PREDICT HATE TWEETS USING LOGISTIC REGRESSION. QMST 5343 |DATA MINING

Sowjanya Koka| Anjali Krishna| Chinmay Bake| Sneha Varghese

TEXAS STATE UNIVERSITY | CIS-QM Department

Table of Contents

1. Executive Summary:

This report encapsulates key findings of classifying sentiments of a bunch of tweets obtained from Twitter. The reason to choose this topic was to summarize the nature of reactions being posted on Twitter and utilizing the methodology described in this report; users with consistent record of posting negative tweets could be monitored closely so that hateful conduct or violence is not promoted. The primary components of the analysis include 1) calculation of sentimental labels 2) data preprocessing and cleansing 3) statistical model for classification 4) critical analysis & summary of findings.

2. Problem Statement:

We live in an era, where the internet is easily accessible. With the internet came a lot of social platforms that connect people albeit their physical distances. These social platforms did not just connect people but boosted the e-commerce sector. With the social platforms, adding values to people's lives it comes with some curses. Social platforms, be it Twitter or Instagram, have a lot of contents that can be labelled as derogatory, insensitive and malicious. This has eventually increased the rate of hate crimes drastically. It is imperative to guard social platforms against these crimes. Because of the volatility and the amount of data on these platforms, it is impossible to manage the contents on these social platforms manually.

Twitter is one of the most prominent social platforms. But at the same time there has been a lot of hate crimes occurring over twitter. The minimum age eligibility to hold a twitter account is thirteen years. The early teens may fall victims of mental stress due to the negative tweets posted on Twitter.

The goal is to build a model that can predict without any bias a tweet is negative or positive.

2.1 Data Description:

Data Source: Data is read into a dataframe using code mentioned in 7.2.1
https://www.kaggle.com/vkrahul/twitter-hate-speech

2.2 Data Dictionary:

Field Name	Description
id	Serial Number
tweet	The twitter tweets
Sentiment	Label of the tweet 1 – Negative tweet 0 – Positive tweet

The project analyzes a dataset from Kaggle, "Twitter hate speech" consisting of about 32k tweets. It consists of tweets twitted by users in the most popular social media platform twitter categorized as hate or not. We tried to think out of the box and generated our own labels for each tweet using Sentiment Analysis.

After sentiment analysis on these tweets, the new target labels are generated by calculating the positive and negative sentiments of each tweet and classifying them into labels 1 and 0 (ignoring the neutral sentiments). 1 and 0 represents negative and positive sentiment tweets respectively. The resulting judging dataset consists of about 23k records and 3 columns which is further considered to train the model. The columns include the serial number of the tweet, the content of the tweets and the label of the sentiment of the tweet under id, tweet and Sentiment respectively.

3. Exploratory data analysis and visualization:

After Sentiment Analysis, the target labels are generated, and the raw dataset is decomposed into judging dataset which is about 23k records and 3 columns. The label column consists of two numerical categories, 1 and 0, where 1 indicates negative sentiments and 0 indicates positive sentiments. Also, neither duplicate tweets nor missing values are present in the dataset.

However, the data is not balanced. The dataset was moderately biased with 59 % of the tweets or 13363 records containing positive sentiment labeled twitter data and 41% of the tweets or 9262 records containing negative sentiments labeled twitter data. The model will not be able to precisely learn which tweet is positive and which is negative, when an imbalanced dataset is used for training or modeling. This leads to falsely categorizing positive tweet as negative and vice-versa.

For instance, Figure 1.1 shows distribution of positive (label 0) and negative (label 1) tweets in the dataset. It can be seen that there is a clear imbalance in the distribution of labels and hence it is required to balance it for better prediction results.

