Twitter Sentiment Analysis

Final Report

Batch details	PGPDSE - May-21-B Online
Team members	1. Raghav Vashisht 2. Pratik Kumar Gupta 3. Karthikeyan A R 4. Kattoju Venkata Manikanta Saiteja 5. James Anto Arnold
Domain of Project	Text Analytics
Proposed project title	Twitter Sentiment Analysis
Group Number	2
Team Leader	James Anto Arnold
Mentor Name	Mr. Animesh tiwari

Date: 27-01-2022

Mr. Animesh Tiwar

Signature of the Mentor

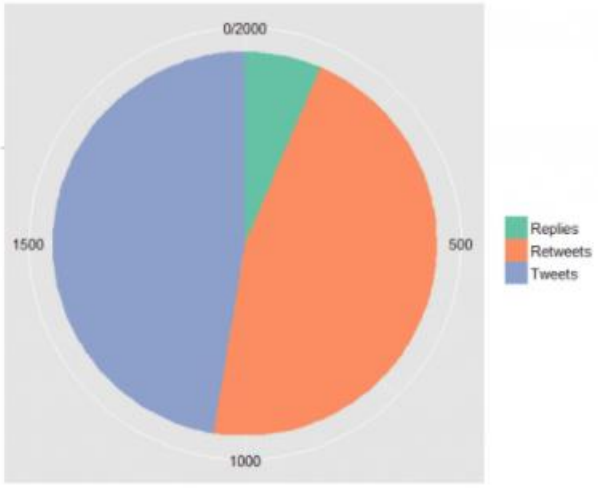
Mr. James Anto Arnold

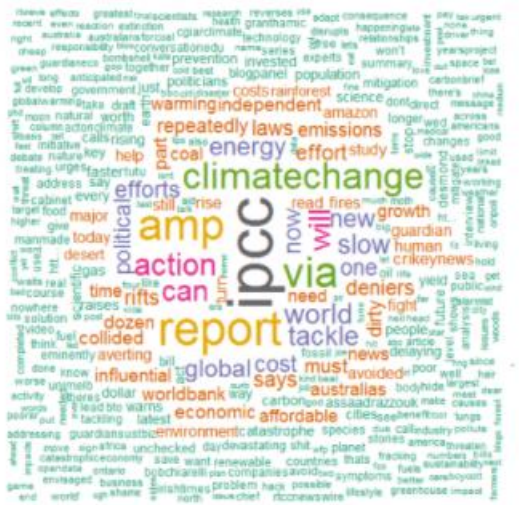
Signature of the Team Leader

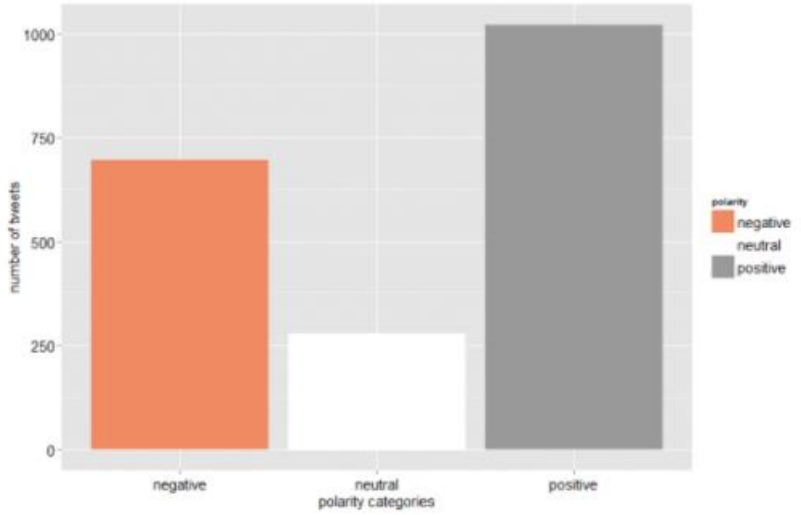
Contents

1. Introduction

Twitter is one of the most popular social media platforms in the world, with 330 million monthly active users and 500 million tweets sent each day. By carefully analyzing the sentiment of these tweets — whether they are positive, negative, or neutral, for example — we can learn a lot about how people feel about certain topics.







1.1 Problem Statement

1.Business Problem Understanding –

Understanding the sentiment of tweets is important for a variety of reasons: business marketing, politics, public behavior analysis, and information gathering are just a few examples. Sentiment analysis of Twitter data can help marketers understand the customer response to product launches and marketing campaigns, and it can also help political parties understand the public response to policy changes or announcements.

2.Business Objective –

The objective of this project is to predict whether a tweet has a positive sentiment or negative sentiment. Twitter sentiment analysis is a great way to find out what might the reasons for the customers to like our services or dislike service. For example: Let us consider the airline industry. By looking at the tweets that the customers post on the twitter, we can find out from the positive tweets what things people like in our services. And from negative tweets we can find out what they dislike in the services. From this the airline company can improve the services where it lacks so that the customers are satisfied. In this way it can attract more customers and generate higher revenue.

3.Approach –

Through the course of this project, we are going to look at twitter dataset. The dataset being used is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the Twitter API. The tweets have been annotated (0 = Negative, 4 = Positive) and they can be used to detect sentiment. We are going to do various text processing, and embedding techniques, and then employ a machine learning model to process our data. 4. Conclusion - After building the Twitter sentiment analysis model we can predict that a tweet belongs to particular class. We will build different models and will use the one which gives maximum accuracy.

Research shows that a single negative review may cost a company up to 30 clients. Therefore, effective analysis of reviews and identifying the reason behind negative feedback plays a crucial role in building and maintaining a strong positive brand image.

1.2 Objective

The primary objective of our project is to:

1.Find the key area, gaps identified in the topic survey where the project can add value to the customers and business

Twitter Sentiment Analysis is a great way to add value to many aspects of Business nowadays. For example by analyzing a person tweet we can find whether the tweet has a positive sentiment or negative sentiment. Business can use this as an opportunity to improve their services based on customers tweets. They can find out where they lack and where they are good in terms of their service. Various business where Twitter sentiment analysis can be used are business marketing, politics, public behaviorr analysis, information gathering etc.

2.What key gaps are you trying to solve?

Emoji Problem - Sentiment analysis for Twitter messages (tweets) is regarded as a challenging problem because tweets are short and informal. The problem with social media content that is text-based, like Twitter, is that they are inundated with emojis. Most analysis solutions treat emojis like special characters that are removed from the data during the process of text mining. But doing so means that companies will not receive holistic insights from the data.

