# Twitter Sentiment Analysis


| | | |
|---|---|---|
| Member 1 | Venkata Naga Sai Rama Krishna Pinnimty | ramapinnimty@vt.edu |
| Member 2 | Sai Deepak Gattidi | saideepakg@vt.edu |


## Introduction

Andrew Ng, a pioneer in Artificial Intelligence (AI), famously quoted that data will be the new electricity. He said that just as electricity transformed everything 100 years ago, AI will transform every known industry in the coming years. Many of the applications we use today generate a huge amount of data. Thus, the task of analyzing this data and delivering valuable insights to the right audience is crucial. One such medium is Twitter where millions of users tweet about their views on a wide variety of topics. While everyone has a right to voice their opinion, sometimes it can lead to heated arguments thereby mentally affecting all the people involved in the conversation. So, using techniques like Sentiment Analysis, we can flag the tweets that contain hate speech and help in reducing the toxicity online.

## Problem Statement

The problem we are trying to solve through this project is to identify and classify the sentiment expressed in tweets. This is especially useful when we want to assess the public pulse on any ongoing issue. Furthermore, any company that tries to build its brand and reputation through customer feedback can use Sentiment Analysis to capture the opinions of its users. For instance, Amazon uses it to separate the positive and the negative reviews for any listed product. Facebook and Twitter use it to censor content that might potentially be contentious. This problem is best approached as a Data Analytics task as manual processing of tweets requires a significant amount of time and effort. Instead, we can automate this process by training a Machine Learning model that can capture the common patterns. The model then tries to understand how each word influences the overall tone being positive, negative, or neutral and can apply this newly acquired knowledge to classify unseen tweets.

## Dataset

We are using the *Sentiment140*[1] dataset openly available on Kaggle. It was curated by the graduate students at Stanford University using the Twitter API. In total, the dataset contains 1.6 million records that are described using 6 feature columns. The dataset is well balanced with each class containing about 800K samples. The list of features along with their type and brief description can be found in Table-1.

One interesting fact about the dataset is that the annotation was automatically created without the need for manual labeling. This was done by considering the tweets with positive emoticons as positive and tweets with negative emoticons as negative.

[1] "*Sentiment140 - A Twitter Sentiment Analysis Tool*", 2016, http://help.sentiment140.com/home

| Feature | Type | Description | Example |
|---|---|---|---|
| id | Nominal | A unique identifier for each tweet. | 1467810917 |
| date | Ordinal | The date of the tweet. | *Sat May 16 23:58:44 UTC 2009* |
| flag | Categorial | The input query. If there is no query, then this value is *NO_QUERY*. | NO_QUERY |
| user | Nominal | Name of the user who tweeted. | swinspeedx |
| text | Textual | The text of the tweet. | @B0RNASTARtrell Oh no. IDK if I even wanna know. |
| **target** | Categorical | The polarity of the tweet (`0` = negative, `4` = positive). | 0 |

