PROJECT REPORT

TWITTER SENTIMENTAL ANALYSIS

Submitted To:

Dr. Nitin Shelke

Ritik Raina 502004131
Akshat Saklani 502004128

ACKNOWLEDGEMENT

We are deeply thankful to our Advisor, **Dr. Nitin Shekle** for tutoring us throughout the course of Machine Learning & Artificial Intelligence which accomplishment of our final project. Their guidance, support and motivation enabled us in achieving the objectives of the project.

ABSTRACT

This project addresses the problem of sentiment analysis in twitter; that is classifying tweets according to the sentiment expressed in them: positive, negative or neutral. Twitter is an online micro-blogging and social-networking platform which allows users to write short status updates of maximum length 140 characters. It is a rapidly expanding service with over 200 million registered users - out of which 100 million are active users and half of them log on twitter on a daily basis - generating nearly 250 million tweets per day. Due to this large amount of usage we hope to achieve a reflection of public sentiment by analyzing the sentiments expressed in the tweets. Analyzing the public sentiment is important for many applications such as firms trying to find out the response of their products in the market, predicting political elections and predicting socioeconomic phenomena like stock exchange. The aim of this project is to develop is not only to create a functional classifier but also to find out efficient classifier among the three Machine Learning models on the Dataset ***Sentiment140*** which has 1,60,000 tweet records.

TABLE OF CONTENT

INTRODUCTION

We have chosen to work with twitter since we feel it is a better approximation of public sentiment as opposed to conventional internet articles and web blogs. The reason is that the amount of relevant data is much larger for twitter, as compared to traditional blogging sites. Moreover the response on twitter is more prompt and also more general (since the number of users who tweet is substantially more than those who write web blogs on a daily basis). Sentiment analysis of public is highly critical in macro-scale socioeconomic phenomena like predicting the stock market rate of a particular firm. This could be done by analyzing overall public sentiment towards that firm with respect to time and using economics tools for finding the correlation between public sentiment and the firm's stock market value. Firms can also estimate how well their product is responding in the market, which areas of the market is it having a favorable response and in which a negative response (since twitter allows us to download stream of geo-tagged tweets for particular locations. If firms can get this information they can analyze the reasons behind geographically differentiated response, and so they can market their product in a more optimized manner by looking for appropriate solutions like creating suitable market segments. Predicting the results of popular political elections and polls is also an emerging application to sentiment analysis. One such study was conducted by Tumasjanetal. In Germany for predicting the outcome of federal elections in which concluded that twitter is a good reflection of offline sentiment. This project of analyzing sentiments of tweets comes under the domain of "Pattern Classification" and "Data Mining". Both of these terms are very closely related and intertwined, and they can be formally defined as the process of discovering "useful" patterns in large set of data, either automatically (unsupervised) or semi-automatically (supervised). The project would heavily rely on techniques of "Natural Language Processing" in extracting significant

patterns and features from the large data set of tweets and on “Machine Learning” techniques for accurately classifying individual unlabeled data samples (tweets) according to whichever pattern model best describes them.

REVIEW OF LITERATURE

Sentimental analysis of in the domain of micro-blogging is a relatively new research topic so there is still a lot of room for further research in this area. Decent amount of related prior work has been done on sentiment analysis of user reviews, documents, web blogs/articles and general phrase level sentiment analysis. These differ from twitter mainly because of the limit of 140 characters per tweet which forces the user to express opinion compressed in very short text. The best results reached in sentiment classification use supervised learning techniques such as Naive Bayes and Support Vector Machines, but the manual labelling required for the supervised approach is very expensive. Some work has been done on unsupervised and semi-supervised approaches, and there is a lot of room of improvement. Various researchers testing new features and classification techniques often just compare their results to base-line performance. There is a need of proper and formal comparisons between these results arrived through different features and classification techniques in order to select the best features and most efficient classification techniques for particular applications.