1.2 Scope

- Consecutive Letter Problem - Also while posting tweets people use to write a word in such a way that they repeat the last alphabet in the word many times. It makes difficult to understand the meaning of these words.
- Stop Word Problem -There are a lot of words which do not add meaning to a sentence.They can safely be ignored without sacrificing the meaning of the sentence. (eg: "the", "he", "have") The problems we are trying to solve are:
- 1.We are trying to solve the emojis problem. We are replacing emojis by using a pre-defined dictionary containing emojis along with their meaning. (eg: ":)" to "EMOJIsmile").
- 2.Removing Consecutive letters: 3 or more consecutive letters are replaced by 2 letters. (eg: "Heyyyy" to "Heyy")
- 3.Removing Stop words: Stop words are the English words which does not add much meaning to a sentence

1.3 Data Source

The freely accessible information-set utilized in this project comes from Kaggle's Twitter Sentiment Analysis data.

The dataset being used is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the Twitter API. The tweets have been annotated (0 = Negative, 4 = Positive) and they can be used to detect sentiment.

It contains the following 6 fields:

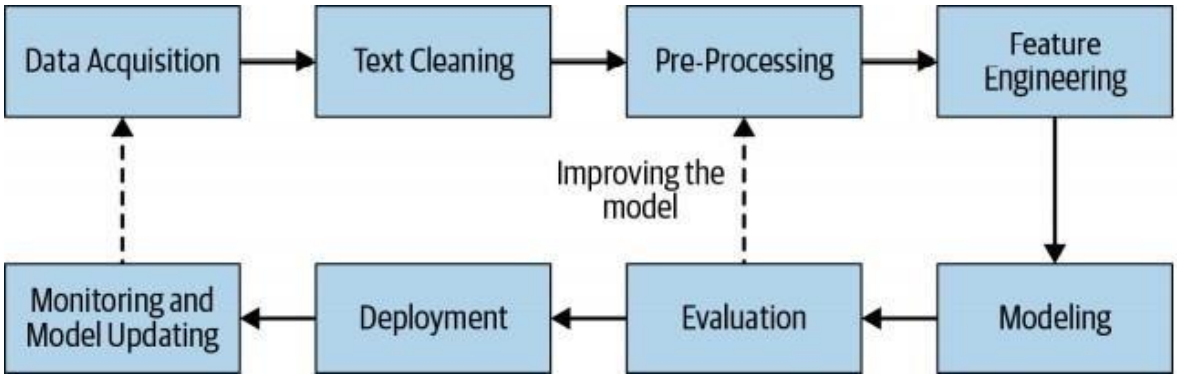
1.Sentiment: the polarity of the tweet (0 = negative, 4 = positive)
2.Ids: The id of the tweet (2087)
3.Date: The date of the tweet (Sat May 16 23:58:44 UTC 2009)
4.Flag: The query (lyx). If there is no query, then this value is NO_QUERY.
5.User: The user that tweeted (robotickilldozr)
6.Text: The text of the tweet (Lyx is cool)

1.4 Tools And Techniques

We have used following tools and methods for analysing and processing data:

There are series of steps involved in building any machine learning model. The step-by-step sequence of processing the data is known as a Pipeline. Some of these steps are commonly shared by every machine learning project and there are additional specific processing techniques which are specific to NLP projects. The key stages in the pipeline are as follows:

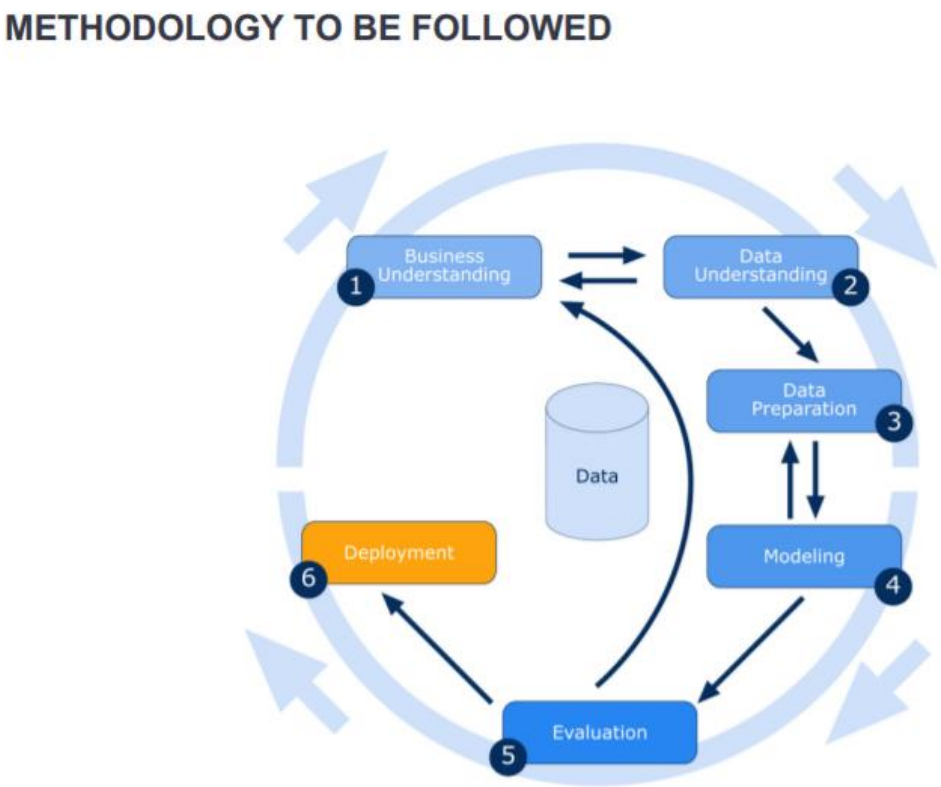
1. Data acquisition
2. Text cleaning
3. Pre – processing
4. Feature Engineering
5. Modelling
6. Evaluation
7. Deployment
8. Monitoring & model updating



1.5 Analytical Approach

Our approach towards project included following steps:

- Importing necessary libraries
- Loading and reading the dataset
- Exploratory Data Analysis
- Visualizing Target and independent Variables
- Data Preprocessing
- Train Test Split- Data should be split into train and test with appropriate test size
- Dataset Transformation using TF-IDF Vectorizer
- Model Evaluation
- Model selection
- Prediction generation
- Conclusion & Future work



Analytical methodology to be followed for model building and deployment

2. Data Preparation and Visualization

2.1 Dataset Information

In this project we will analyse the twitter sentiment analysis dataset. By using the dataset, we will build a model. The model can be used to predict whether a tweet has a positive sentiment or a negative sentiment. This is helpful for many industries like business marketing, politics, public behavior analysis, and information gathering are just a few examples. Sentiment analysis of Twitter data can help marketers understand the customer response to product launches and marketing campaigns, and it can also help political parties understand the public response to policy changes or announcements.