**Table 1:** *Sentiment140* Dataset Features and Description

### Machine Learning Pipeline

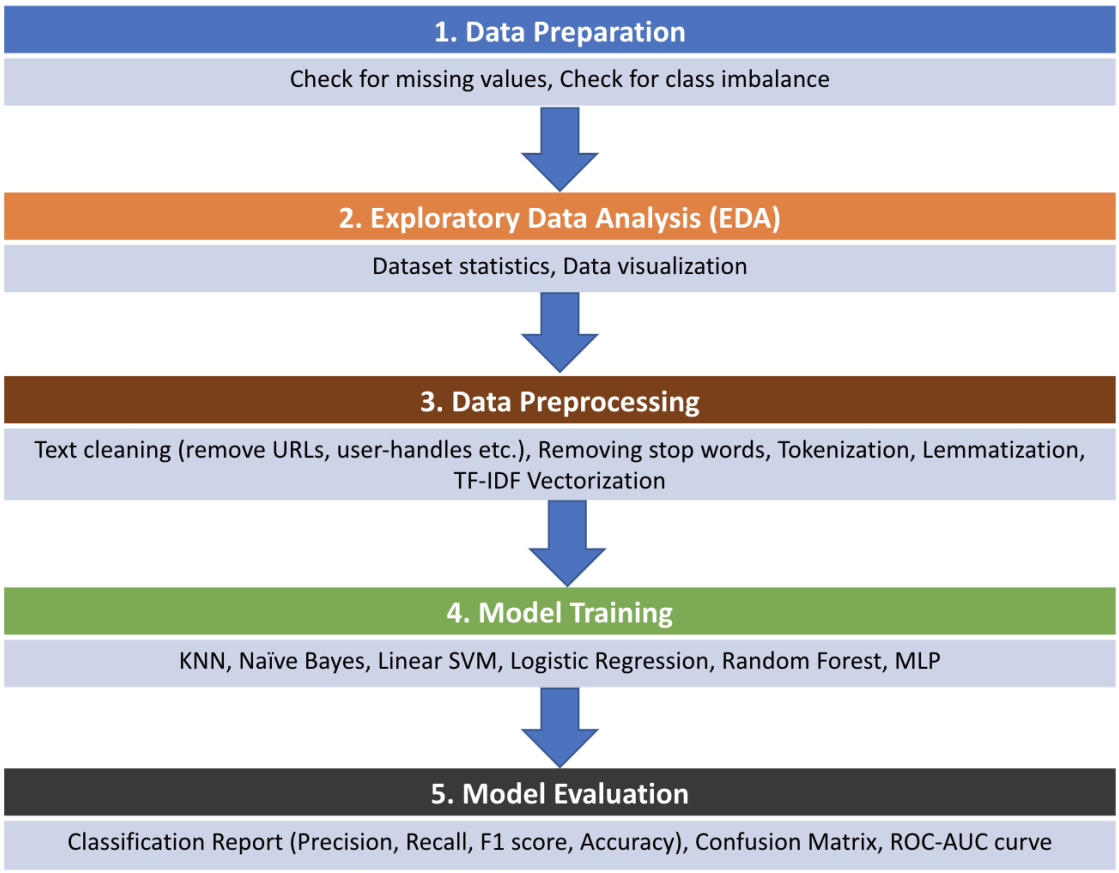



Figure 1

**Data Preparation**

Generally, we have to look for missing values, duplicate rows and/or columns so that we can discard them or perform some imputation on them. Fortunately, there are no missing values and there is no class imbalance in the dataset. We'll pick only `200K` samples from the dataset so that we can train the ML models in time.

**Data Visualization**

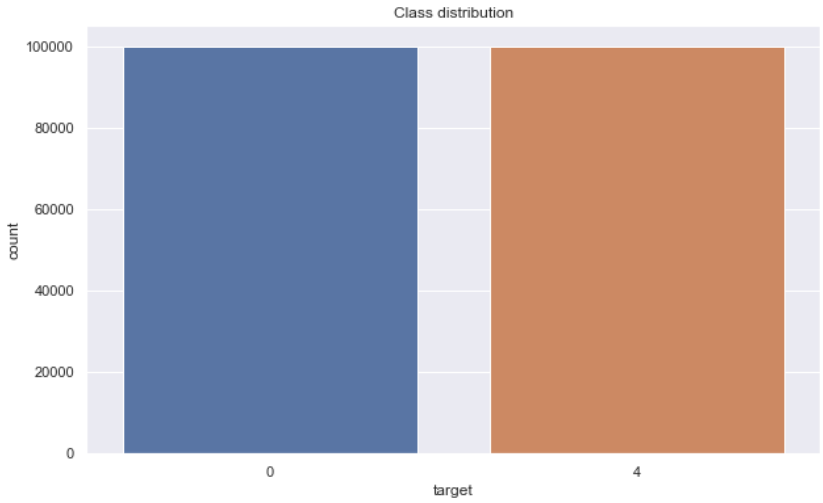



Figure 2

From the countplot in Figure-2, we can see that the dataset is balanced with `100K` samples in each class.

