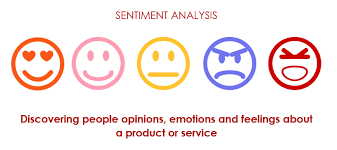
**SENTIMENT ANALYSIS IN MARKETING**

**PHASE V**

**Project: Sentiment Analysis In Marketing With Twitter Responses**

Sentiment analysis is a marketing tool that helps you examine the way people interact with a brand online. This method is more comprehensive than traditional online marketing tracking, which measures the number of online interactions that customers have with a brand, like comments and shares. Using sentiment analysis, you can label individual interactions as positive, negative or neutral. Once you've figured out how to determine and track these labels, you can use this new data set for a variety of marketing purposes, including your online strategy.

Sentimental analysis is an extremely useful tool to have since higher numbers of interactions don't always equate to better results. For example, if you were to receive 10 replies on a social post and all of them were positive, your post likely had a more compelling effect on your audience than if you receive 100 replies with only 10 of them being positive. The primary purpose of sentiment analysis is to respond to commentary more constructively.

****

**Perks of SENTIMENT ANALYSIS:**

1. **Understanding Customer Feedback:** By analyzing the sentiment of customer feedback, companies can identify areas where they need to improve their products or services.
2. **Reputation Management**: Sentiment analysis can help companies monitor their brand reputation online and quickly respond to negative comments or reviews.
3. **Political Analysis**: Sentiment analysis can help political campaigns understand public opinion and tailor their messaging accordingly.
4. **Crisis Management:**In the event of a crisis, sentiment analysis can help organizations monitor social media and news outlets for negative sentiment and respond appropriately.
5. **Marketing Research:** Sentiment analysis can help marketers understand consumer behavior and preferences, and develop targeted advertising campaigns

Here we aim to analyze Twitter sentiment analysis using machine learning algorithms, the sentiment of tweets provided from the **Sentiment140 dataset**by developing a machine learning pipeline involving the use of three classifiers (**Logistic Regression, Bernoulli Naive Bayes, and SVM**)along with using **Term Frequency- Inverse Document Frequency**(**TF-IDF)**. The performance of these classifiers is then evaluated using **accuracy** and **F1 Scores**.

For data preprocessing, we will be using Natural Language Processing’s (NLP) NLTK library.

**Twitter Sentiment Analysis: Problem Statement**

In this project, we try to implement an NLP **Twitter sentiment analysis model** that helps to overcome the challenges of sentiment classification of tweets. We will be classifying the tweets into positive or negative sentiments. The necessary details regarding the dataset involving the Twitter sentiment analysis project are:

RSVP!

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 this Twitter data 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

**Twitter Sentiment Analysis: Project Pipeline**

The various steps involved in the **Machine Learning Pipeline** are:

* Import Necessary Dependencies
* Read and Load the Dataset
* Exploratory Data Analysis
* Data Visualization of Target Variables
* Data Preprocessing
* Splitting our data into Train and Test sets.
* Transforming Dataset using TF-IDF Vectorizer
* Function for Model Evaluation
* Model Building
* Model Evaluation

Let’s get started,

**Step-1: Import the Necessary Dependencies**

# utilities

**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

**Step-2: Read and Load the Dataset**

# Importing the dataset

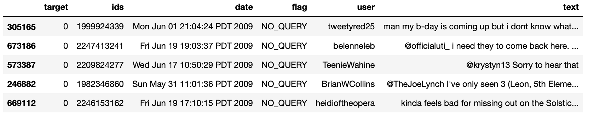
DATASET\_COLUMNS=['target','ids','date','flag','user','text']

DATASET\_ENCODING = "ISO-8859-1"

df = pd.read\_csv('Project\_Data.csv', encoding=DATASET\_ENCODING, names=DATASET\_COLUMNS)

df.sample(5)

**Output:**

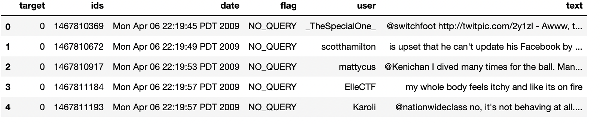


**Step-3: Exploratory Data Analysis**

***3.1: Five top records of data***

df.head()

**Output:**



***3.2: Columns/features in data***

df.columns

**Output:**

Index(['target', 'ids', 'date', 'flag', 'user', 'text'], dtype='object')