2.2 Data Dictionary

The dataset used in this project comes from Kaggle's Twitter sentiment analysis data which have been combined into a single dataset. In total, we have 16 lakh records.

```
In [2]:
# Importing dataset
# The dataset being used is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the Twitter A
# have been annotated (0 = Negative, 4 = Positive) and they can be used to detect sentiment.

# Importing the dataset
DATASET_COLUMNS  = ["sentiment", "ids", "date", "flag", "user", "text"]
DATASET_ENCODING = "ISO-8859-1"
dataset = pd.read_csv('training.1600000.processed.noemoticon.csv',
                      encoding=DATASET_ENCODING , names=DATASET_COLUMNS)

dataset.head()
```

Out[2]:

	sentiment	ids	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

```
In [3]:
dataset.shape
```

```
Out[3]: (1600000, 6)
```

Name	Description	Data type
sentiment	Polarity of the tweet	Integer
ids	The id of the tweet	Integer
date	The date of the tweet	Object
flag	The query (lyx). If there is no query, then this value is NO_QUERY.	Object
user	The user that tweeted	Object
text	The text of the tweet	Object

Table 2.2 – Data dictionary for Twitter Sentiment Analysis

2.3 Data Preprocessing

The base dataset for our analysis. The shape of the dataset is (1600000,6). Overview of the dataset is shown in **Figure 2.3.1**.

```
In [2]:
# Importing dataset
# The dataset being used is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the Twitter A
# have been annotated (0 = Negative, 4 = Positive) and they can be used to detect sentiment.

# Importing the dataset
DATASET_COLUMNS  = ["sentiment", "ids", "date", "flag", "user", "text"]
DATASET_ENCODING = "ISO-8859-1"
dataset = pd.read_csv('training.1600000.processed.noemoticon.csv',
                      encoding=DATASET_ENCODING , names=DATASET_COLUMNS)

dataset.head()
```

Out[2]:

	sentiment	ids	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

```
In [3]:
dataset.shape
```

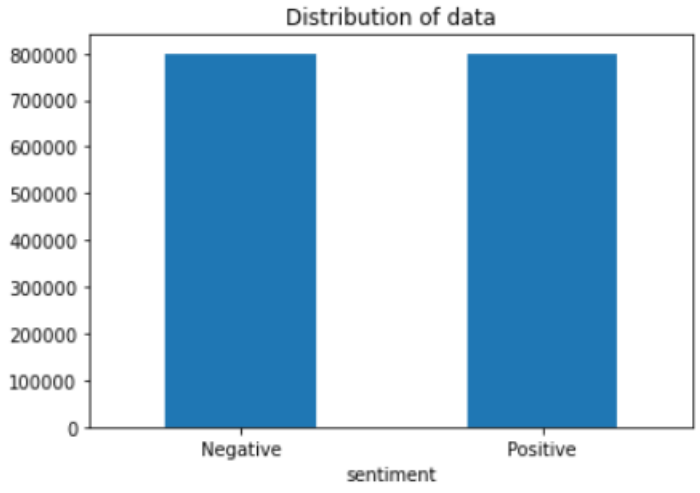
```
Out[3]: (1600000, 6)
```

Figure 2.3.1 – Overview of the dataset

2.3.1 Data Understanding

The dataset being used is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the Twitter API.

Equal number of Positive and Negative tweets are observed in the dataset.




```
dataset["sentiment"].value_counts()
```

```
1    800000
0    800000
Name: sentiment, dtype: int64
```

2.3.2 Count of Missing Values

Count of Missing Values or Null Values are Zero in the dataset.

```
dataset.isnull().sum()
```

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

2.3.3 Redundant Columns

Based on the objective of our project, there are some columns which are redundant and will not be used in the model building process. Therefore, these columns/features can be dropped. The list of columns dropped from the dataset are:

- ids
- date
- flag
- feature

	sentiment	text
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....

Figure 2.3.3 Dataset after removing redundant columns

2.3.4 Handling Null Values

The review dataset is checked for null values and there are about Zero null values in the dataset.

2.4 Preparing the Target Variable

Since our objective is to detect the sentiment of the tweets i.e., 1 - positive and 0 - negative. After filtering we get the following no. of records as shown in table below.

Polarity	No. of records	%
1 - positive	800000	50
0 - negative	800000	50

Table 2.4 – Number of records for 1- & 0- polarity

2.5 Creating a Balanced Dataset Target

From the above table 2.2, we see 0 and 4 as the target variable. Hence, 4-polarity in the target variable is replaced by 1.

	sentiment	text
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....

Figure 2.3 – Balanced dataset with 0 & 1 as target variable

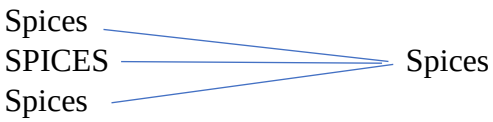
3. Text Preprocessing

Text is the least organized of all data types, which means we have a lot of cleaning to do. These pre-processing processes aid in the conversion of noise from high-dimensional characteristics to low-dimensional space, allowing for the extraction of as much correct information from the text as feasible. Simply said, preprocessing text data implies converting it into a format that is predictable and analyzable for your purpose.

Following changes were made to reviews column so that it can be fed into the model

3.1 Lower case conversion

Since python is case sensitive, the machine will treat upper and lower case differently.
The different capitalization variations in input (e.g., 'Product' vs. 'product') may result in different forms of output or no output at all. This was most likely due to the dataset's mixed-case occurrences of the term "Product". To overcome this, we have lowercased the text column of our dataset.



3.2 Accented characters conversion

Diacritics are marks placed above or below (or sometimes next to) a letter in a word to indicate a particular pronunciation. English letters with diacritics above or below them are known as accented characters. E.g., ü, ï. Our model may interpret same word with and without diacritics as two different words. This may lead to inconsistent and inaccurate results. Hence, we have these accents from the string data by using a Python module called Uni decode.