The bag-of-words model is one of the most widely used feature model for almost all text classification tasks due to its simplicity coupled with good performance. The model represents the text to be classified as a bag or collection of individual words with no link or dependence of one word with the other, i.e. it completely disregards grammar and order of words within the text. This model is also very popular in sentiment analysis and has been used by various researchers. The simplest way to incorporate this model in our classifier is by using unigrams as features. Generally speaking n-grams is a contiguous sequence of “n” words in our text, which is completely independent of any other words or grams in the text. So unigrams is just a collection of individual words in the text to be classified, and we assume that the probability of occurrence of one word will not be

affected by the presence or absence of any other word in the text. This is a very simplifying assumption but it has been shown to provide rather good performance (for example in [7] and [2]). One simple way to use unigrams as features is to assign them with a certain prior polarity, and take the average of the overall polarity of the text, where the overall polarity of the text could simply be calculated by summing the prior polarities of individual unigrams. Prior polarity of the word would be positive if the word is generally used as an indication of positivity, for example the word "sweet"; while it would be negative if the word is generally associated with negative connotations, for example "evil". There can also be degrees of polarity in the model, which means how much indicative is that word for that particular class. A word like "awesome" would probably have strong subjective polarity along with positivity, while the word "decent" would although have positive prior polarity but probably with weak subjectivity.

There are three ways of using prior polarity of words as features. The simpler un-supervised approach is to use publicly available online lexicons/dictionaries which map a word to its prior polarity. The Multi-Perspective-Question-Answering (MPQA) is an online resource with such a subjectivity lexicon which maps a total of 4,850 words according to whether they are "positive" or "negative" and whether they have "strong" or "weak" subjectivity [25]. The SentiWordNet 3.0 is another such resource which gives probability of each word belonging to positive, negative and neutral classes [15]. The second approach is to construct a custom prior polarity dictionary from our training data according to the occurrence of each word in each particular class. For example if a certain word is occurring more often in the positive labelled phrases in our training dataset (as compared to other classes) then we can calculate the probability of that word belonging to positive class to be higher than the probability of occurring in any other class. This approach has been shown to give better performance, since the prior polarity of words is more suited and fitted to a particular type of text and is not very general like in the former

approach. However, the latter is a supervised approach because the training data has to be labelled in the appropriate classes before it is possible to calculate the relative occurrence of a word in each of the class. Kouloumpis et al. noted a decrease in performance by using the lexicon word features along with custom n-gram word features constructed from the training data, as opposed to when the n-grams were used alone [7].

The third approach is a middle ground between the above two approaches. In this approach we construct our own polarity lexicon but not necessarily from our training data, so we don't need to have labelled training data. One way of doing this as proposed by Turney et al. is to calculate the prior semantic orientation (polarity) of a word or phrase by calculating it's mutual information with the word "excellent" and subtracting the result with the mutual information of that word or phrase with the word "poor" [11]. They used the number of result hit counts from online search engines of a relevant query to compute the mutual information.

The final formula they used is as follows:

$$Polarity(phrase) = log_2 \frac{hits(phrase\ NEAR\ \text{"excellent"}).hits(\text{"poor"})}{hits(phrase\ NEAR\ \text{"poor"}).hits(\text{"excellent"})}$$

Where *hits (phrase NEAR "excellent")* means the number documents returned by the search engine in which the phrase (whose polarity is to be calculated) and word "excellent" are co-occurring. While *hits ("excellent")* means the number of documents retuned which contain the word "excellent". Prabowo et al. have gone ahead with this idea and used a seed of 120 positive words and 120 negative to perform the internet searches. So the overall semantic orientation of the word under consideration can be found by calculating the closeness of that word with each one of the seed words and taking and average of it. Another graphical way of calculating polarity of adjectives has been discussed by Hatzivassiloglou et al. The process involves first identifying all conjunctions of adjectives

from the corpus and using a supervised algorithm to mark every pair of adjectives as belonging to the same semantic orientation or different. A graph is constructed in which the nodes are the adjectives and links indicate same or different semantic orientation. Finally a clustering algorithm is applied which divides the graph into two subsets such that nodes within a subset mainly contain links of same orientation and links between the two subsets mainly contain links of different orientation. One of the subsets would contain positive adjectives and the other would contain negative.

Many of the researchers in this field have used already constructed publicly available lexicons of sentiment bearing words while many others have also explored building their own prior polarity lexicons.

The basic problem with the approach of prior polarity approach has been identified by Wilson et al. who distinguish between prior polarity and contextual polarity. They say that the prior polarity of a word may in fact be different from the way the word has been used in the particular context. The paper presented the following phrase as an example:

Philip Clapp, president of the National Environment Trust, sums up well the general thrust of the reaction of environmental movements: "There is no reason at all to believe that the polluters are suddenly going to become reasonable."

In this example all of the four underlined words "trust", "well", "reason" and "reasonable" have positive polarities when observed without context to the phrase, but here they are not being used to express a positive sentiment. This concludes that even though generally speaking a word like "trust" may be used in positive sentences, but this doesn't rule out the chances of it appearing in non-positive sentences as well.