***3.3: Length of the dataset***

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

**Output:**

length of **datais**1048576

***3.4: Shape of data***

df. shape

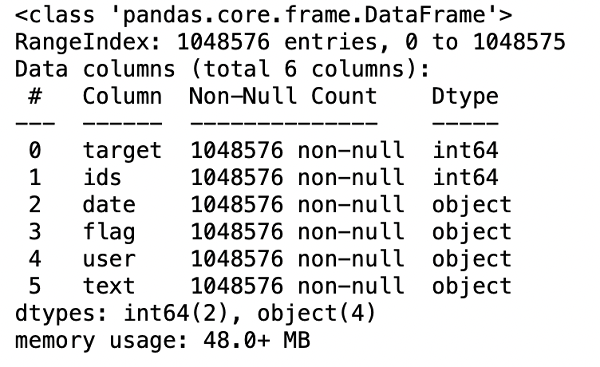
**Output:**

(1048576, 6)

***3.5: Data information***

df.info()

**Output:**



***3.6:*** **Datatypes of all columns**

df.dtypes

**Output:**

target int64

ids int64

dateobject

flag object

user object

**text**object

dtype:object

***3.7: Checking for null values***

**np**.sum(df.isnull().any(axis=1))

**Output:**

0

***3.8: Rows and columns in the dataset***

print('Count of columns in the data is: ', len(df.columns))

print('Count of rows in the data is: ', len(df))

**Output:**

Count of columns **in** the **datais**: 6

Count of rows **in** the **datais**: 1048576

***3.9: Check unique target values***

df['target'].unique()

**Output:**

array([0, 4], dtype=int64)

***3.10: Check the number of target values***

df['target'].nunique()

**Output:**

2

**Step-4: Data Visualization of Target Variables**

# Plotting the distribution for dataset.

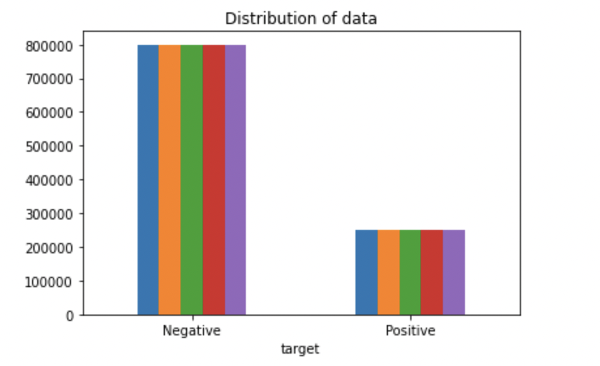
ax = df.groupby('target').count().plot(kind='bar', title='Distribution of data',legend=False)

ax.set\_xticklabels(['Negative','Positive'], rotation=0)

# Storing data in lists.

text, sentiment = list(df['text']), list(df['target'])

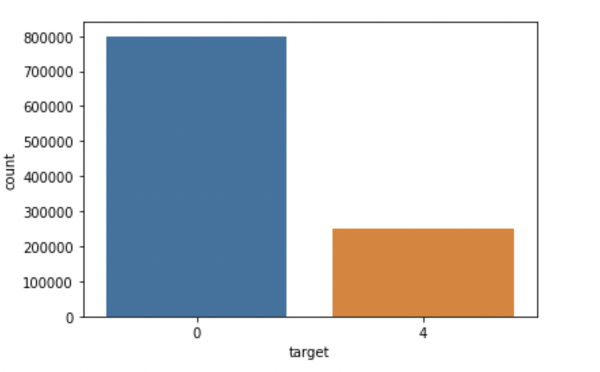
**Output:**



**import** seaborn **as**sns

sns.countplot(x='target', **data**=df)

**Output:**



**Step-5: Data Preprocessing**

In the above-given problem statement, before training the model, we performed various pre-processing steps on the dataset that mainly dealt with removing stopwords, removing special characters like emojis, hashtags, etc. The text document is then converted into lowercase for better generalization.

Subsequently, the punctuations were cleaned and removed, thereby reducing the unnecessary noise from the dataset. After that, we also removed the repeating characters from the words along with removing the URLs as they do not have any significant importance.

At last, we then performed **Stemming(reducing the words to their derived stems)** and **Lemmatization(reducing the derived words to their root form, known as lemma)** for better results.