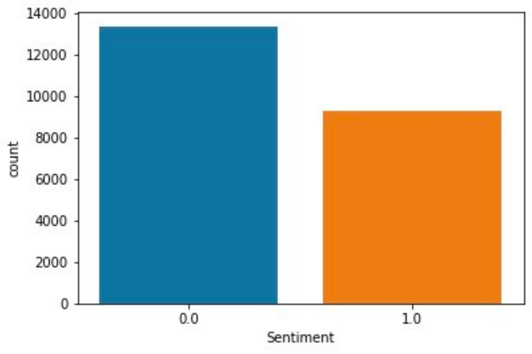


Figure 1.1: Plot of imbalanced dataset

Therefore, we perform random sampling. Here, we under sample Positive tweets as we cannot up-sample negative tweets. By taking a subset of negative and positive tweets from the actual dataset, we chose n random positive tweets, where n is the number of negative tweets and concatenate this with the subset of negative tweets. The resulting dataset is now used as the training data for the feature generation and modelling purposes. Figure 1.2 shows the balanced dataset after random sampling. Both the labels are equally distributed with each consisting of about 10k records. The code for sampling can be found in 7.2.5

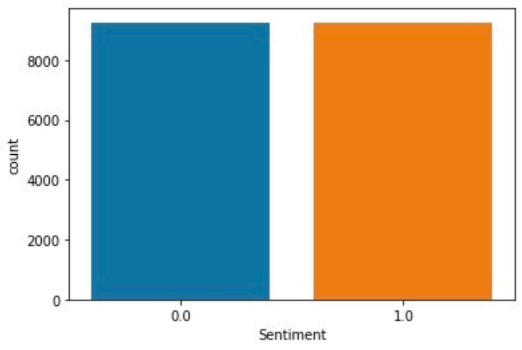


Figure 1.2: Plot of balanced dataset

Exploring and Visualizing data is an essential step in gaining insights. Hence, answering the questions regarding the data through visualization is one of the most appealing ways to solve a problem. Some of the predominant questions that needs to be answered are:

- What are the most frequently used words in the dataset?
- What are the topmost Positive and Negative words in the dataset?
- Do the hashtags throw the same sentiments as its parent tweet?

A Word Cloud is a visualization technique that can be used to visualize texts based on their frequency of use. This means that the most common words appear in large size and the less frequent words appears in smaller size. Figure 1.3 shows a word cloud with the most common words in the entire dataset. The code for word cloud can be found at 7.2.3

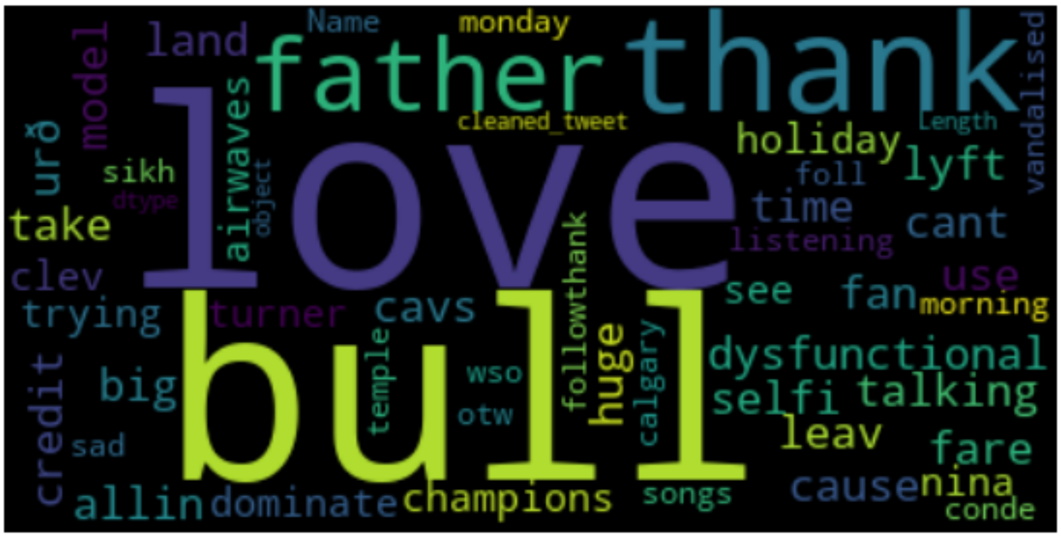


Figure 1.3: Top Words in the Twitter Hate Speech dataset

The presence of top words like love, bull, dominate, thank, dysfunctional, positive, etc. in the word cloud shows that the dataset contains a mixture of positive and negative words. In order to get a more detailed view, we plot the top positive and negative words in the dataset separately.