3.3 Expanding contracted words:

Contraction is the shortened form of a word (like don't, aren't which stands for do not and are not respectively) used frequently in sentences. Our model will consider these shortened words and their root words to be different words. To avoid this, we have expanded all the contractions in the text data with the help of a dictionary which had contracted words as keys and respective root word as values.

3.4 Other noise entity removal

- **Removing Stopwords**
  Stopwords are the most commonly occurring words in a text which do not provide any valuable information. Examples: they, there, this, were, etc. NLTK library includes approximately 180 stopwords which can be removed from the text data. It is also possible to add new stopwords using add() method if required. **However here, we are going to define a set of stop words and remove them from our text data. Note:** In our case, stop words like 'No', 'Not' have been excluded as they are important forms of representing negation/negative sentiments.

- **Replacing URLs**
  Links starting with "http" or "https" or "www" are replaced by "URL".

- **Replacing Emojis:**
  Replace emojis by using a pre-defined dictionary containing emojis along with their meaning. (eg: ":)" to "EMOJIsmile")

- **Replacing Usernames:**
  Replace @Usernames with word "USER". (eg: "@Kaggle" to "USER")

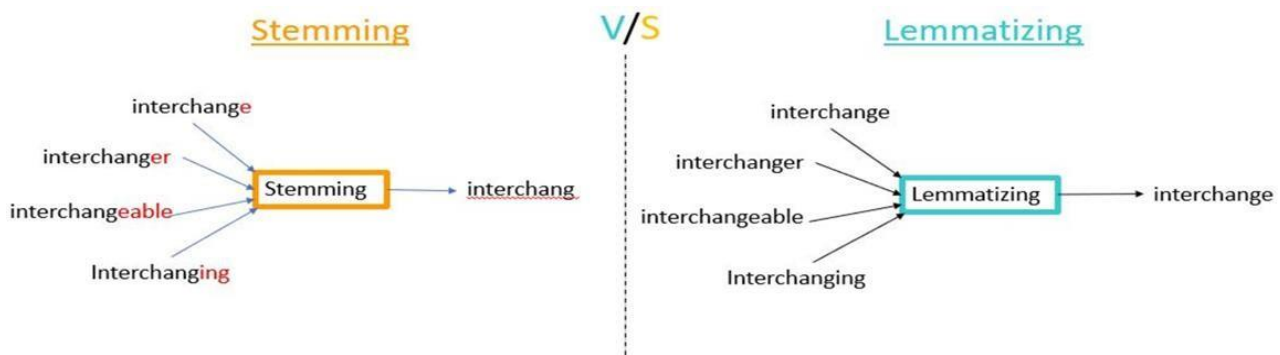
- **Removing Non-Alphabets:**
  Replacing characters except Digits and Alphabets with a space.

- **Removing Consecutive letters:**
  3 or more consecutive letters are replaced by 2 letters. (eg: "Heyyyy" to "Heyy")

- **Removing Short Words:**
  Words with length less than 2 are removed.

3.5 Lemmatization

It is the process where different inflected forms of a word are grouped together so they can be analysed as a single item. Lemmatization is similar to another concept called stemming but it results in more meaningful word forms. Text preprocessing includes both stemming as well as Lemmatization. Stemming is a crude process; it might randomly remove few letters from the end resulting in misspelled words that might not be found in dictionary. Lemmatization is preferred over Stemming because lemmatization performs morphological analysis of the words and leaves us with the base from which can be easily interpreted. Hence in our dataset we have performed lemmatization.



There are various packages available in python to perform lemmatisation. Here, we would be using the simple **WordNet Lemmatizer with pos (parts of speech tag)** of NLTK. A POS tag (or part-of-speech tag) is a special label assigned to each token (word) in a text corpus to indicate the part of speech and often also other grammatical categories such as tense, number (plural/singular), case etc.

3.6 Visualization using Word Cloud

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analysing data from social network websites. They provide you with quick and simple visual insights that can lead to more in-depth analyses. The bigger and bolder the word appears, the more often it's mentioned within a given text and the more important it is. We have created two different word cloud images for positive and negative reviews.

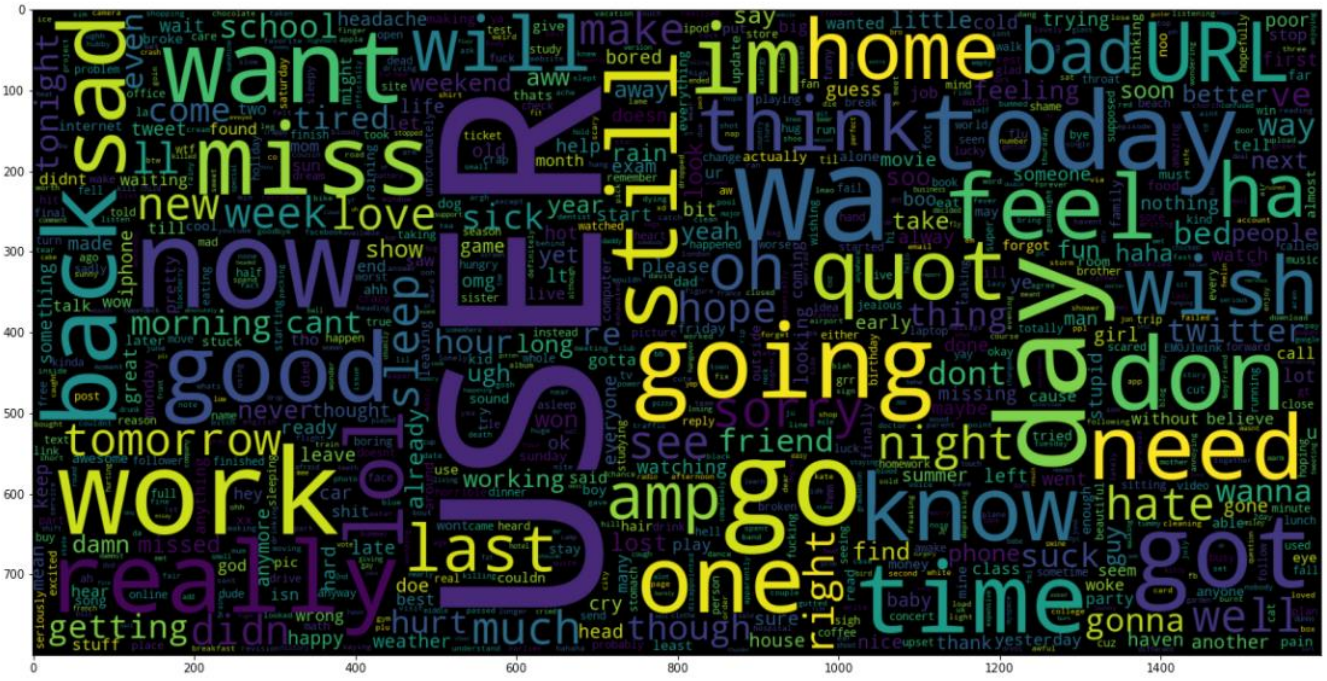


Figure 3.1 Word cloud for negative sentiment

We can see here that words like 'user', 'sad' are bigger which implies that they have been frequently used in 0-sentiment.