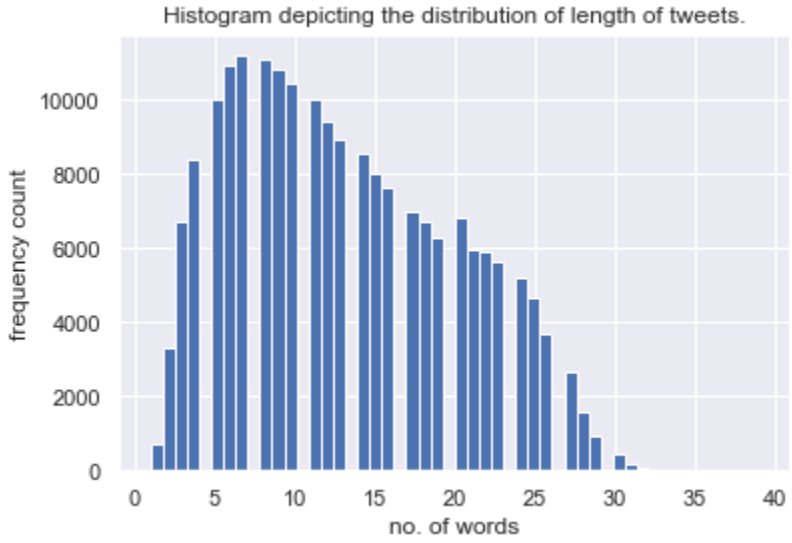



Figure 3

From the histogram in Figure-3, we can clearly see that most of the tweets are under `30` words.

Shortest tweet : `'Agh...snow!!! '`

Longest tweet : `'4 I am high &amp; u can see. 4 I am lost &amp; u find me. 4 I am held &amp; u break free. Well I am am &amp; u can only wish to be of such a masterpiece. '`

**Data Pre-processing**

Before we actually start training a model, we need to preprocess the raw tweets. This is because they often contain irrelevant information like user-handles, hyperlinks, punctuations, etc. Thus, in order to learn a good language model, we have to first clean the text using various pre-processing techniques. Primarily, it includes the following six major steps: -

Also for better illustration purposes, let's use the text from the first training sample and check the resulting output after each step.

**Input: -**

`@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D`

1) Text cleaning -

a. *Converting to lowercase*: for convenience and uniformity when performing operations.

**Output: -**

`@switchfoot http://twitpic.com/2y1zl - awww, that's a bummer.  you shoulda got david carr of third day to do it. ;d`

b. *Removing repeating characters*: getting rid of redundant letters.

**Output: -**

`@switchfoot http://twitpic.com/2y1zl - aww, that's a bummer.  you shoulda got david carr of third day to do it. ;d`

c. *Removing user-handles and URLs*: these are unnecessary as they do not have any useful information pertinent to the task at hand.

**Output: -**

`- aww, that's a bummer.  you shoulda got david carr of third day to do it. ;d`

d. *Removing punctuations*: getting rid of unwanted noise.

**Output: -**

`aww thats a bummer  you shoulda got david carr of third day to do it d`

e. *Discarding numeric values*: numbers neither carry nor add any weight to the sentiment in a text.

**Output: -**

`aww thats a bummer  you shoulda got david carr of third day to do it d`

2) Removing stop words - Stop words are commonly appearing words (such as “a”, “an”, “the”, “in” etc.) which do not help in discriminating between a positive and a negative tweet. We do not want these words to take up space and our valuable processing time.

**Output: -**
```
aww thats bummer shoulda got david carr third day
```

3) Tokenization - Tokenization is the process of breaking down a given text into its smallest units called tokens. We do this so that the models can understand text more easily and perform operations effectively.