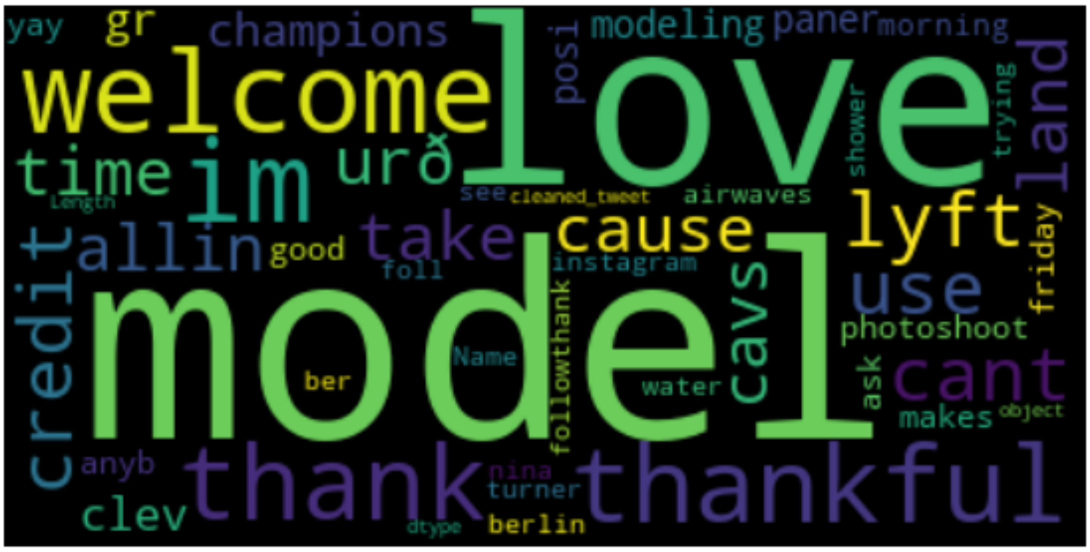


Figure 1.4: Top Positive Words in the Twitter Hate Speech dataset

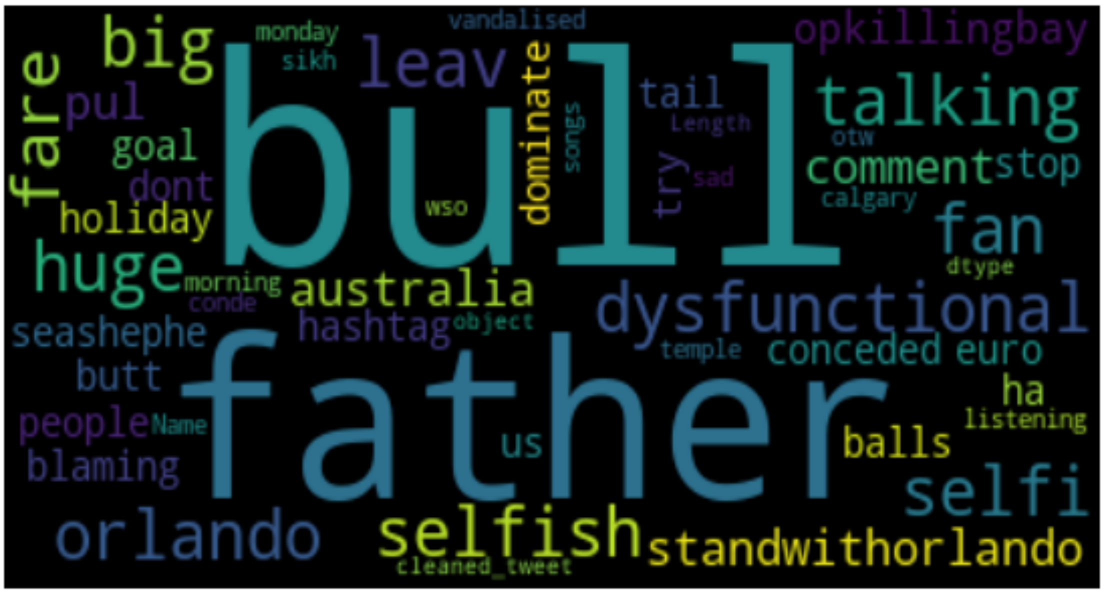


Figure 1.5: Top Negative Words in the Twitter Hate Speech dataset

The above Word Clouds are created to visualize the most frequently occurring words for two emotional labels in the tweets: Positive and Negative Sentiment tweets. In Figure 1.4, we can see that words like, love, positive, thankful, etc. shows the positive nature of the tweets and in Figure 1.5, words like bull, dominate, selfish, etc. shows the negative side of the tweets. In Figure 1.5, it can also be seen that words like father, Orlando, holiday, Australia are used in negative tweets, may be depicting some bad experience the user might have had.

The hashtags in twitter has the same meaning as the words in the tweet and hence represents the ongoing trend in Social media. Therefore, checking whether these hashtags add any value to the sentiment analysis is crucial. For instance, consider the following tweet:

i am thankful for sunshine. #thankful #positive

This is a positive tweet and the hashtags convey the same meaning.

Also, consider another example,

a transformer blew in my neighborhood. stuck w/o power, hopefully not for too long. i don't food in the fridge w/o coldness. #blackout

This is a negative sentiment tweet where #blackout emphasize a bad experience of the user. Therefore, visualizing the top hash words is very important to understand tweet sentiments.