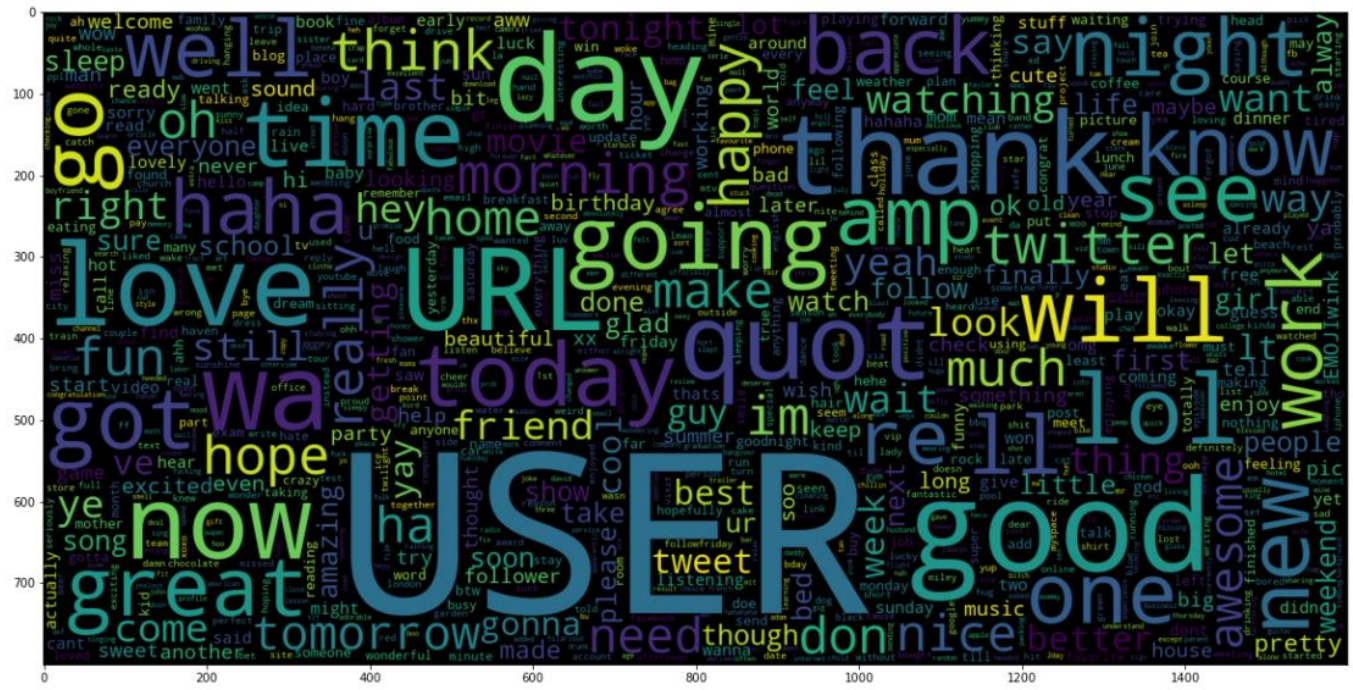


Figure 3.2 Word cloud for Positive sentiment

We can observe that words representing positive sentiments such as **"good", "user", "Love"** have been highlighted in a bigger font size which implies the frequencies of these words in records of 1-polarity is very much higher.

3.7 TRAIN- TEST SPLIT

Only the processed 'reviewText' column that has customer's review and 'overall' column that contains ratings in the form of 1 and 0 are required for model building. We split the data in the ratio of 95: 5 such that 95% of the data will be used for training the model and 5% will be used for testing purposes.

3.8 Text to Number Conversion

Algorithms use mathematics to train machine learning models. However, mathematics only works with numbers. To make statistical algorithms work with text data, we need to first convert text into numbers. To do so, we have approaches like Bag of Words, TF-IDF, Word2Vec, count vectorizer. In our project, we have used TF-IDF vectorization over all other approaches.

Term Frequency Inverse Document Frequency is a very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction. Count Vectorizer give number of frequencies with respect to index of vocabulary

whereas TF-IDF consider overall documents of weight of words. TF-IDF works by **penalizing the common words by assigning them lower weights while giving importance to words which are rare in the entire corpus but appear in good numbers in few documents**.

- **Term Frequency –** $\frac{\text{No. of times a term appears (say good) in a document}}{\text{Total No. of terms in the given document}}$
- **Inverse Document Frequency –** log(N/n)

$$\textbf{TF-IDF} = \textbf{TF} * \textbf{IDF}$$

Term frequency can be thought of as how often does a word '*w'* occur in a document *'d'*. More importance is given to words frequently occurring in a document. Idf is used to calculate rare words' weight across all documents in corpus. Words rarely occurring in the corpus will have higher IDF values.

4. Model Building

4.1 Model approach

After applying the TF – IDF vectorizer we have the data in the desired format to classify the sentiment of the text. we have made about 95:5 train test split i.e., 15,20,000 records for training and 80,000 records for testing.