**Output: -**
```
['aww', 'thats', 'bummer', 'shoulda', 'got', 'david', 'carr', 'third', 'day']
```

4) Stemming - Stemming is the process of reducing a word to its root/base. A word stem need not be the same as a dictionary-based morphological root, it's just the same or a smaller form of the word. *We discarded this for reasons cited in the challenges.*

**Output: -**
```
['aww', 'that', 'bummer', 'shoulda', 'got', 'david', 'carr', 'third', 'day']
```

5) Lemmatization - The aim of lemmatization is to reduce inflectional forms to a common base form. It is different from stemming in that it involves an even longer process to calculate. As opposed to stemming, lemmatization does not simply chop-off inflections. Instead, it uses lexical knowledge to get the correct base forms of the words.

**Output: -**
```
['aww', 'that', 'bummer', 'shoulda', 'got', 'david', 'carr', 'third', 'day']
```

6) Data Transformation - After performing all the above pre-processing steps, we have to transform our data using the *TF-IDF Vectorizer*. This is a technique used to quantify words in a set of documents and we generally compute a score for each word to signify its importance in the document and the corpus. It assigns a value to a term according to its importance in a document scaled by its importance across all documents in a corpus. Therefore, it essentially eliminates all the naturally occurring words and selects words that are more relevant.

   For our case, we chose to use both unigrams and bigrams and generate `50K` features.

**Methods and Models**

The task requires us to build a Machine Learning model that can easily tell apart a positive tweet from a negative tweet. This is an example of a Supervised Machine Learning problem that comes under binary classification.

We plan to implement K-Nearest Neighbors (kNN), Bernoulli Naïve Bayes, Linear-Support Vector Machine (SVM), Logistic Regression, Random Forest, and Multi-Layer Perceptron (MLP) models and also perform a detailed comparison. 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 overall performance.

Brief intuition behind the models: -

- k-Nearest Neighbors: This algorithm works by finding the training examples closest to the test example. Furthermore, it doesn't need any training time (i.e, training time is 0).
- Naïve Bayes: This algorithm is known to work well for many text classification problems and requires relatively few training examples. We used the Bernoulli variant of Naïve Bayes as we are dealing with a binary classification problem.
- Support Vector Machine: Like Naïve Bayes classifiers, support vector classifiers also work well for text classification and require relatively few training examples.
- Logistic Regression: Logistic Regression is a classifier that serves to solve the binary classification problem (In our case, determining positive or negative). It is one of the simplest models to understand and implement. Also, it works well with text classification.
- Random Forest: Generally, Decision Trees often do a good job of learning to classify. This algorithm takes it one step further by using an ensemble of decision trees. It is widely known for being both fast and achieving high accuracy.
- Multi-Layer Perceptron: MLP usually works well for large datasets and they can be modeled to solve any kind of problem as they are universal solvers (a.k.a. universal approximators).

**Results**

We'll be performing 3-fold cross-validation along with Grid Search to determine the best set of parameters. Additionally, in order to validate the performance of our models, we'll use the following evaluation metrics: -