PROJECT FORMULATION

Problem Objective: In this project, we try to implement a **Twitter sentiment analysis model** that helps to overcome the challenges of identifying the sentiments of the tweets. The necessary details regarding the dataset are:

The dataset provided is the **Sentiment140 Dataset** which consists of **1,600,000 tweets** that have been extracted using the Twitter API. The various columns present in the dataset are:

- **target:** the polarity of the tweet (positive or negative).
- **ids:** Unique id of the tweet.
- **date:** the date of the tweet.
- **flag:** It refers to the query. If no such query exists then it is NO QUERY.
- **user:** It refers to the name of the user that tweeted.
- **text:** It refers to the text of the tweet.

The Machine Learning which will be used in the above mentioned objective are as follows:

- **Bernoulli Naive Bayes**

  Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi|y).

  In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to

estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.) Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

- **Support Vector Machine**

  Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

  The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

  SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

- **Logistic Regression**

  Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Logistic Regression is much similar to the Linear Regression except that how they are used.

METHODOLOGY

The methodology to solve the objective of the problem was divided among various steps. These steps were as follows:

- Import Necessary Dependencies

```
import re
import numpy as np
import pandas as pd
# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt
# nltk
from nltk.stem import WordNetLemmatizer
# sklearn
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
```

- Read and Load the Dataset

```
DATASET_COLUMNS=['target','ids','date','flag','user','text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('Sentiment140 dataset.csv', encoding=DATASET_ENCODING, names=DATASET_COLUMNS)
df.sample(5)
```

	target	ids	date	flag	user	text
790149	0	2325582632	Thu Jun 25 05:42:55 PDT 2009	NO_QUERY	Squemily2113	my keyboard is all sticky from the spilt Morph...
1093638	4	1970082646	Sat May 30 02:22:55 PDT 2009	NO_QUERY	katnorman	@RosiieS8 http://twitpic.com/682sv - IT ANTONY...
1460867	4	2063881736	Sun Jun 07 04:26:33 PDT 2009	NO_QUERY	SteveHealy	@AceMas21 Very good! No nice plans at all - ha...
427076	0	2063665077	Sun Jun 07 03:35:17 PDT 2009	NO_QUERY	BeCcKkkyyY	dying hair tomorrow - dark brown....avec no bl...
948636	4	1823481847	Sat May 16 22:05:21 PDT 2009	NO_QUERY	heidzillaa	watchingg 'waiting' with britt // nice // txt...

❖ Exploratory Data Analysis

```
print('length of data is', len(df))
```

```
length of data is 1600000
```

```
df.info()
```

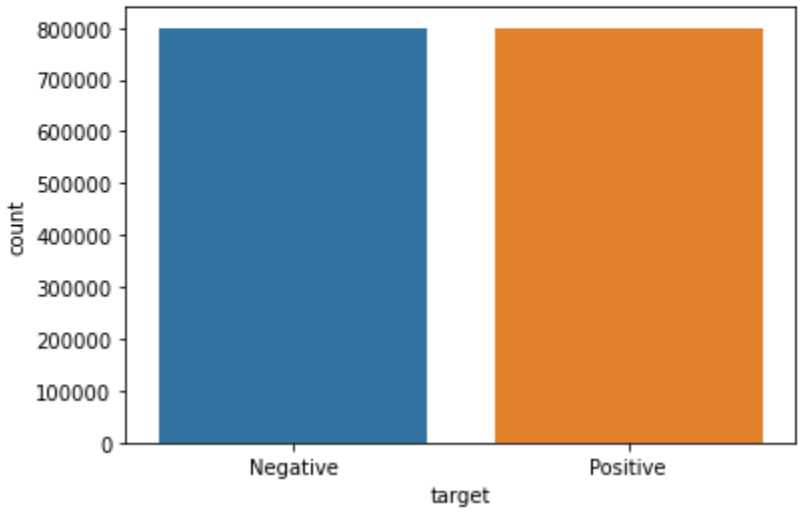
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   target  1600000 non-null  int64
 1   ids     1600000 non-null  int64
 2   date    1600000 non-null  object
 3   flag    1600000 non-null  object
 4   user    1600000 non-null  object
 5   text    1600000 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

```
df.isnull().sum()
```

```
target    0
ids       0
date      0
flag      0
user      0
text      0
dtype: int64
```

❖ Data Visualization of Target Variables

```
fig = sns.countplot(x='target', data=df)
fig.set_xticklabels(['Negative','Positive'], rotation=0)

# Storing data in lists.
text, sentiment = list(df['text']), list(df['target'])
```



❖ Data Preprocessing

It includes various steps to process the data at hand to format it according to the needs of the processing required on it.

