Sentiment Analysis for movie reviews

# Submitted by

# Rohit Chawla

## Under Guidance of:

## **Pallavi Peri**

## **Kanimozhi G**

### Report Submitted to

### ***WiseTechSource***



**Abstract**

Sentiment analysis is the analysis of emotions and opinions from any form of text. Sentiment analysis of the data is very useful to express the opinion of the mass or group or any individual. This technique is widely used to find the sentiment of the person with respect to a given source of content. Social media and other platforms contain a huge amount of the data in the form of tweets, blogs, and updates on the status, posts, etc. In purpose of project is to analyse the large text data/reviews of the users who genuinely write the reviews. We can easily read the small texts but it becomes hard to read large paragraphs, Here Sentiment analysis help us to focus on which part to read and gives the result in abstract way. I have classified the Movie reviews using machine learning algorithms like Logistic Regression and Naïve Bayes into Positive and Negative polarity.

**Introduction**

Sentiment analysis refers to use of natural language processing/ text analysis that uses a computational approach to identify opinionated content and categorize it as positive or negative. The unstructured textual data on the any platform often carries expression of opinions of users. Sentiment analysis tries to identify the expressions/opinions and mood of writers. A simple sentiment analysis algorithm attempts to classify a document as ‘positive’ or ‘negative’, based on the opinion expressed in it.

In many cases our decisions are influenced by the opinion of others. Earlier we use to seek out to friend or neighbours for the opinion of movies but that was before internet awareness. Now we browse for the reviews online and we get tons of opinions and now-a-day’s people go beyond and write about their opinions in detail to help others in deciding what was good and what was bad. Now the problem here is with their telling the whole story it’s gets hard to read whole thing. Here Sentiment will help us classifying the overall nature of the opinion as Positive or Negative and the results can be used in research.

Following are the phases of Sentiment Analysis:

Pre-Processing:

* Analysing Target and Features
* The data is cleaned to remove redundancy/noise.
* Transforming data to binary format.

Creating Bag of Words: A model is trained on tokens of words.

Classification: Based on different algorithms the tokens are put under certain category.

**Libraries required**

*import numpy as np*

*import pandas as pd*

*import re,string*

*from nltk.corpus import stopwords*

*from nltk.stem import WordNetLemmatizer*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.model\_selection import train\_test\_split*

*import json, sqlite3,pickle*

*from sklearn.feature\_extraction.text import CountVectorizer, MultinomialNB*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*%matplotlib inline*

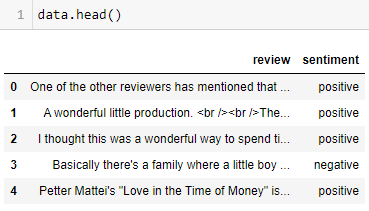
*from flask import Flask, request, render\_template*

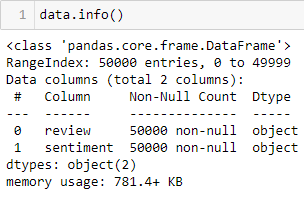
**Data Pre-Processing**

**Data Collection**

The dataset was downloaded from Kaggle <https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews> in the .csv format.

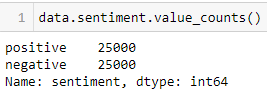
**Performing EDA**

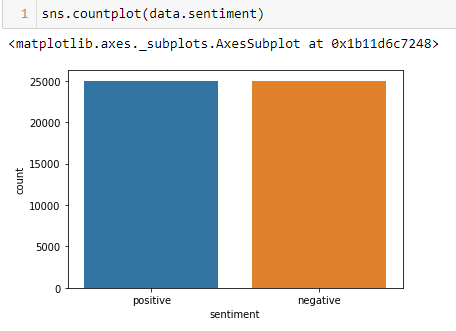




*We have one Feature column “review” and one Target column “sentiment” and no null values!*

Let’s check for the number of the positive and negative values.