1. Accuracy
2. Precision, Recall, and F1-score
3. Confusion Matrix
4. ROC-AUC curve

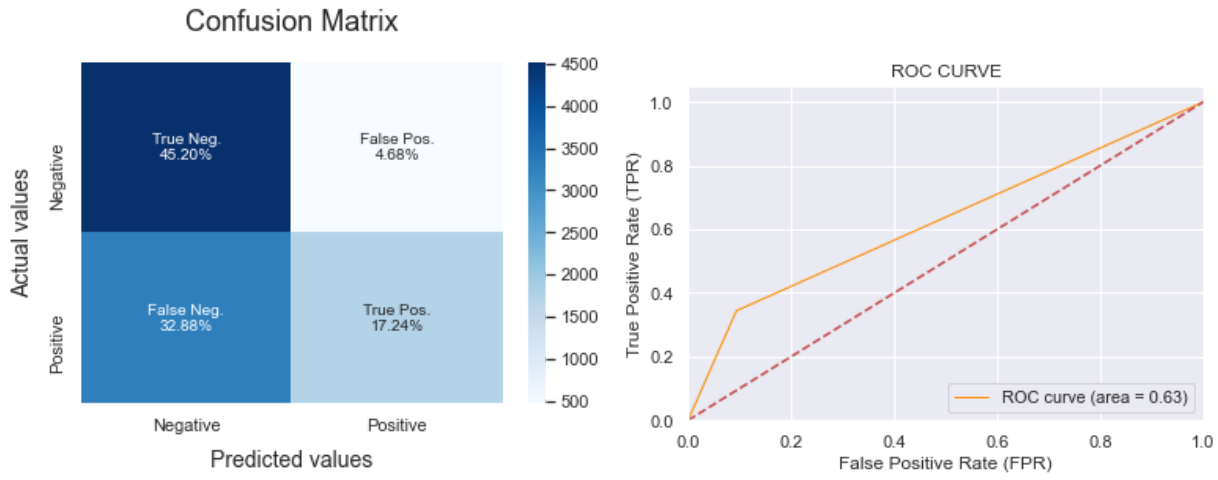
| Model | Accuracy | Precision | | Recall | | F1-score | |
|---|---|---|---|---|---|---|---|
| | | Class-0 | Class-1 | Class-0 | Class-1 | Class-0 | Class-1 |
| **KNN** | 62% | 0.58 | **0.79** | **0.91** | 0.34 | 0.71 | 0.48 |
| **Naïve Bayes** | 78% | **0.80** | 0.77 | 0.75 | **0.81** | 0.77 | 0.79 |
| **Linear SVM** | 78% | 0.79 | 0.78 | 0.77 | 0.80 | 0.78 | 0.79 |
| **Logistic Regression** | **80%** | **0.80** | **0.79** | 0.78 | **0.81** | **0.79** | **0.80** |
| **Random Forest** | 75% | 0.78 | 0.73 | 0.70 | **0.81** | 0.74 | 0.77 |
| **MLP** | 76% | 0.78 | 0.74 | 0.71 | 0.80 | 0.75 | 0.77 |