- Cleaning and removing the above stop words list from the tweet text.

```
STOPWORDS = set(stopwordlist)
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords(text))
dataset['text'].head()
```

- Cleaning and removing punctuations.

```
import string
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
dataset['text']= dataset['text'].apply(lambda x: cleaning_punctuations(x))
dataset['text'].tail()
```

- Cleaning and removing repeating characters.

```
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
dataset['text'].tail()
```

- Cleaning and removing URL's.

```
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'].tail()
```

- Cleaning and removing Numeric numbers.

```
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
dataset['text'].tail()
```

- Stemming & Lemmatizing

```
#Stemming
import nltk
st = nltk.PorterStemmer()
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data
dataset['text']= dataset['text'].apply(lambda x: stemming_on_text(x))
dataset['text'].head()
```

```
#Lemmatizing
lm = nltk.WordNetLemmatizer()
def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data
dataset['text'] = dataset['text'].apply(lambda x: lemmatizer_on_text(x))
dataset['text'].head()
```

❖ Splitting our data into Train and Test Subset

```
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size = 0.05,
                                                    random_state =26105111)
```

❖ Transforming Dataset using TF-IDF Vectorizer.

```
#Fit the TF-IDF Vectorizer

vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

```
No. of feature_words:  500000
```

❖ Function for Model Evaluation

```
def model_Evaluate(model):
# Predict values for Test dataset
    y_pred = model.predict(X_test)
# Print the evaluation metrics for the dataset.
    print(classification_report(y_test, y_pred))
# Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)
    categories = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten() / np.sum(cf_matrix)]
    labels = [f'{v1}n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',
    xticklabels = categories, yticklabels = categories)
    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

❖ Model Building

```
#Model:3

LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
LRmodel.fit(X_train, y_train)
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test)
```

```
#Model:2

SVCmodel = LinearSVC()
SVCmodel.fit(X_train, y_train)
model_Evaluate(SVCmodel)
y_pred2 = SVCmodel.predict(X_test)
```

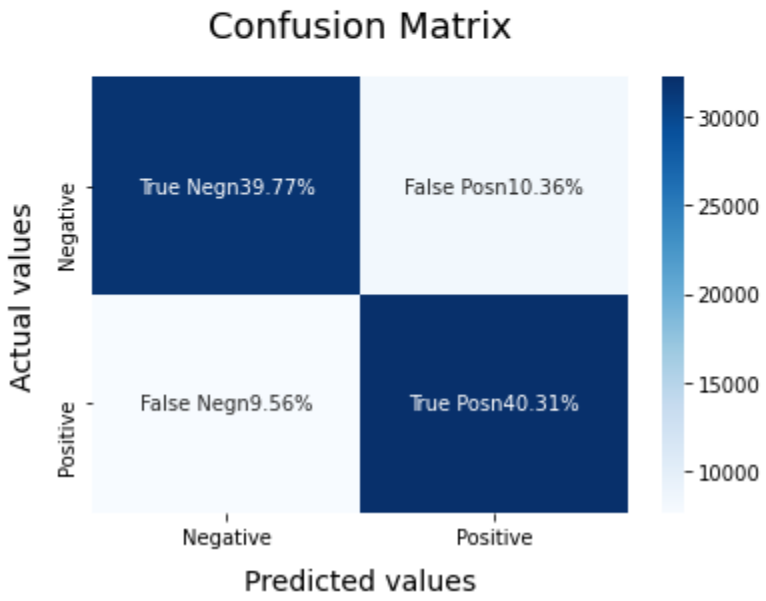
```
#Model:1

BNBmodel = BernoulliNB()
BNBmodel.fit(X_train, y_train)
model_Evaluate(BNBmodel)
y_pred1 = BNBmodel.predict(X_test)
```