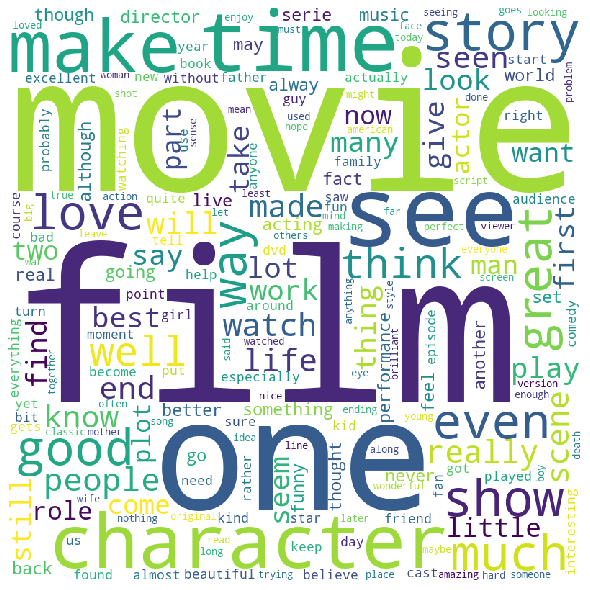




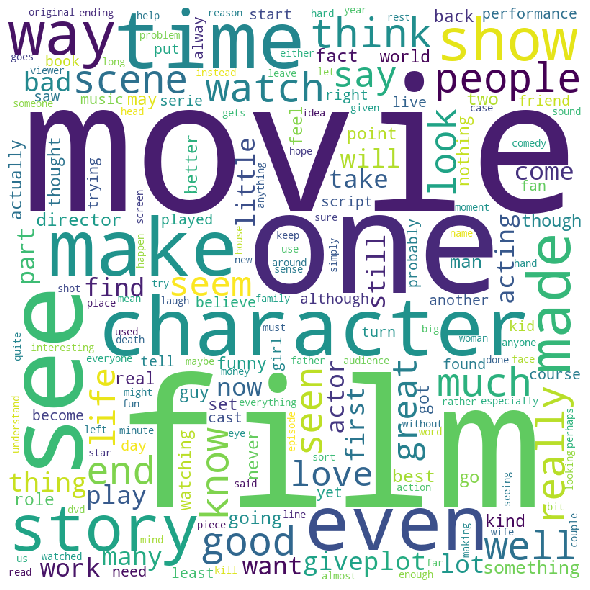
*As We have equal number of positive and negative values our target features will be weighted equally!*

**Word Could of polarities**

Positive Words



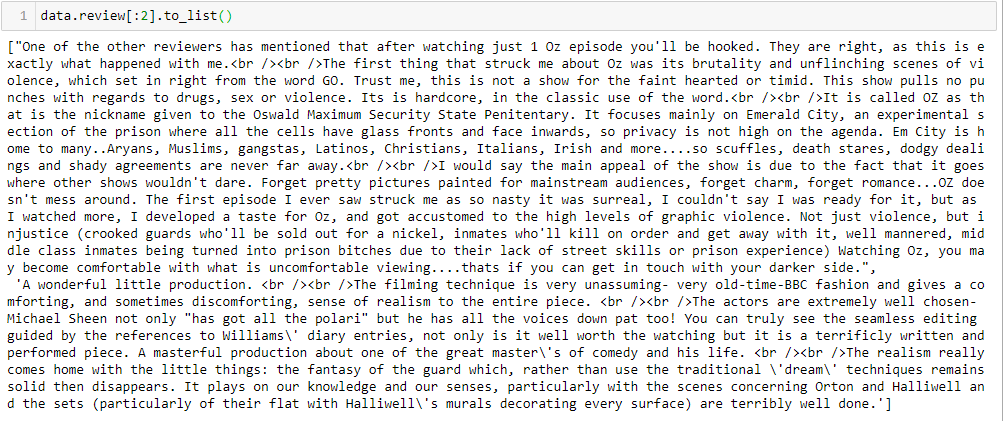
Negative Words



**Data Cleaning**

Since, collected data might be gathered from some website or some other platform we have some junk/ noise (redundancy) in our data.

Here is the example of the raw data collected.