1. Logistic Regression
2. Bernoulli Naive Bayes Model
3. Decision Tree Classifier
4. Random Forest Classifier
5. Ada Boost Classifier
6. Gradient Boosting Classifier

4.2 Logistic Regression

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes. After applying logistic regression, the following result is obtained

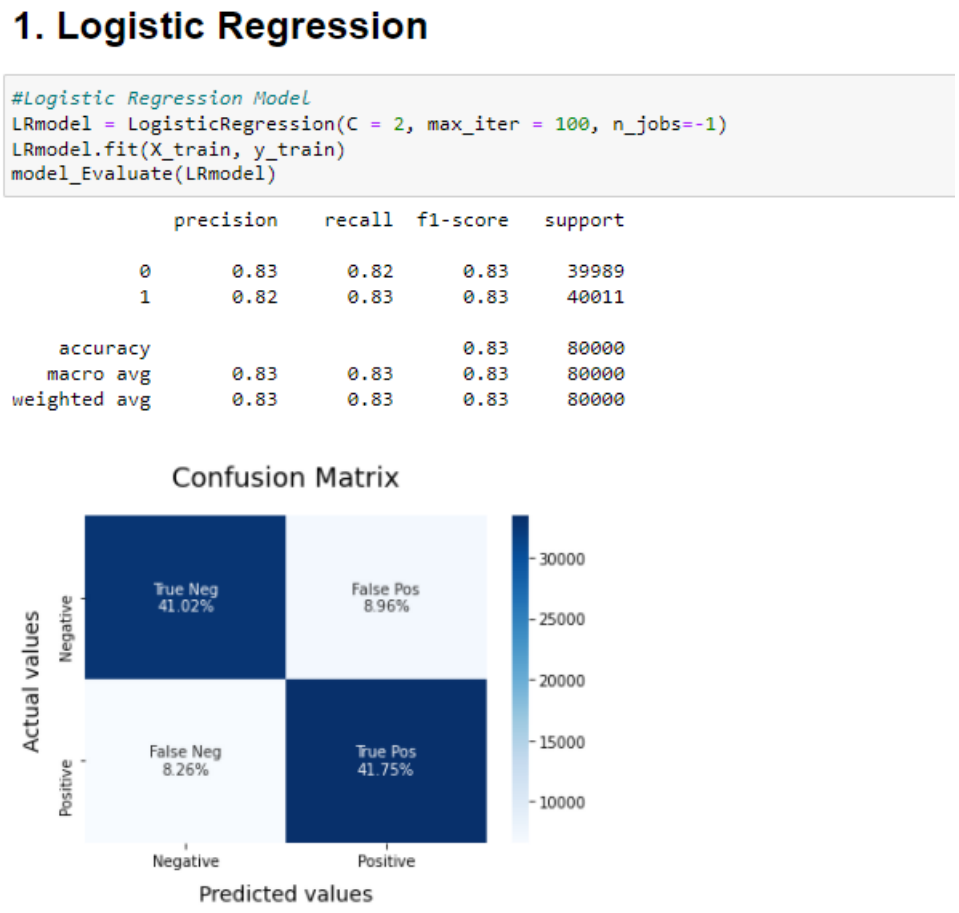


Figure 4.2 – Logistic Regression results

4.3 Bernoulli Naive Bayes Model

BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable,

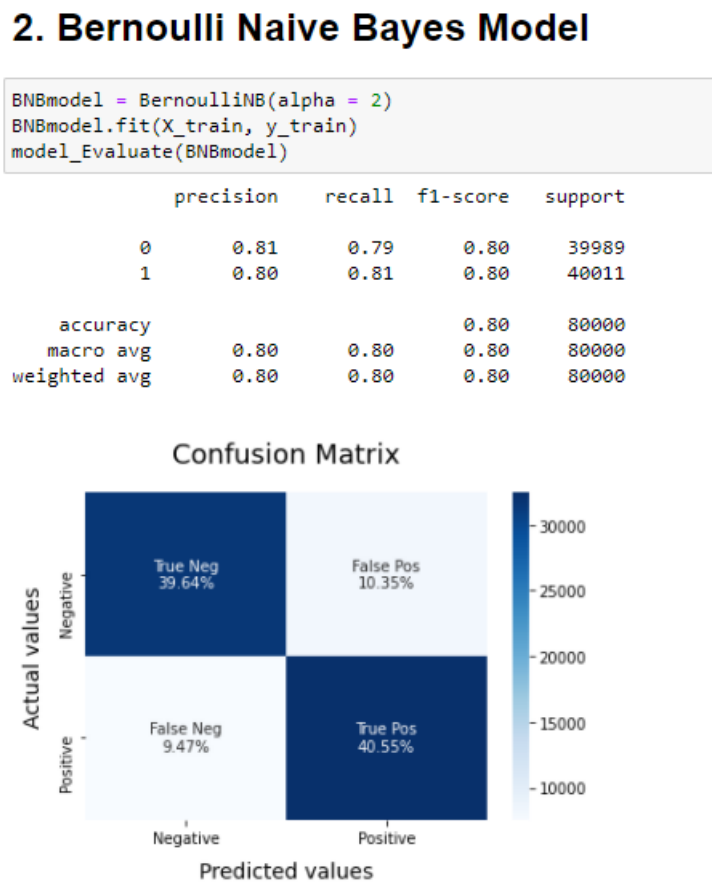


Figure 4.3 – Bernoulli Naive Bayes results

4.4 Decision Tree Classifier

Decision tree learning or induction of decision trees is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision making). This page deals with decision trees in data mining.

3. Decision Tree Classifier

```
dt = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=10)
dt.fit(X_train, y_train)
model_Evaluate(dt)
```

	precision	recall	f1-score	support
0	0.60	0.65	0.62	39989
1	0.62	0.57	0.59	40011
accuracy			0.61	80000
macro avg	0.61	0.61	0.61	80000
weighted avg	0.61	0.61	0.61	80000

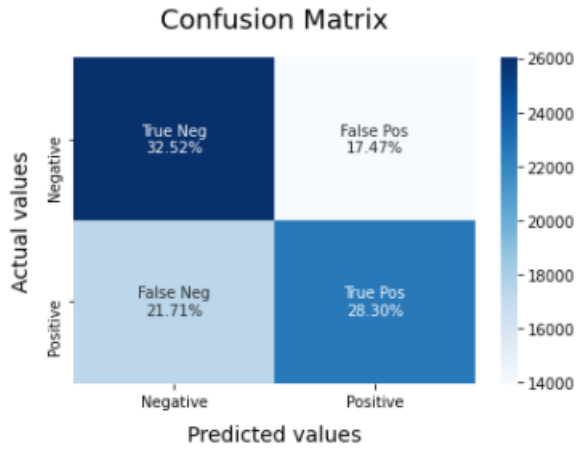


Figure 4.3 – Decision Tree Classifier results

4.5Random Forrest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. After applying Random Forest the following result is obtained