***5.1: Selecting the text and Target column for our further analysis***

data=df[['text','target']]

***5.2: Replacing the values to ease understanding. (Assigning 1 to Positive sentiment 4)***

**data**['target'] = **data**['target'].replace(4,1)

***5.3: Printing unique values of target variables***

data['target'].unique()

**Output:**

array([0, 1], dtype=int64)

***5.4: Separating positive and negative tweets***

data\_pos = data[data['target'] == 1]

data\_neg = data[data['target'] == 0]

***5.5: Taking one-fourth of the data so we can run it on our machine easily***

data\_pos = data\_pos.iloc[:int(20000)]

data\_neg = data\_neg.iloc[:int(20000)]

***5.6: Combining positive and negative tweets***

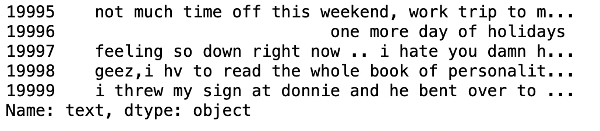
dataset = pd.concat([data\_pos, data\_neg])

***5.7: Making statement text in lowercase***

dataset['text']=dataset['text'].str.lower()

dataset['text'].tail()

**Output:**



***5.8: Defining set containing all stopwords in English.***

stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',

'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',

'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do',

'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',

'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',

'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',

'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',

'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'once',

'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', 're','s', 'same', 'she', "shes", 'should', "shouldve",'so', 'some', 'such',

't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',

'themselves', 'then', 'there', 'these', 'they', 'this', 'those',

'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was',

'we', 'were', 'what', 'when', 'where','which','while', 'who', 'whom',

'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre",

"youve", 'your', 'yours', 'yourself', 'yourselves']

***5.9: Cleaning and removing the above stop words list from the tweet text***

STOPWORDS = set(stopwordlist)

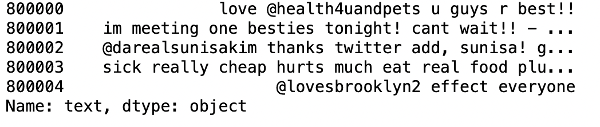
**defcleaning\_stopwords**(text):

**return**" ".join([word **for** word **in**str(text).split() **if** word **notin** STOPWORDS])

dataset['text'] = dataset['text'].apply(**lambda** text: cleaning\_stopwords(text))

dataset['text'].head()

**Output:**



***5.10: Cleaning and removing punctuations***

**import** string

english\_punctuations = string.punctuation

punctuations\_list = english\_punctuations

**defcleaning\_punctuations**(text):

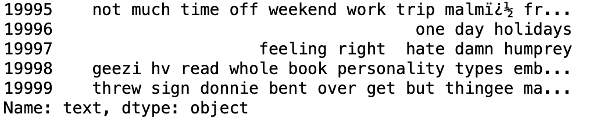
translator = str.maketrans('', '', punctuations\_list)

**return**text.translate(translator)

dataset['text']= dataset['text'].apply(**lambda** x: cleaning\_punctuations(x))

dataset['text'].tail()

**Output:**



***5.11: Cleaning and removing repeating characters***

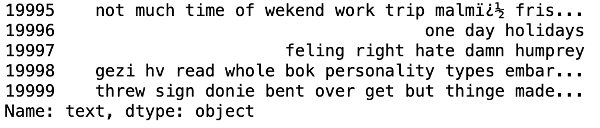
**defcleaning\_repeating\_char**(text):

**return**re.sub(r'(.)1+', r'1', text)

dataset['text'] = dataset['text'].apply(**lambda** x: cleaning\_repeating\_char(x))

dataset['text'].tail()

**Output:**



***5.12: Cleaning and removing URLs***

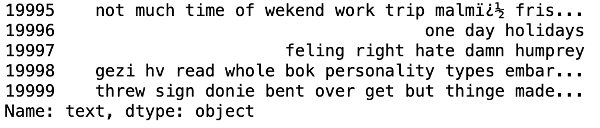
def cleaning\_URLs(data):

return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)

dataset['text'] = dataset['text'].apply(lambda x: cleaning\_URLs(x))

dataset['text'].tail()

**Output:**



***5.13: Cleaning and removing numeric numbers***

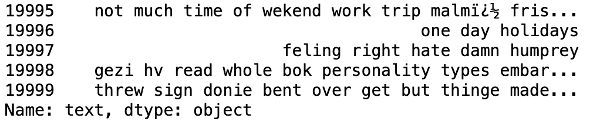
def cleaning\_numbers(data):

return re.sub('[0-9]+', '', data)

dataset['text'] = dataset['text'].apply(lambda x: cleaning\_numbers(x))

dataset['text'].tail()