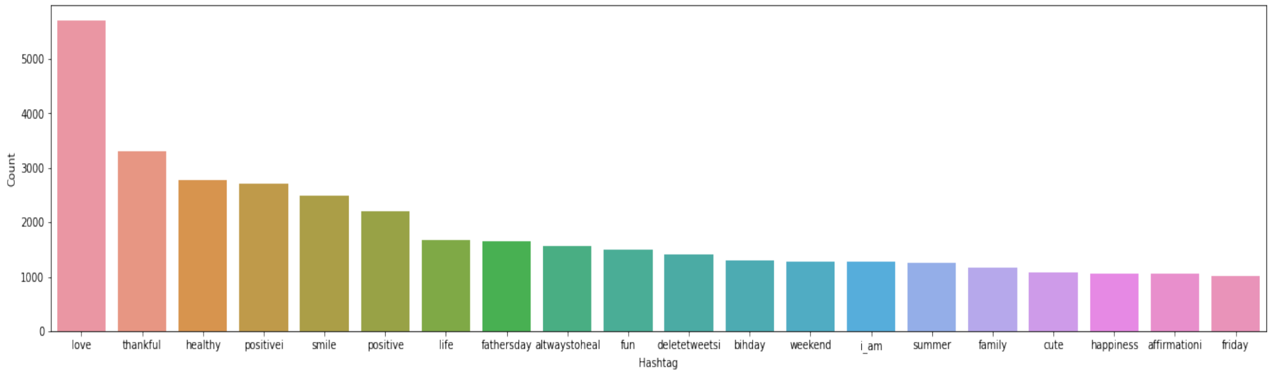


Figure 1.6: Top Positive Hashtags in the Twitter Hate Speech dataset

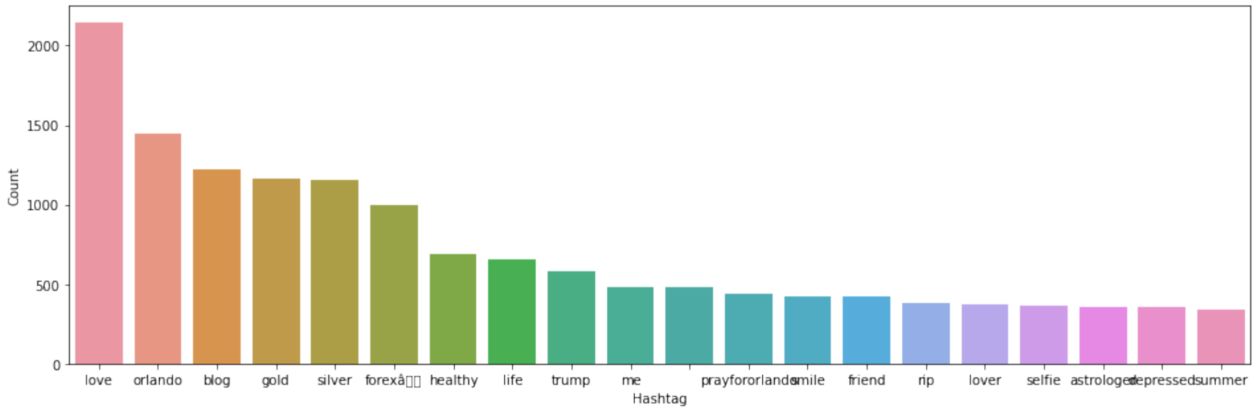


Figure 1.7: Top Negative Hashtags in the Twitter Hate Speech dataset

As we can see, in Figure 1.7, the negative hashtags have words like love, Orlando, gold, etc., most probably referencing to a bitter experience that the user might have had when he was expecting positive outcome.

The figure 1.6 shows the hashtags present in the positive sentiment labeled Tweets, where words like love, thankful, positive, etc. clearly depicts the it is part of a non-negative tweet. Hence, hashtags can also be used to understand the sentiments of a twitter tweet.

4. Approach:

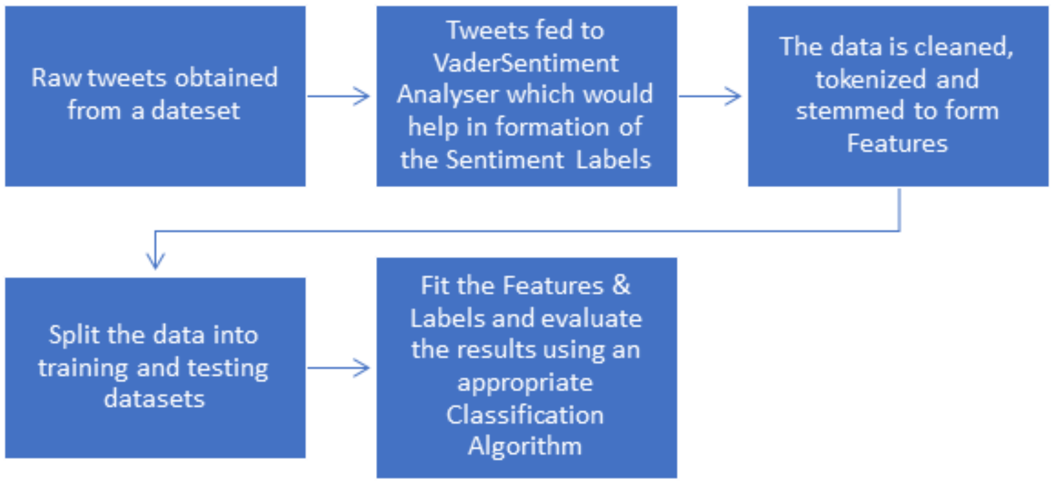

Raw tweets obtained from a dateset
Tweets fed to VaderSentiment Analyser which would help in formation of the Sentiment Labels
The data is cleaned, tokenized and stemmed to form Features
Split the data into training and testing datasets
Fit the Features & Labels and evaluate the results using an appropriate Classification Algorithm

Figure 1.8: Workflow of the approach

4.1 Creation of Labels

From the above Figure 1.8 represents the workflow of our Approach. The first step after we obtain a set of raw tweets is that we need to assign them with labels based on the overall sentiment associated with that tweet. For this purpose, we utilize Vader Sentiment Analyzer. It would help us in measuring the intensity or magnitude of a sentiment to which the tweet is specifically gravitated to. To illustrate this, let us assume that for a specific tweet, we have obtained values for the overall positivity, negativity and neutrality as 0.5, 0.2 and 0.3 respectively. The scoring mechanism works in such a way that the summation of all these values would be equal to 1. The sentiment analyzer would only indicate the intensity of sentiments by calculating these scores.