PROJECT ANALYSIS

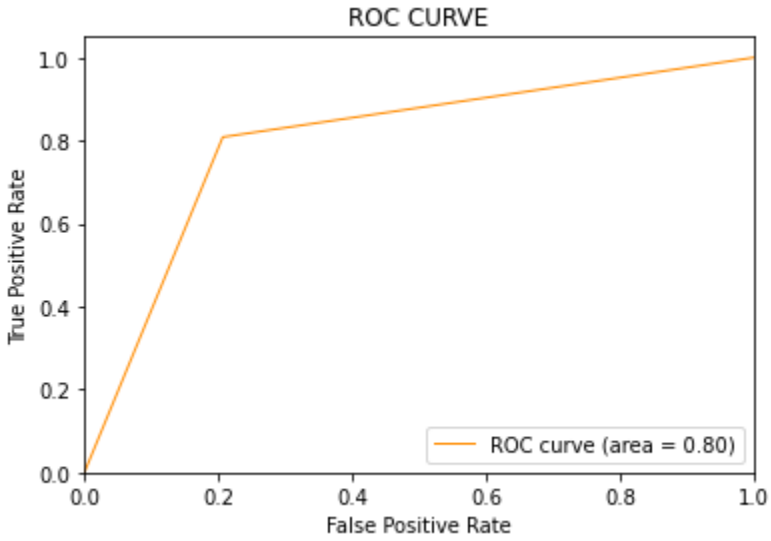
The project analysis can be evaluated by looking at the results generated from the three models which were created based on the historical data of the Dataset: Sentiment140.

- Bernoulli Naves Bayes
  - Classification Report & HeatMap

	precision	recall	f1-score	support
0	0.81	0.79	0.80	40100
1	0.80	0.81	0.80	39900
accuracy			0.80	80000
macro avg	0.80	0.80	0.80	80000
weighted avg	0.80	0.80	0.80	80000



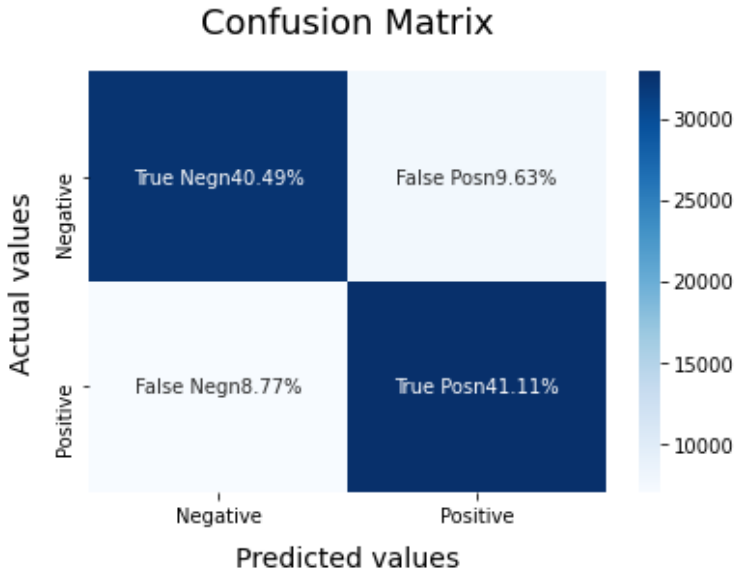
- ROC Curve



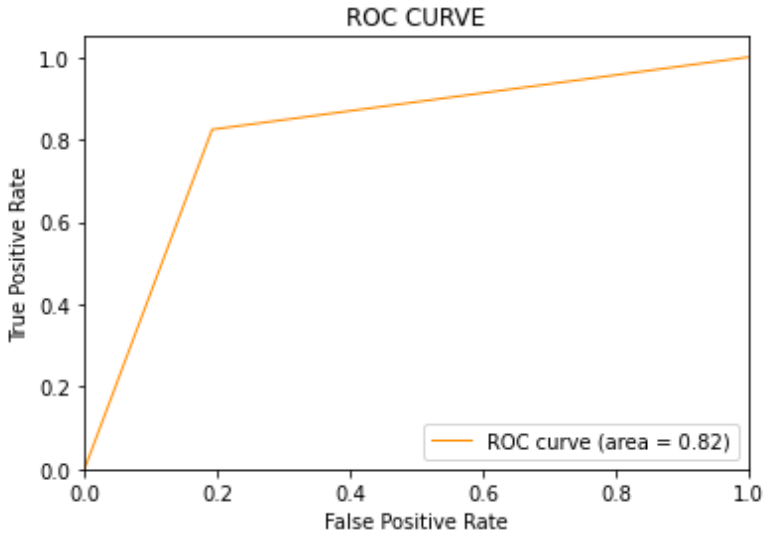
❖ Support Vector Machine

- Classification Report & HeatMap

	precision	recall	f1-score	support
0	0.82	0.81	0.81	40100
1	0.81	0.82	0.82	39900
accuracy			0.82	80000
macro avg	0.82	0.82	0.82	80000
weighted avg	0.82	0.82	0.82	80000