**Output:**



***5.14: Getting tokenization of tweet text***

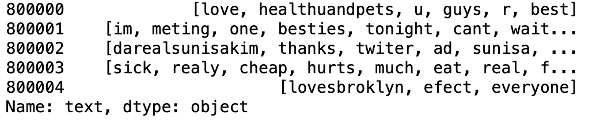
**from**nltk.tokenize**import**RegexpTokenizer

tokenizer = RegexpTokenizer(r'w+')

dataset['text'] = dataset['text'].apply(tokenizer.tokenize)

dataset['text'].head()

**Output:**



***5.15: Applying stemming***

**import**nltk

st = nltk.PorterStemmer()

def stemming\_on\_text(**data**):

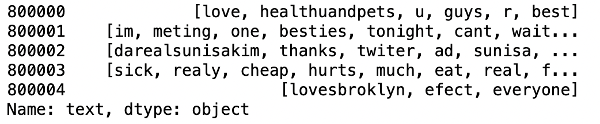
text = [st.stem(word) **for** word **indata**]

**returndata**

dataset['text']= dataset['text'].apply(lambda x: stemming\_on\_text(x))

dataset['text'].head()

**Output:**



***5.16: Applying lemmatizer***

lm = nltk.WordNetLemmatizer()

def lemmatizer\_on\_text(**data**):

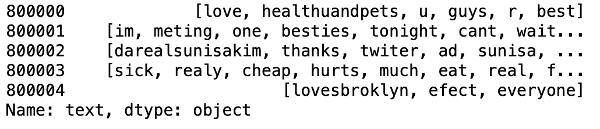
text = [lm.lemmatize(word) **for** word **indata**]

**returndata**

dataset['text'] = dataset['text'].apply(lambda x: lemmatizer\_on\_text(x))

dataset['text'].head()

**Output:**



***5.17: Separating input feature and label***

X=data.text

y=data.target

***5.18: Plot a cloud of words for negative tweets***

data\_neg = data['text'][:800000]

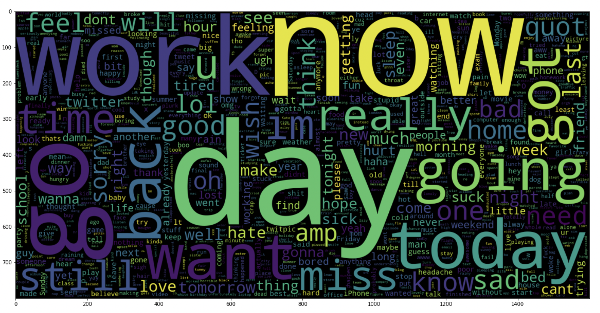
plt.figure(figsize = (20,20))

wc = WordCloud(max\_words = 1000 , width = 1600 , height = 800,

collocations=False).generate(" ".join(data\_neg))

plt.imshow(wc)

**Output:**



***5.19: Plot a cloud of words for positive tweets***

data\_pos = data['text'][800000:]

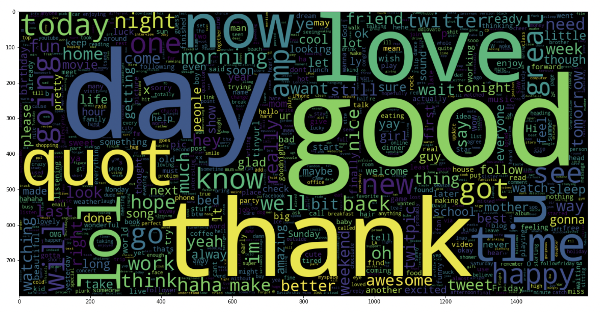
wc = WordCloud(max\_words = 1000 , width = 1600 , height = 800,

collocations=False).generate(" ".join(data\_pos))

plt.figure(figsize = (20,20))

plt.imshow(wc)

**Output:**



**Step-6: Splitting Our Data Into Train and Test Subsets**

# Separating the 95% **datafor** training **data** and 5% **for** testing **data**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.05, random\_state =26105111)

**Step-7: Transforming the Dataset Using TF-IDF Vectorizer**

***7.1: 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()))

**Output:**

No. of feature\_words:500000

***7.2: Transform the data using TF-IDF Vectorizer***

X\_train = vectoriser.transform(X\_train)