Upon this, we devise a logic to form labels based on these scores. Before we define a simple conditional logic for formulating this, we need to calculate the lower threshold values or cutoff points for determination of label values. For this, we calculate mean values of the overall sentiment scores. For instance, the mean value obtained for positive scores is 0.1322. Therefore, we build a conditional logic such that if the positive sentiment score is greater than or equal to 0.1322, then it would be assigned with label 0.

Likewise, we perform the same operation to form label 1. We obtain certain values which do not satisfy any of the above conditions. Essentially, they would be having a very strong neutral score. We drop such values from our dataset as the Vader Model itself has an ambiguity on judging the exact sentiment of that tweet as either positive or negative.

5. Analysis:

5.1. Data Cleaning

Data Cleaning is the first step of Analysis. It is imperative that we have a clean and consistent. The data is first verified for duplicates. We found no duplicate data points in the dataset. Then we check data for missing values. There were no missing values found. In order to have a uniform format of data, the data is converted into lowercase and ASCII characters. The characters that do not add value to the data is eliminated. Data cleaning code can be found in 7.2.2. The model is as good as the data that is fed into it. The steps followed in data cleaning are as follows in Figure 1.9:

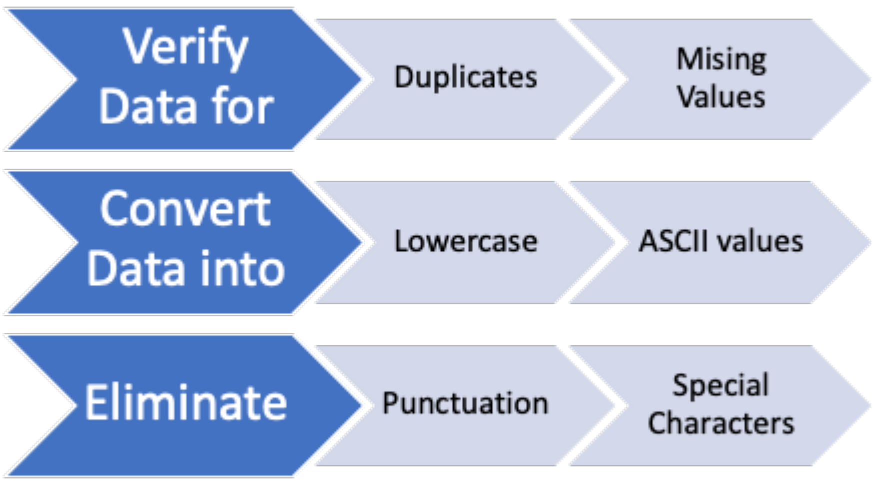


Figure 1.9: Data Cleaning Steps

5.2. Data Preprocessing & Feature Engineering

5.2.1. Data Preprocessing:

We now have obtained a dataset with the label column as a dependent variable and the tweets column as an independent variable. Our methodology to obtain processed features starts from cleaning up the tweets by removing special characters and converting the texts into lower case. We preprocess this data through three critical components of data manipulation pipeline, namely, Tokenization, Stemming and Stop Words removal as shown in the below Figure 1.10. The code for tokenization and stemming and removal of stop words can be found in 7.2.5

