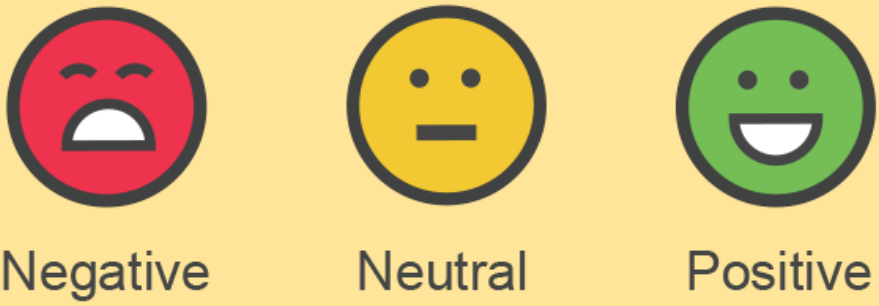
IMDb SENTIMENT ANALYZER



-By Shivam Kumar Giri

OBJECTIVE

To build a custom **sentiment analyzer** that works on movie reviews and effectively categorizes sentiments around them.

ABSTRACT

The large movie view dataset contains a collection of 50,000 reviews from IMDB. The dataset contains an even number of positive and negative reviews. Sentiment analysis (also known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Hence, we are building a custom **sentiment analyzer** that works on movie reviews and effectively categorizes sentiments around them.

DATA

Use a publicly available dataset consisting of 50,000 reviews from IMDB. The dataset has three columns. The first column specifies a sequential number of the record. The second column has a text which is the review and the third column is a class denoting the sentiment of the reviewer. The class has value 1 if the sentiment is positive and 0 if the sentiment is negative.

Our objective is to build a custom sentiment analyzer that can classify sentiment (positive or negative) of reviewers out of their movie reviews. a sentiment analyzer is build based on first 25,000 reviews and validate your classifier using other 25,000 reviews.

METHODOLOGY

Step 1: Import the Libraries

The following libraries are used:

tabulate: to tabularize the final output

pandas: to create dataframe and manipulate data

numpy: for numerical computations

time: for getting the time measurements for training and predictions

re: for Regular Expressions

nltk: for natural language processing

sklearn: features various classification, regression and clustering algorithms

matplotlib: for plotting and visualizing the data

Step 2: Import the DataSet

df=pd.read_csv('./**movie_review_data.csv'**,sep=",", names=['sentiment','reviews'])

create a dataframe from data imported in **movie_review_data.csv.**

Step 3: Basic data exploration

Explore various features of dataset like shape and descriptions:

```
df.shape
```

(50001, 2)

Step 4: Splitting csv into testing and training data

Here Data is splited into set of training data and test data, the training data is trained by decision tree while test data is validated for accuracy.

Here We used 50% of data of 25000 entries for training and rest 50% for testing, which is recommended ratio of splitting.

Step 5: Setup functions for tokenizing and lemmetizing

Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.
Lemmatization reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.

Step 5: Transform to feature vector and convert to sentiments

we can create our model using our training data. In creating the model, we used the TF-IDF as the vectorizer and the Stochastic Gradient Descend algorithm, Ridge Penalty and SVM as the classifier.

Step 6: Setup for dictionary of vocabulary

vectorizer.vocabulary_ is used to setup the dictionary for vocabulary.

Step 7: Setup for test sentiments and vectorize test reviews

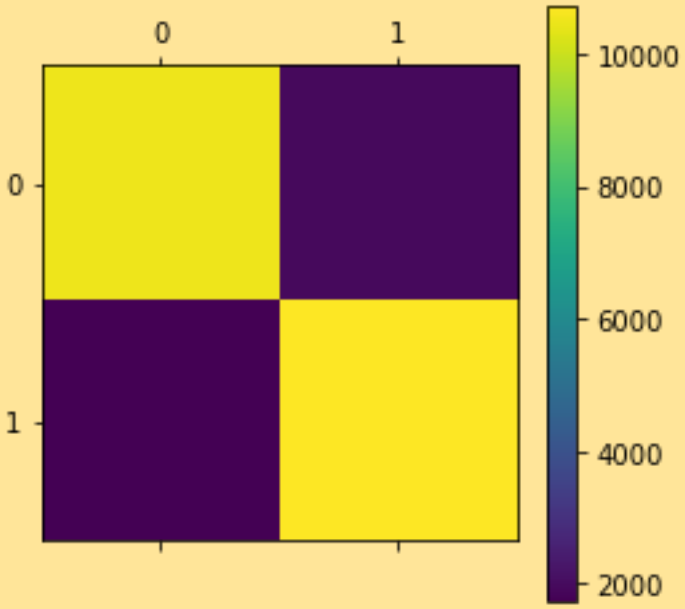
Here, we used the TF-IDF as the vectorizer that is used for vectorize test reviews.