X\_test =vectoriser.transform(X\_test)

Step-8: Function for Model Evaluation

After training the model, we then apply the evaluation measures to check how the model is performing. Accordingly, we use the following evaluation parameters to check the performance of the models respectively:

* Accuracy Score
* Confusion Matrix with Plot
* ROC-AUC Curve

**defmodel\_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','TruePos']

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)

**Step-9: Model Building**

In the problem statement, we have used three different models respectively :

* Bernoulli Naive Bayes Classifier
* SVM (Support Vector Machine)
* Logistic Regression

The idea behind choosing these models is that we want to try all the classifiers on the dataset ranging from simple ones to complex models, and then try to find out the one which gives the best performance among them.

***8.1: Model-1***

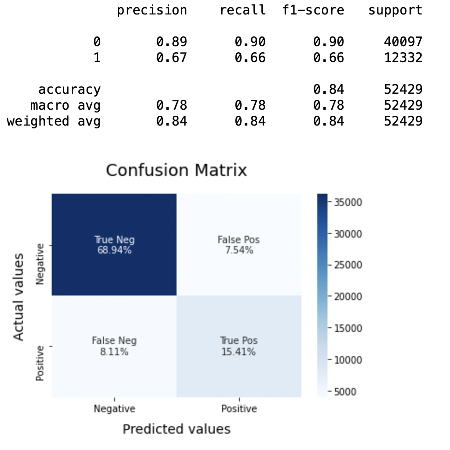
BNBmodel = BernoulliNB()

BNBmodel.fit(X\_train, y\_train)

model\_Evaluate(BNBmodel)

y\_pred1 = BNBmodel.predict(X\_test)

**Output:**



***8.2: Plot the ROC-AUC Curve for model-1***

**from**sklearn.metrics**import**roc\_curve, auc

fpr, tpr, thresholds = **roc\_curve**(y\_test, y\_pred1)

roc\_auc = **auc**(fpr, tpr)

plt.**figure**()

plt.**plot**(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.**xlim**([0.0, 1.0])

plt.**ylim**([0.0, 1.05])

plt.**xlabel**('False Positive Rate')

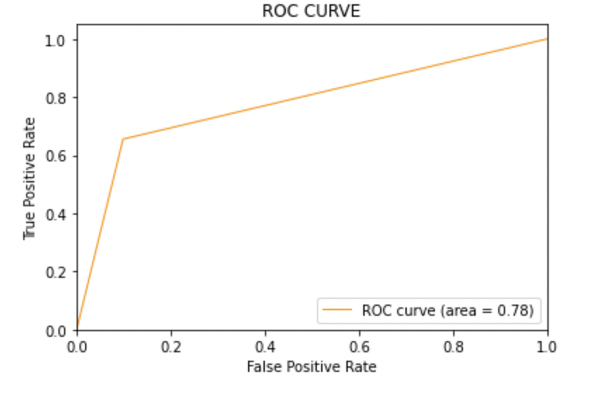
plt.**ylabel**('True Positive Rate')

plt.**title**('ROC CURVE')

plt.**legend**(loc="lower right")

plt.**show**()

**Output:**



***8.3: Model-2:***

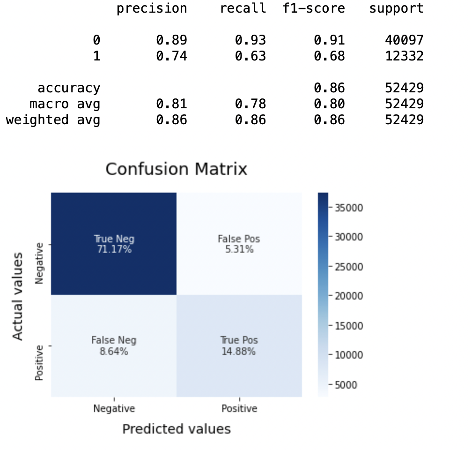
SVCmodel = LinearSVC()

SVCmodel.fit(X\_train, y\_train)

model\_Evaluate(SVCmodel)

y\_pred2 = SVCmodel.predict(X\_test)

**Output:**



***8.4: Plot the ROC-AUC Curve for model-2***

**from**sklearn.metrics**import**roc\_curve, auc

fpr, tpr, thresholds = **roc\_curve**(y\_test, y\_pred2)

roc\_auc = **auc**(fpr, tpr)

plt.**figure**()

plt.**plot**(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.**xlim**([0.0, 1.0])

plt.**ylim**([0.0, 1.05])

plt.**xlabel**('False Positive Rate')