**Table 2:** Table showing the results of different models using various metrics

**K-Nearest Neighbors (kNN)** :

Best parameters: {*'n_neighbors'*: 3}

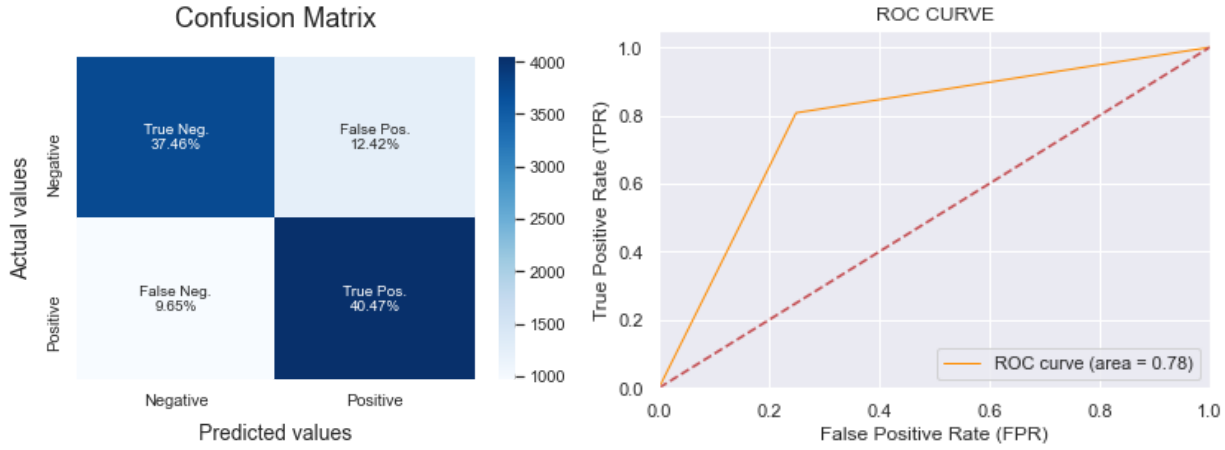
ROC-AUC score = `0.63`



**Bernoulli Naïve Bayes** :

Best parameters: {*'alpha'*: 2.0}

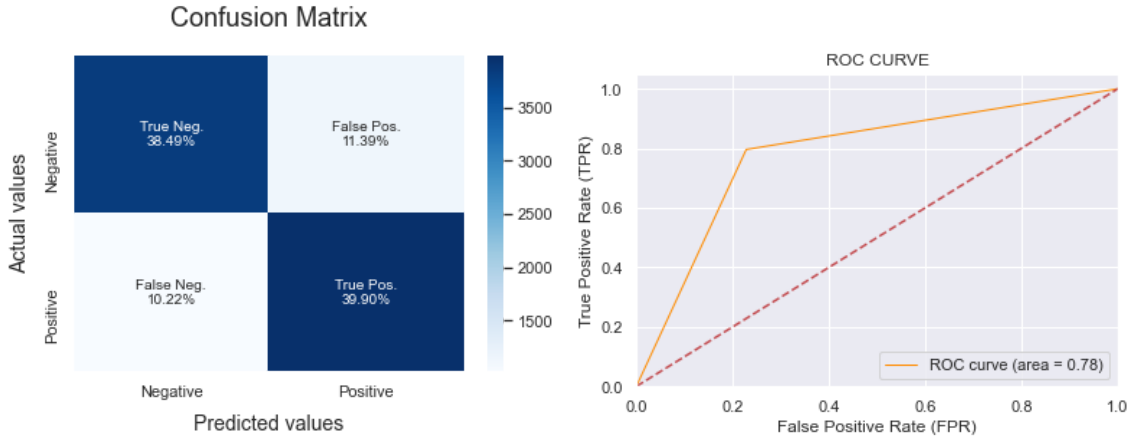
ROC-AUC score = 0.78




**Linear-Support Vector Machine (SVM)** :

Best parameters: {*'C'*: 1,
*'dual'*: False}

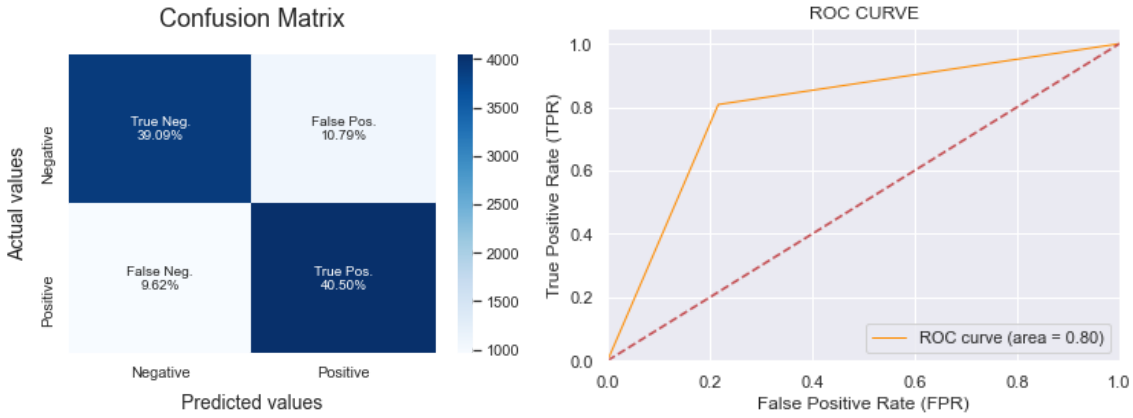
ROC-AUC score = 0.78



**Logistic Regression** :

Best parameters: {*'C'*: 2.782559402207126,
*'max_iter'*: 1000,
*'solver'*: 'liblinear'}