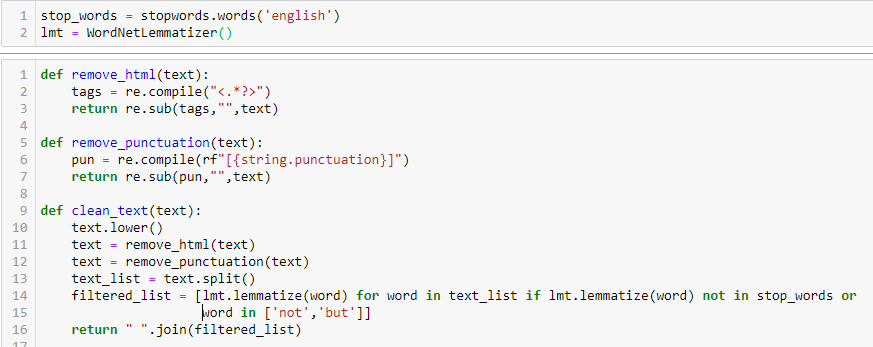
*As we can see, we have some garbage in our data (html tags, punctuations, etc.)*

Following methods are used to clean the data:

**Clean HTML**: HTML tags are nothing but the tags used in the HTML code of the webpage which means that our data was gathered from some webpages.

**Clean Punctuation**: Punctuations are nothing but noise to our data and if not removed will have unnecessary weight to it as no. of punctuations used in a sentence are too high and we want our words to be weighted not punctuations.

**Lemmatization**: Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.



*def remove\_html()*:

Anything between “<>” tags is the html tag. I have used regular expression substitute function to remove the tags.

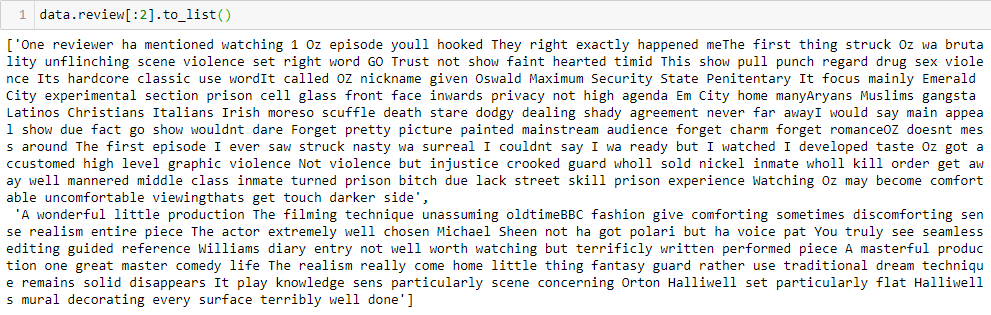
*def remove\_punctuation()*:

Following are the punctuations used in the sentences. And I have used regular expression substitute function.

'!"#$%&\'()\*+,-./:;<=>?@[\\]^\_`{|}~'

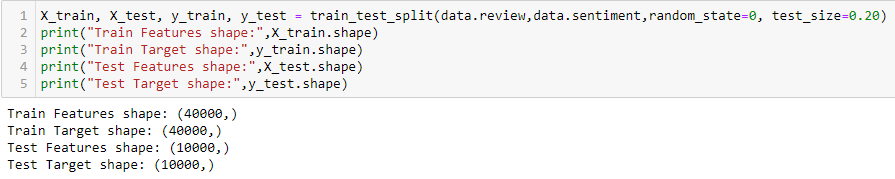
*def clean\_text()*:

It will remove first remove the stop words and then lemmatize the words to its root form. Finally, it will return the combined clean sentence.



**Splitting the data**

Data needs to be split between train and test sample to later check the accuracy of the model.



*data.review*: It is the feature on which our model will be trained.

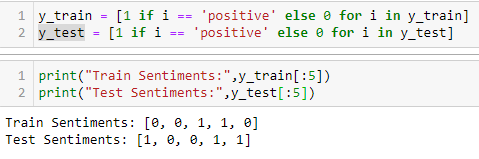
*data.sentiment*: It is the target value our model will predict.

*random\_state*: It is optional but if the state is set it will split the data in same order every time the code ran.