Step 8: Using Stochastic gradient descent Classifier

The class SGDClassifier implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties for classification. Here we are measuring the time in training as well as prediction

Step 9: Measuring the Accuracy, printing Confusion Matrix and report

Confusion matrix:



SGDClassifier Report

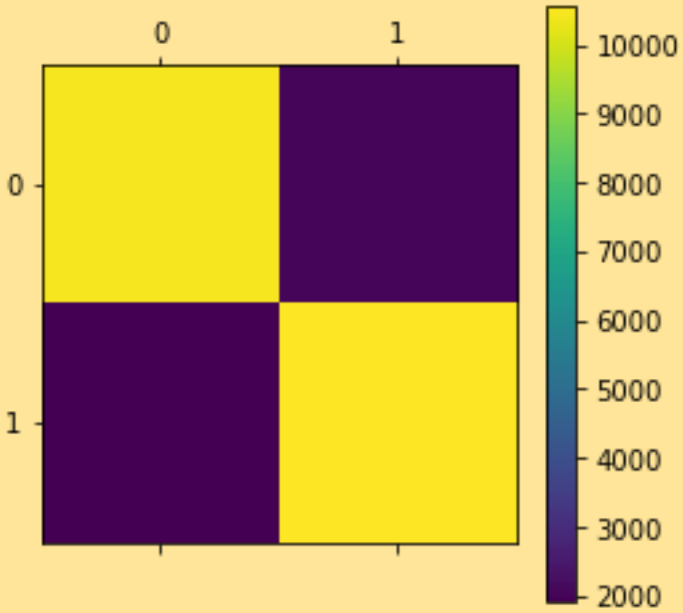
	precision	recall	f1-score	support
Positive	0.86	0.84	0.85	12500
Negative	0.84	0.86	0.85	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

Step 10: Using Ridge Classifier

RidgeClassifier() works differently compared to LogisticRegression() with l2 penalty. The loss function for RidgeClassifier() is not cross entropy. Here we are measuring the time in training as well as prediction

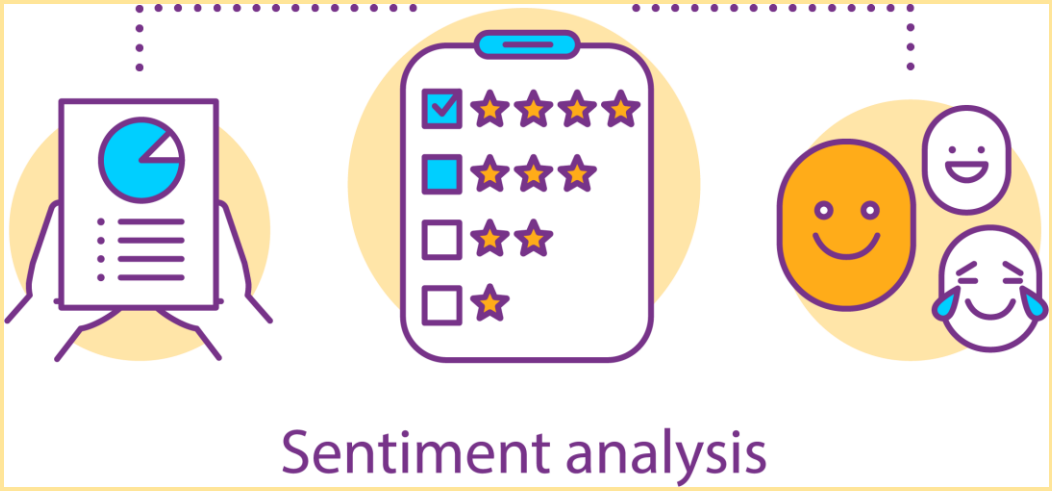
Step 11: Measuring the Accuracy, printing Confusion Matrix and report

Confusion matrix:



RidgeClassifier Report

	precision	recall	f1-score	support
Positive	0.85	0.84	0.84	12500
Negative	0.84	0.85	0.84	12500
accuracy			0.84	25000
macro avg	0.84	0.84	0.84	25000
weighted avg	0.84	0.84	0.84	25000