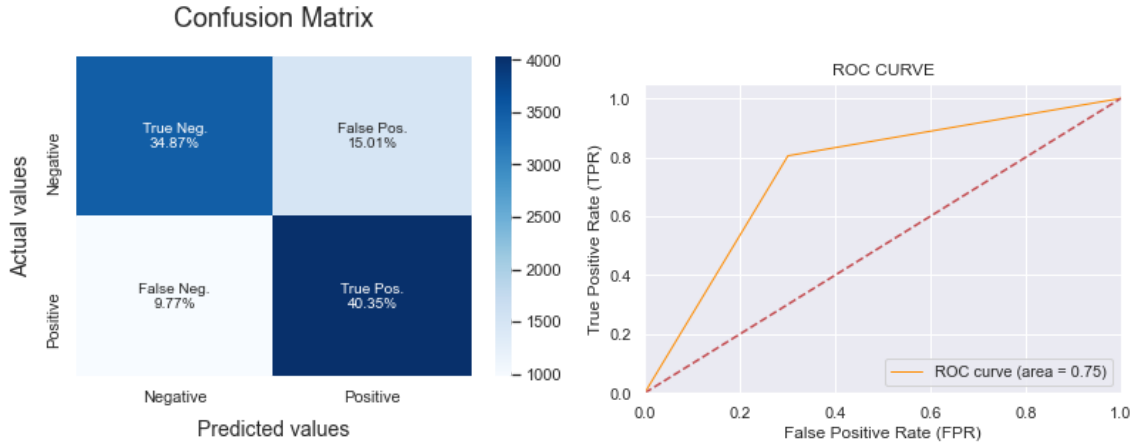
ROC-AUC score = **0.80**




**Random Forest** :

Best parameters: {*'max_depth'*: 50,
*'n_estimators'*: 200}

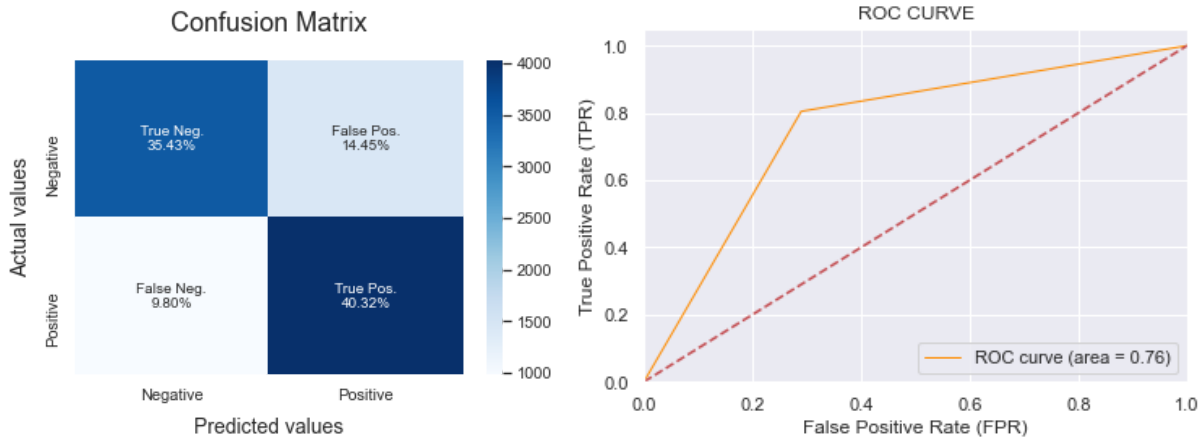
ROC-AUC score = 0.75



**Multi-Layer Perceptron (MLP)** :

Best parameters: {*'activation'*: 'relu',
*'alpha'*: 0.05,
*'hidden_layer_sizes'*: (10, 30, 10),
*'learning_rate'*: 'constant',
*'solver'*: 'sgd'}