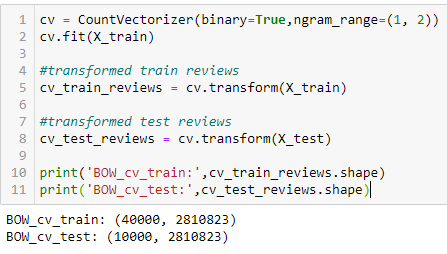
*test\_size*: Percentage of the test data.

**Normalizing the data:** Normalization is needed for the machine to read any data.

Transforming Target values**:**



Transform Feature Values (Bag of Words):



CountVectorizer() is the method to normalize the text and count the frequency of words.

ngram\_range(1, 2) will create the pair of 2 words. i.e. for every word in the sentence it will create 2 pairs with its previous and next word.

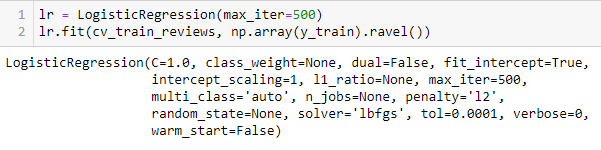
fit() will one column for every word in the data provided.

transform() will convert every review into the SciPy matrix and will fit it into the above created columns.

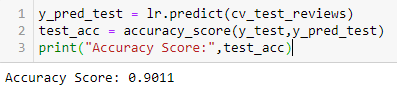
**Training the model**

**Logistic Regression**

**Training:**

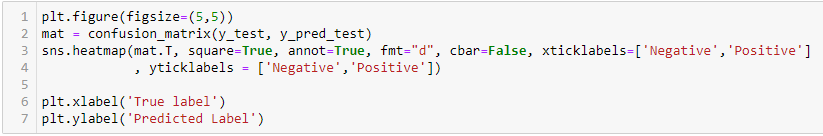


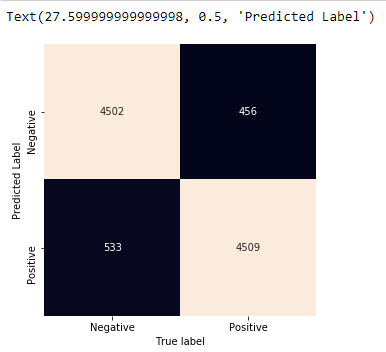
**Checking Accuracy:**



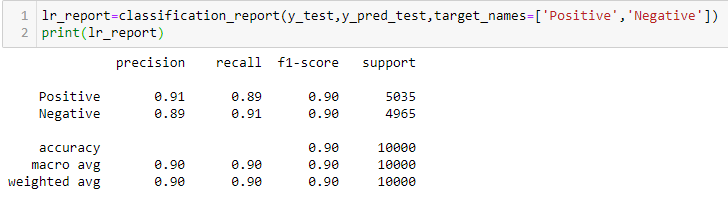
I have got 90% Accuracy, which is really good.