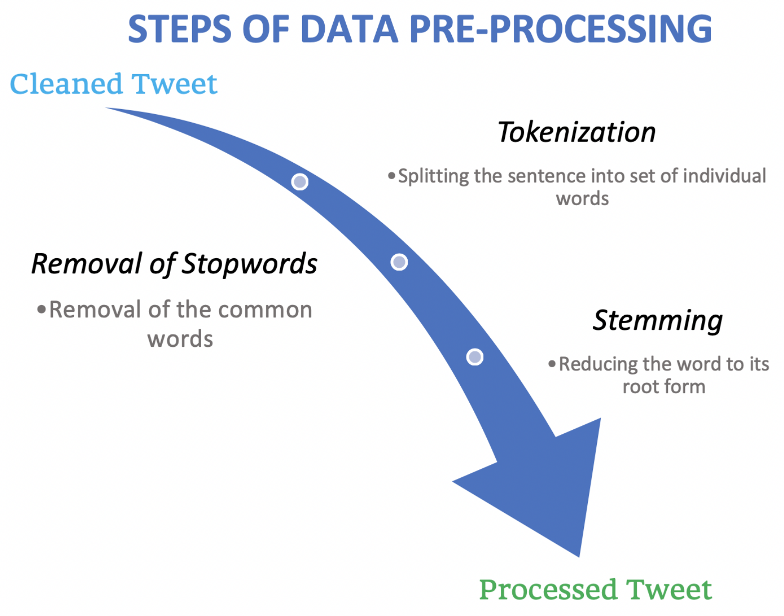

STEPS OF DATA PRE-PROCESSING
Cleaned Tweet
Tokenization
•Splitting the sentence into set of individual words
Removal of Stopwords
•Removal of the common words
Stemming
•Reducing the word to its root form
Processed Tweet

Fig: 1.10 Steps of Data Pre-processing

5.2.1.1. Tokenization

We have performed the tokenization process on the cleaned tweet data using tokenizer function as a first step for preparing our data to train the model.

For example, let us consider the following cleaned tweet,

INPUT

Cleaned_Tweet : when a father is dysfunctional and is so selfish he drags his kids into his dysfunction run

OUTPUT

Tokenized Words: [when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, run]

5.2.1.2.Removal of Stopwords

The tokenized words are further processed to remove the Stopwords and common words so that we can have on the important information from the tweet to train our model

INPUT

Tokenized_Words: [when, a, father, is, dysfunctional, and, is, so, selfish, he, drags, his, kids, into, his, dysfunction, run]

OUTPUT

Meaningful_Words : *[father, dysfunctional, selfish, drags, kids, dysfunction, run]*

5.2.1.3 Stemming

In stemming we used Lancaster stemmer for taking the stem(root word) of the meaningful words we have in order to reducing different forms of a word to a core root word for example, the words “help” and “helped” are just different tenses of the same verb so it would be cut short to its root word help.

INPUT

Meaningful_Words : *[user, father, dysfunctional, selfish, drags, kids, dysfunction, run]*

OUTPUT

Stemmed_Words : [fath, dysfunct, self, drag, kid, dysfunct, run]

As a final step of preprocessing we have combined the stemming step output to a single sentence as a processed tweet.

Processed : fath dysfunct self drag kid dysfunct run

5.2.2. **Feature Engineering**:

Further, the processed tweet is used for feature engineering where the tweet texts are converted into numeric forms using the Term frequency and Inverse Document Frequency (TF-IDF) vectorizer which indicates how relevant a term is in a given document with corresponding frequency values of the words. Feature Engineering code can be found in 7.2.6

TFIDF vectorizer algorithm focusses on TF frequency of a word with respect to a document(row) and the whole corpus. Likewise, the algorithm calculates frequency of a specific word and frequency of all the documents(rows) where that word occurs. Eventually, the texts are mapped into indices forming vectors. So as seen in the above steps, if we look at the tfidf features of the example tweet the output of tfidf vectorizer would be,

INPUT: *fath dysfunct self drag kid dysfunct run*

	fath	**dysfu nct**	**self**	**drag**	**kid**	**dysfu nct**	**run**	**hate**
0	0.190 438	0.790 007	0.339 848	0.321 566	0.229 332	0.790 007	0.246 934	0.0

From, above output we can see how the tfidf vectorizer is assigning frequency values of the words present in the tweet compared with the whole corpus. The hate word frequency value is zero as the word hate is not present in the tweet, we are vectorizing, similarly all the words that are not present in the tweet would be assigned a value of zero for the word. The top five features of the tweet that we have taken as example are,

feature	**tfidf**	
0	dysfunct	0.790007
1	self	0.339848
2	drag	0.321566
3	run	0.246934
4	kid	0.229332

So, the words with highest tfidf score are in the tweet we have taken as example can be seen above.