ROC-AUC score = `0.76`




**Observations**

- Accuracy of the Logistic Regression model is the highest -- `80%`.
- Highest *Precision* for Class-`0` is achieved by Naïve Bayes & Logistic Regression models (`0.80`) and for Class-`1`, kNN & Logistic Regression models (`0.79`).
- Highest *Recall* for Class-`0` is achieved by the kNN model (`0.91`) and Naïve Bayes, Logistic Regression & Random Forest models (`0.81`) for Class-`1`.
- Highest *F1-score* for both Class-`0` (`0.79`) and Class-`1` (`0.80`) is achieved by the Logistic Regression model.
- We can therefore conclude that **Logistic Regression** is the best model for the given dataset. One way to look at this is that Logistic Regression is following the principle of *Occam's Razor*, which defines that for a particular problem, 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.
- Below table shows three sample tweets taken from the test set along with their sentiments as predicted by the Logistic Regression classifier.

| S.No | Input tweet | Output |
|---|---|---|
| 1. | @kenburbary You'll love your Kindle2. I've had mine for a few months and never looked back. The new big one is huge! No need for remorse! :) | Positive |
| 2. | After using LaTeX a lot, any other typeset mathematics just looks hideous. | Negative |
| 3. | Reading the tweets coming out of Iran... The whole thing is terrifying and incredibly sad... | Negative |

**Table 3:** Table showing sample tweets from the test set and their predictions

- By using stemming, we observed that the accuracy of our models decreased as it even reduced good and useful words.
- Using KNN with GridSearchCV proved to be expensive as we'll be computing the distances between every pair of points.
- We started out with `500K` features, but later decided to use only `50K` as it resulted in large sparse vectors and didn't really improve the model performance.

**Challenges**

1. Oftentimes, it is really difficult to decide on what is unnecessary in tweets.
2. Understanding sarcasm: Tweets may contain irony & sarcasm and the act of expressing negative sentiment using backhanded compliments. This can make things difficult for sentiment analysis tools to detect the true context of what the response is actually implying.
3. Emojis: In general, NLP tasks are trained to be language specific. While they can extract text from even images, emojis are a language in itself. We generally treat emojis like special characters that are removed from the data during the process of text mining. But doing so means that we might not receive holistic insights from the data. So, we employed a dictionary-based emotion-analyzer module that can decode the language in emojis and not just consider them to be special characters like commas, spaces or full stops. Still, this approach is a temporary fix as we need to update the dictionary over time as we encounter new emojis.
4. Negations: Words such as not, never, cannot, were not, etc. can confuse the models. For example, the algorithm fails to understand a phrase like, "I can't say no to my class reunion". To overcome this, the model has to be trained to understand that double negatives outweigh each other and turn a sentence into a positive. This can only be done when there is enough corpus to train the algorithm and has the

maximum number of negation words possible to make the optimum number of permutations and combinations.

While we can solve some of these issues by having precise and massive amounts of data, it requires a lot of computing power and time to train Machine Learning models.

**Future work**

Unfortunately due to time constraints, we could not deal with contractions in the tweets. Python packages such as "`contractions`" and "`pycontractions`" can be a good starting point. By using Feature Engineering, we could generate new features that capture the variance between the distributions of positive and negative tweets. We could also use different Vectorization methods such as Bag of Words (BoW), Word2Vec, etc. Deep Learning models like RNN, LSTM, BERT etc. tend to improve the accuracy beyond `80%`. Importantly, models need to be trained so that they can interpret idioms as well. Analyzing audio and video (if any) of tweets would produce even better results.

**Conclusion**

Towards the end of this project, we plan to build a fully functional Sentiment Classification model that can effectively classify tweets into positive and negative classes. The results can be used by the government and the department of justice to monitor communications. Some specific use cases can be identifying hate speech and crisis management. The model can be further tuned or extended to suit any domain of interest.

**Individual Contribution**

| | |
|---|---|
| Venkata Naga Sai Rama Krishna Pinnimty | ❖ Idea<br>❖ Data Pre-processing<br>❖ Results & Observations |
| Sai Deepak Gattidi | ❖ Dataset<br>❖ Data Preparation & Data Visualization<br>❖ Methods and Models |