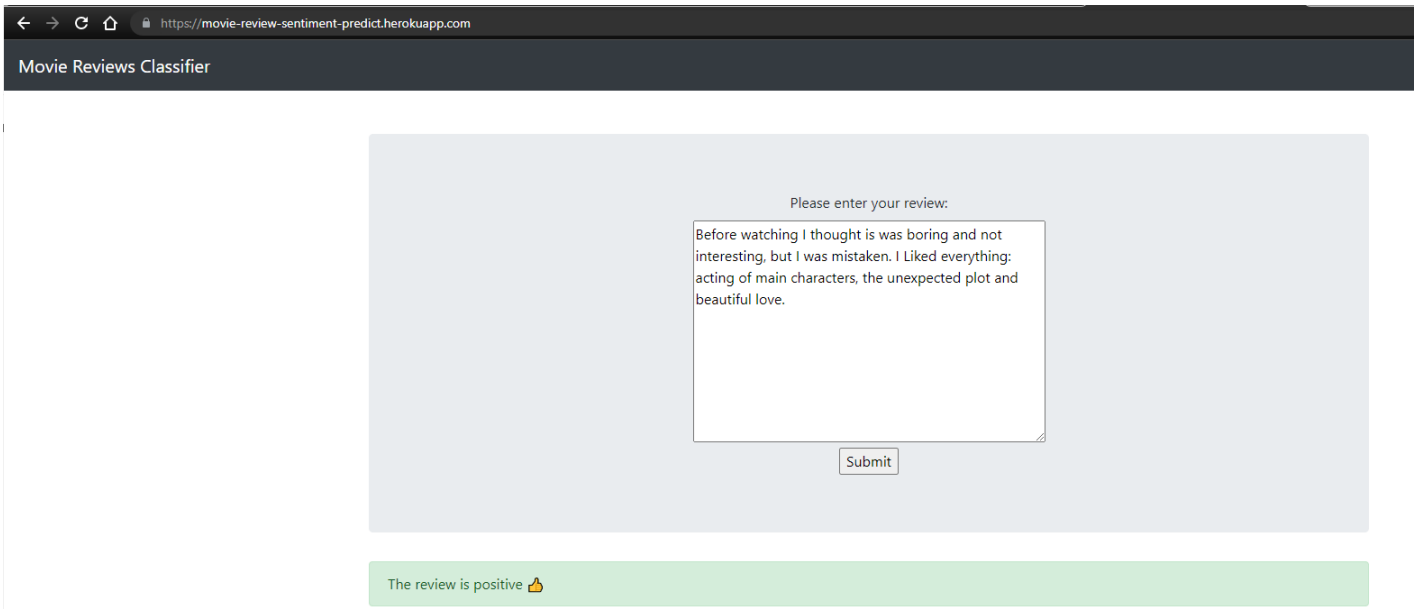


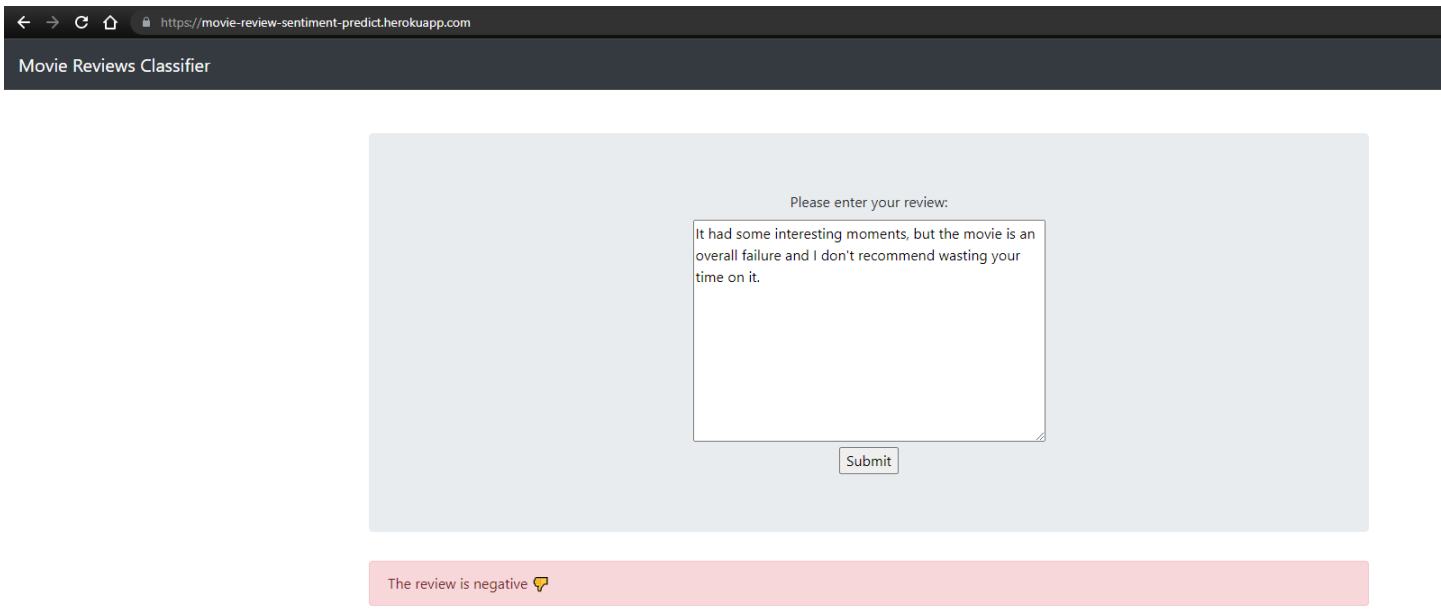




2.3 TESTING THE APPLICATION AND THE API

Let's see how our application looks when opened in browser and let's test it with some sample reviews.





To test the api we will use Postman application to send the request and to check out the response.