**Confusion Matrix**: Will check for the frequency of True Positive, True Negative, False Positive and False Negative



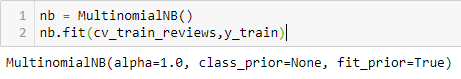


**Classification Report**:

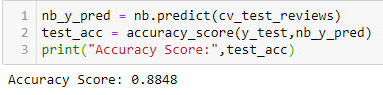


**Naïve Bayes**

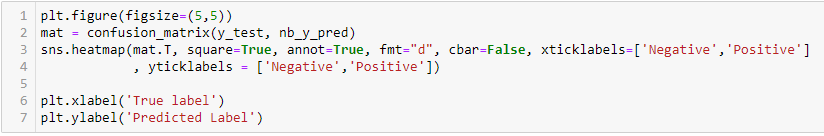
**Training**

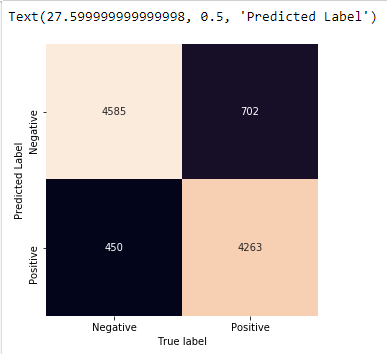


**Checking Accuracy:**

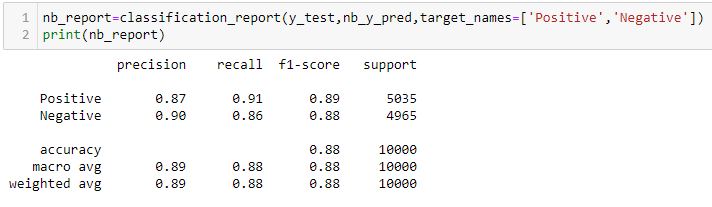


**Confusion Matrix**: Will check for the frequency of True Positive, True Negative, False Positive and False Negative





**Classification Report**:



***Conclusion:*** Comparing both the model overall Logistic Regression shows more accuracy. Hence, for further analysis I have used Logistic regression.

**Front End Using Flask**

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. Hence it provides the most flexibility in terms of customisation and also supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be “micro”, but it’s ready for production use on a variety of needs.

*This app id divided into following parts.*

train.py: Does the cleaning/ pre-processing of the data.

functions.py: Contains the necessary functions require for cleaning.

train\_data.json: Clean data dumped by train.py.

app.py: Load the json data and trains the model. Also handle the API Calls.

HTML/CSS: Static Pages, Forms, etc.

The app has 4 pages:

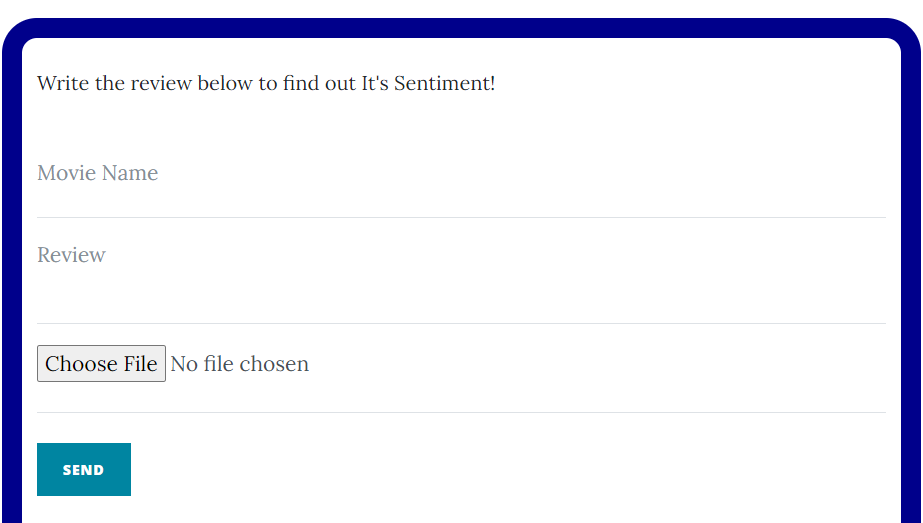
**Home**: Static page shows some review and corresponding sentiment.

**About me**: Tells about the app and the output it provides.

**Post**: Take a text/file as an input, connects with apps and returns their sentiments.

**Get**: Takes the movie name as an input, connects API and if review exists for that movie, it will return the average rating.

**Post page:**



If file is uploaded it will pop download button to download the excel file which has sentiment for every review.

**Get Page:**



**Future Representation**

1. **Technical**
   1. Currently the model approach is to create Bag of Words which returns sparse matrix. Further we can use Word embedding for the same which will create dense matrix instead (i.e. less no. of features)
   2. Word embedding will also increase the semantics between the words.
   3. Hence our model will be much more efficient and accurate.
   4. Two methods to apply word embeddings
      1. Word2Vec
      2. Glove
2. **User Interface**
   1. Currently user can only upload one file excel specifically.
   2. In future more functionalities like upload multiple files, formats can be added.
   3. Also, it will interpret multiple sheets in same workbook i.e. if user provide wanted to analyse reviews for multiple movies provided every sheet in file contains reviews for separate movies.

**Reference**

All the necessary documents including readme file are added to my [git-hub](https://github.com/chawlarohit/sentiment-analysis.git).