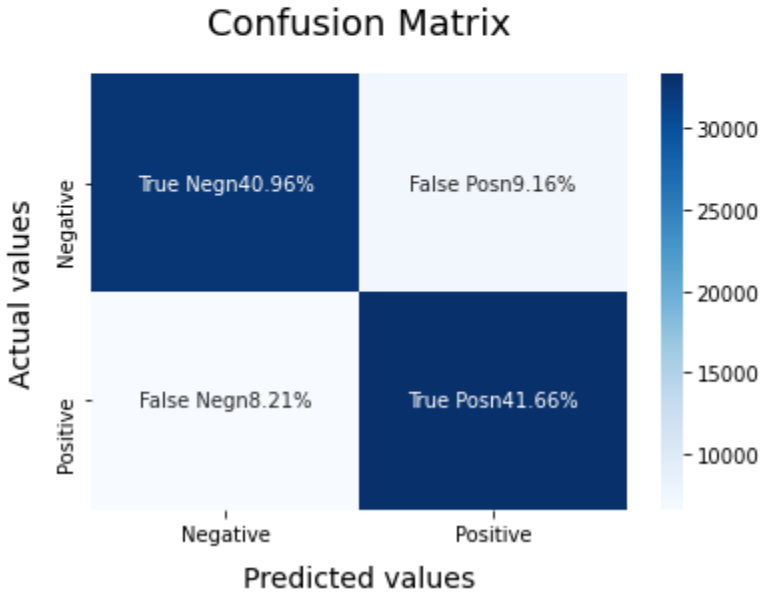
- ROC Curve



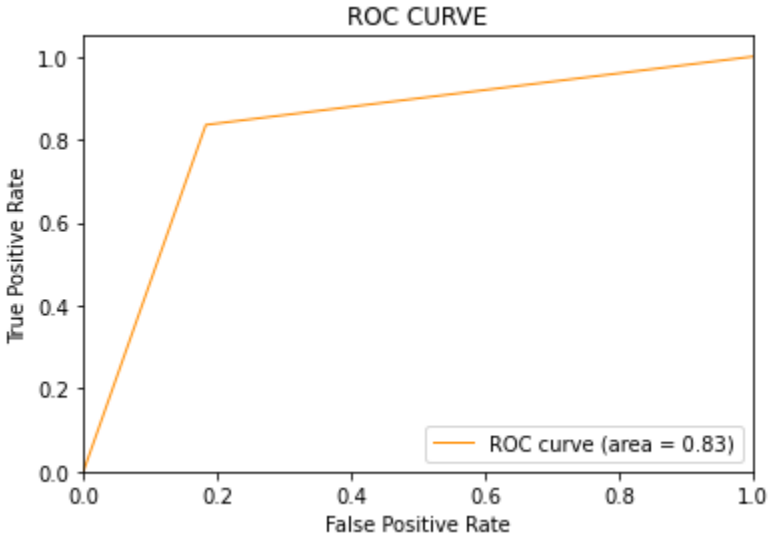
❖ Logistic Regression

- Classification Report & HeatMap

	precision	recall	f1-score	support
0	0.83	0.82	0.83	40100
1	0.82	0.84	0.83	39900
accuracy			0.83	80000
macro avg	0.83	0.83	0.83	80000
weighted avg	0.83	0.83	0.83	80000



- ROC Curve


ROC CURVE
True Positive Rate
False Positive Rate
ROC curve (area = 0.83)
0.0
0.2
0.4
0.6
0.8
1.0

CONCLUSION

Upon evaluating all the models we can conclude the following details i.e.

Accuracy: As far as the accuracy of the model is concerned Logistic Regression performs better than SVM which in turn performs better than Bernoulli Naive Bayes.

F1-score: The F1 Scores for class 0 and class 1 are :

(a) For class 0: Bernoulli Naive Bayes(accuracy = 0.90) < SVM (accuracy =0.91) < Logistic Regression (accuracy = 0.92)

(b) For class 1: Bernoulli Naive Bayes (accuracy = 0.66) < SVM (accuracy = 0.68) < Logistic Regression (accuracy = 0.69)

AUC Score: All three models have the same ROC-AUC score.

We, therefore, conclude that the **Logistic Regression** is the best model for the above-given dataset.

REFERENCES

- https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-use-case-for-beginners/#h2_3