4.Random Forest Classifier

```
rfc = RandomForestClassifier(n_estimators=20, max_depth=4, criterion='entropy', max_features=0.3, max_samples=0.7)
rfc.fit(X_train, y_train)
model_Evaluate(rfc)
```

```
              precision    recall  f1-score   support

           0       0.61      0.65      0.63     39989
           1       0.63      0.59      0.61     40011

    accuracy                           0.62     80000
   macro avg       0.62      0.62      0.62     80000
weighted avg       0.62      0.62      0.62     80000
```

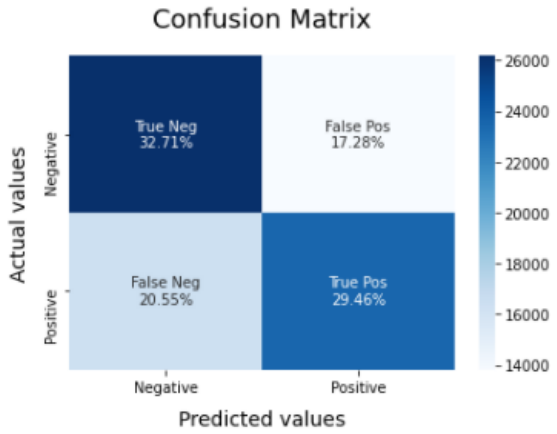


Figure 4.5– Random Forest results

4.6 AdaBoost

AdaBoost, short for Adaptive Boosting, is a statistic classification meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

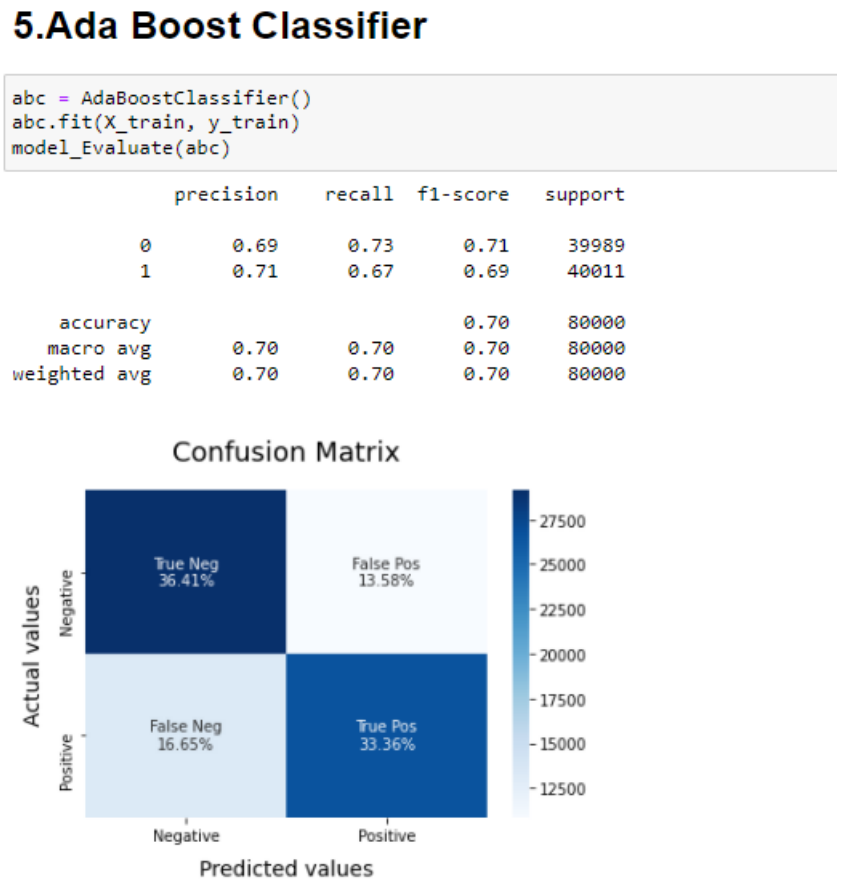


Figure 4.6 – XG Boost results

4.7 Gradient Boosting Classifer

Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting.

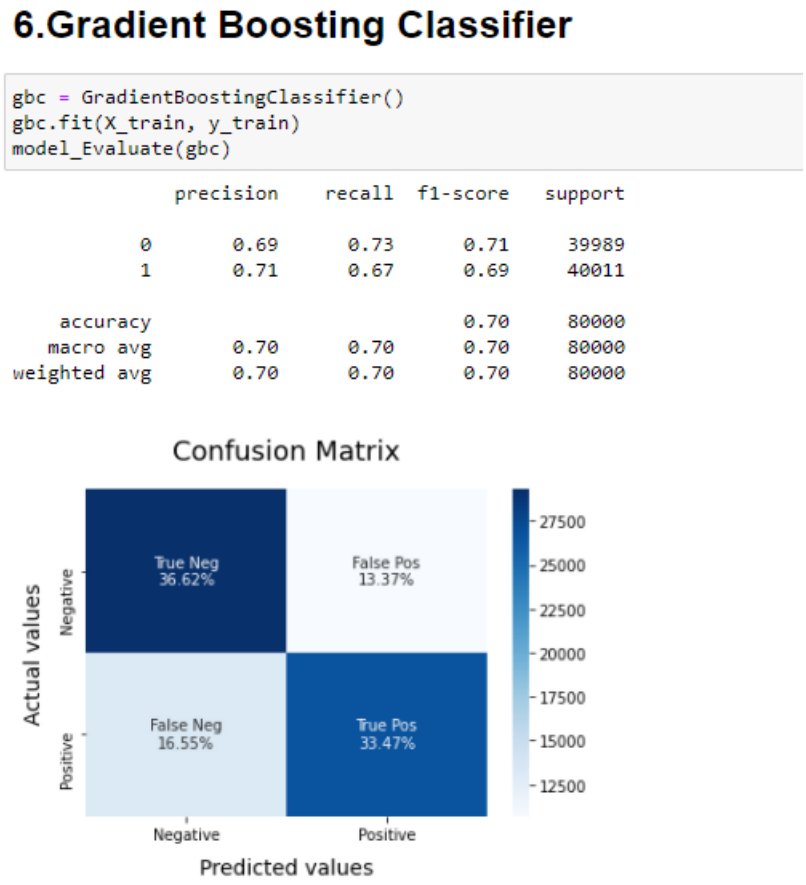


Figure 4.7 – Gradient Boosting results

Inferences

From the above figure which compares our 6 classifier models on basis of different evaluation metrices, we could observe the following:

4.2 Logistic Regression model is giving us best accuracy and F1-Score i.e., about 83%.
4.3 Bernoulli Naive Bayes model is giving us about 80% accuracy score.
4.4 Decision Tree model is giving about 61% accuracy score.
4.5 Random Forest classifier is giving about 62% accuracy score.
4.6 AdaBoost model is giving about 70% accuracy score.
4.7 Gradient Boosting model is giving about 70% accuracy score.