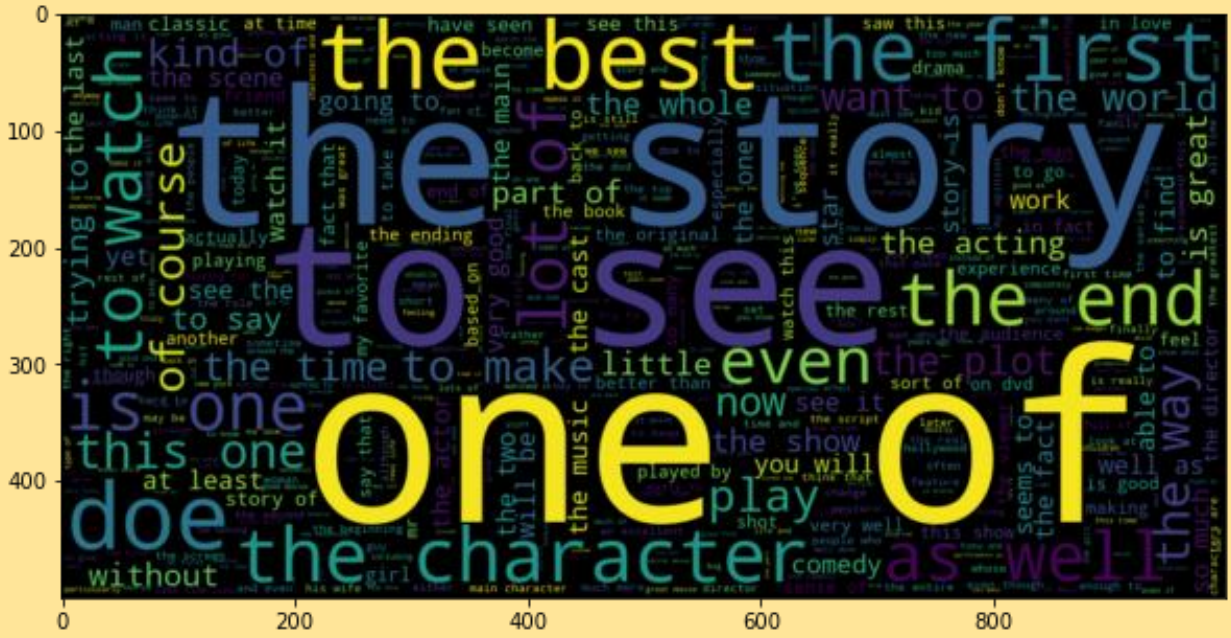


Step 13: Merging the positive and negative reviews

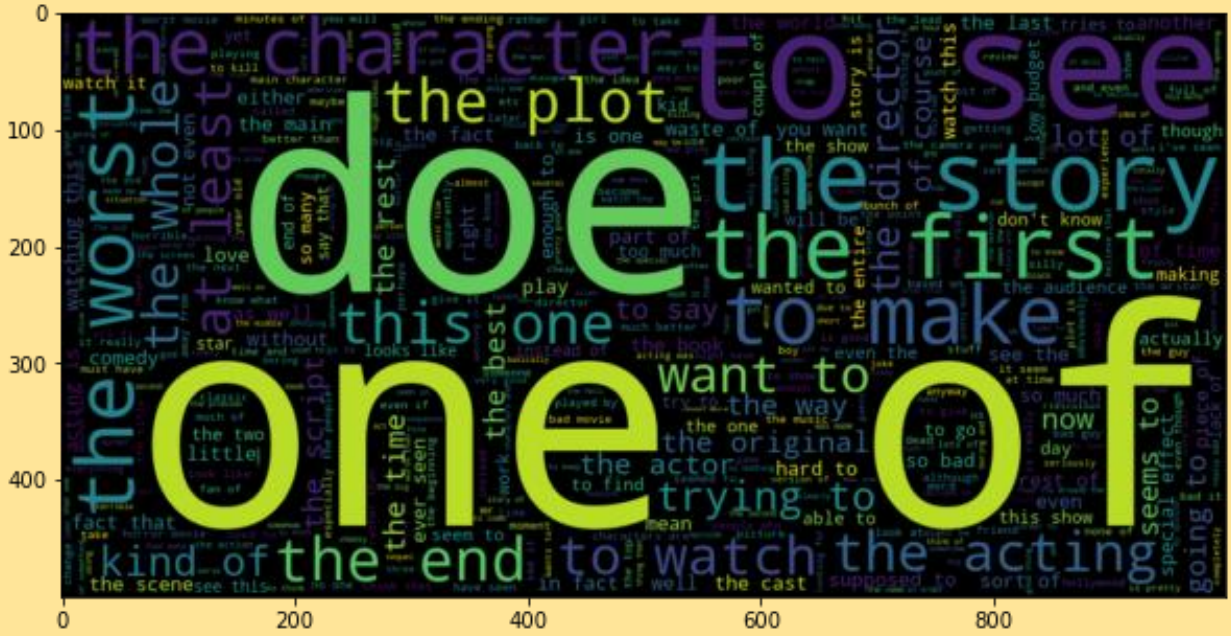
We are merging the positive and negative reviews to create a word cloud of positive and negative reviews.

Step 14: Plotting the positive and negative world cloud

Positive Word cloud



Negative Word Cloud



Step 15: Priniting the final Result

Accuracy by the three Classifier

Classifier	Accuracy	Confusion Matrix	Time for Training(s)	Time for prediction(s)
SGD Classifier :	0.85016	[[10455 2045] [1701 10799]]	3.70609	0.0119658
Ridge Classifier :	0.84204	[[10471 2029] [1920 10580]]	0.812829	0.0119691

CONCLUSION

Hence, we found **SGD classifier as more accurate compare to RidgeClassifier**, but **time for training is more in case of SGD Classifier.** Hence sentiment analyzer with two classifiers is completed with word-cloud.