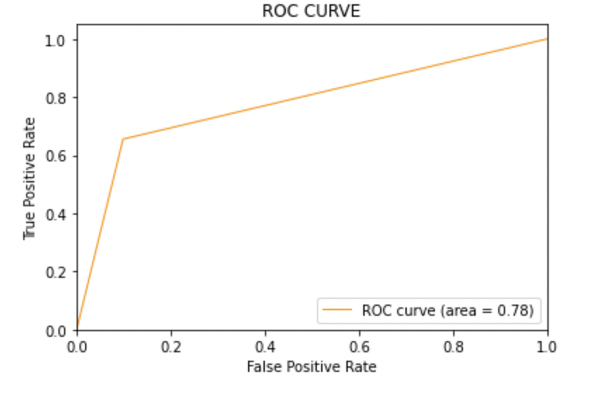
plt.**ylabel**('True Positive Rate')

plt.**title**('ROC CURVE')

plt.**legend**(loc="lower right")

plt.**show**()

**Output:**



***8.5: Model-3***

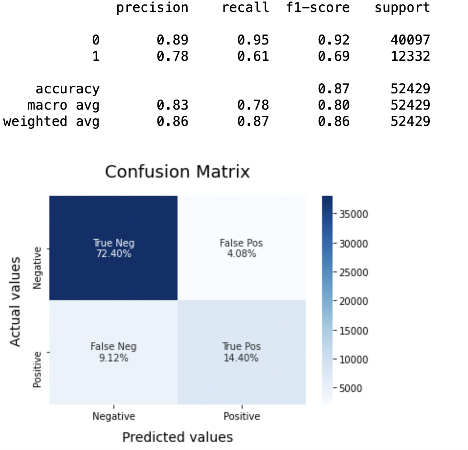
LRmodel = LogisticRegression(C = 2, max\_iter = 1000, n\_jobs=-1)

LRmodel.fit(X\_train, y\_train)

model\_Evaluate(LRmodel)

y\_pred3 = LRmodel.predict(X\_test)

**Output:**



***8.6: Plot the ROC-AUC Curve for model-3***

**from**sklearn.metrics**import**roc\_curve, auc

fpr, tpr, thresholds = **roc\_curve**(y\_test, y\_pred3)

roc\_auc = **auc**(fpr, tpr)

plt.**figure**()

plt.**plot**(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.**xlim**([0.0, 1.0])

plt.**ylim**([0.0, 1.05])

plt.**xlabel**('False Positive Rate')

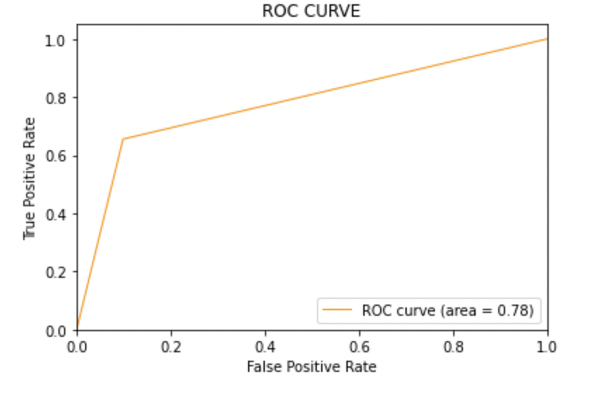
plt.**ylabel**('True Positive Rate')

plt.**title**('ROC CURVE')

plt.**legend**(loc="lower right")

plt.**show**()

**Output:**



**Step-10: Model Evaluation**

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.

In our problem statement, **Logistic Regression** follows the principle of **Occam’s Razor,** which defines that for a particular problem statement, if the data has no assumption, then the simplest model works the best. Since our dataset does not have any assumptions and Logistic Regression is a simple model. Therefore, the concept holds true for the above-mentioned dataset.

**Conclusion**

Twitter Sentimental Analysis helps us preprocess the data (tweets) using different methods and feed it into ML models to give the best accuracy.

**Key Takeaways**

* Twitter Sentimental Analysis is used to identify as well as classify the sentiments that are expressed in the text source.
* Logistic Regression, SVM, and Naive Bayes are some of the ML algorithms that can be used for Twitter Sentimental Analysis.