4.8 Hyperparameter Tuning

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning. Grid-search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions. We are doing hyperparameter tuning for Logistic regression and Bernoulli Naive Bayes as they are giving us the highest accuracy.

4.8.1 Tuning Logistic Regression

```
# Tunned Logistic Regression
params = {'C':[0.01,0.1,1,2,10,100]}
lr = LogisticRegression(max_iter = 100, n_jobs=-1)
tuned_lr = GridSearchCV(lr,param_grid=params,cv=3,scoring='f1_weighted')
tuned_lr.fit(X_train,y_train)
print("Best parameters:",tuned_lr.best_params_)
print("Best F1 score:",tuned_lr.best_score_)
```

```
Best parameters: {'C': 2}
Best F1 score: 0.8228636161572084
```

We can see that C = 2 is the optimal value for the Logistic Regression. We had used C = 2 for building the logistic regression model. It is giving the accuracy of 83 %

4.8.3 Tuning Bernoulli Naive Bayes

```
# Tunned Bernoulli Naive Bayes
params = {'alpha':[0.01,0.1,0.5,1,2,3,5,10]}
BNB = BernoulliNB()
tuned_bnb =  GridSearchCV(BNB,param_grid=params,cv=3,scoring='f1_weighted')
tuned_bnb.fit(X_train,y_train)
print("Best parameters:",tuned_bnb.best_params_)
print("Best F1 score:",tuned_bnb.best_score_)
```

```
Best parameters: {'alpha': 1}
Best F1 score: 0.7998025398381569
```

We can see that alpha = 1 is the optimal value for the Bernoulli Naive Bayes. But We had used alpha = 2 for building the Bernoulli Naïve Bayes model. It is giving the accuracy of 83 % which is approximately equal to what we got from alpha = 2.

5 Comparison and selection of models

Model Name	Accuracy
Logistic regression	**83%**
BNB model	**80%**
Decision tree classifier	**61%**
Random forest classifier	**63%**
Ada boost	**70%**
Gradient boosting	**70%**

As we can see Logistic regression is giving us the highest accuracy of 83 %.We will use it as our final model. Although Bernouli naive bayes is fastest to train but it is giving us the accuracy of 80 %

6 Results and discussion

We can clearly see that the Logistic Regression Model performs the best out of all the different models that we tried. It achieves nearly 83% accuracy while classifying the sentiment of a tweet.

Although it should also be noted that the BernoulliNB Model is the fastest to train and predict on. It also achieves 80% accuracy while calssifying.

7 Analyzing the misclassified tweets

We analyze the first 50 tweets of the processed tweets to find which tweets are wrongly classified. These are the ten tweets which we got from it. As we can see it is very difficult to interpret whether the sentiment is positive or negative after analyzing these tweets. Thus, we got accuracy of 83 %.

1. 'USER no it not behaving at all mad why am here because can see you all over there ', # Actual 1 predicted 0
2. 'just re pierced my ear ' # Actual 1 predicted 0
3. 'USER would ve been the first but didn have gun not really though zac snyder just doucheclown ' # Actual 1 predicted 0
4. 'USER ahh ive always wanted to see rent love the soundtrack '# Actual 0 predicted 1
5. 'USER baked you cake but ated it ' # Actual 0 predicted 1
6. 'USER cry my asian eye to sleep at night ' # Actual 0 predicted 1
7. 'ok sick and spent an hour sitting in the shower cause wa too sick to stand and held back the puke like champ bed now ' # Actual 0 predicted 1
8. 'USER sorry bed time came here gmt URL ' # Actual 0 predicted 1
9. 'he the reason for the teardrop on my guitar the only one who ha enough of me to break my heart ' # Actual 0 predicted 1

8 Saving the models

We're using PICKLE to save Vectoriser and BernoulliNB, Logistic Regression Model for later use.

```
file = open('vectoriser-ngram-(1,2).pickle','wb')
pickle.dump(vectoriser, file)
file.close()


file = open('Sentiment-LR.pickle','wb')
pickle.dump(LRmodel, file)
file.close()


file = open('Sentiment-BNB.pickle','wb')
pickle.dump(BNBmodel, file)
file.close()
```

9 Random Tweets to our model

Using the Model. To use the model for Sentiment Prediction we need to import the Vectoriser and LR Model using Pickle.The vectoriser can be used to transform data to matrix of TF-IDF Features. While the model can be used to predict the sentiment of the transformed Data. The text whose sentiment has to be predicted however must be preprocessed.

In [66]:
```
def load_models():
    '''
    Replace '..path/' by the path of the saved models.
    '''

    # Load the vectoriser.
    file = open('..path/vectoriser-ngram-(1,2).pickle', 'rb')
    vectoriser = pickle.load(file)
    file.close()
    # Load the LR Model.
    file = open('..path/Sentiment-LRv1.pickle', 'rb')
    LRmodel = pickle.load(file)
    file.close()

    return vectoriser, LRmodel
```

In [67]:
```
def predict(vectoriser, model, text):
    # Predict the sentiment
    textdata = vectoriser.transform(preprocess(text))
    sentiment = model.predict(textdata)

    # Make a list of text with sentiment.
    data = []
    for text, pred in zip(text, sentiment):
        data.append((text,pred))

    # Convert the list into a Pandas DataFrame.
    df = pd.DataFrame(data, columns = ['text','sentiment'])
    df = df.replace([0,1], ["Negative","Positive"])
    return df
```

In [68]:
```
if __name__=="__main__":
    # Loading the models.
    #vectoriser, LRmodel = load_models()

    # Text to classify should be in a list.
    text = ["I hate twitter",
            "May the Force be with you.",
            "Mr. Stark, I don't feel so good"]

    df = predict(vectoriser, LRmodel, text)
    print(df.head())
```

```
                          text sentiment
0                I hate twitter  Negative
1      May the Force be with you.  Positive
2  Mr. Stark, I don't feel so good  Negative
```

In [ ]:
```
# We gave some random tweets from ourself to our model.As we can see from above our model is able to predict them correctly.
# Hence it is good model.
```

We gave some random tweets from ourself to our model. As we can see from above our model is able to predict them correctly. Hence it is good model.