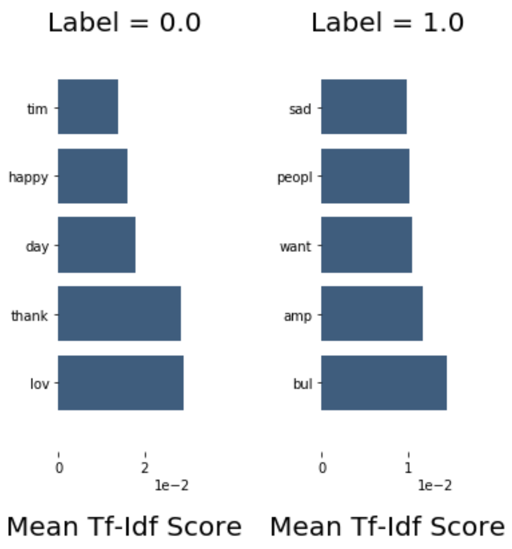


Fig: 1.11 Mean tf-idf score

Similarly, we TFIDF vectorizer calculates the frequency values for the whole processed tweets we have and we use this as the input for our model generation.

5.3. Model Generation

Our main objective here is to build a Text Classifier which could best guess or estimate the overall sentiment of the tweets from the historical data. The vectors generated from the tweets would have certain patterns within themselves, perhaps an indication of recurring words or phrases. We expect our statistical model to capture this underlying trend within the independent words provided with their TF-IDF Scores and determine if the tweet is hate or not-hate tweets. Also, as we are aware that our dependent variable has two values, we thereby utilize the Logistic Regression algorithm for classification of tweets.

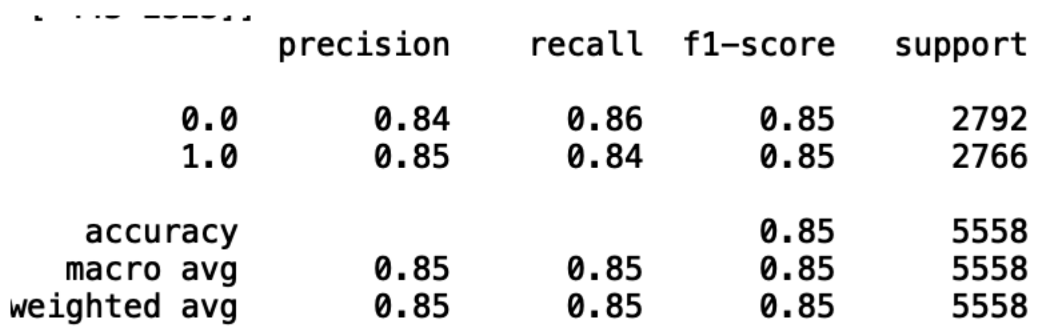
As our data is highly unbalanced, we used random sampling to balance our data. As Model greatly depends on the data fed into it.

We use sklearn package to build our logistic regression model. Logistic regression model is a statistical model, uses logistic function a S shaped curve with a function to predict the class of the independent variable. The independent variable is the bag of words we created with their TF-IDF values passed into the model as a vector. The dependent variable for our model is the Sentiment classification label we obtained by using Vader sentiment analysis.

We split our data into training and testing datasets in the ratio 70:30. The training dataset is used to train the Model. The model applies linear regression and a logistic function to determine whether a tweet is hateful or not- hate tweet. Code for model generation and building can be found at 7.2.8

5.4. Result Analysis:

The result analysis of the model built to predict the hate or no-hate tweet can be summarized in the table below. The accuracy, precision, recall and f1 score indicates that our model is a pretty good model. The model can predict whether a tweet is hateful of not hateful with 84% accuracy. Code for model Evaluation can be found at 7.2.9



```
              precision    recall  f1-score   support

         0.0       0.84      0.86      0.85      2792
         1.0       0.85      0.84      0.85      2766

    accuracy                           0.85      5558
   macro avg       0.85      0.85      0.85      5558
weighted avg       0.85      0.85      0.85      5558
```

The ROC curve for the model summarize the trade-off between the true positive rate and false positive rate for this predictive model and the Figure 1.13 indicates that we have a pretty good and balanced model.

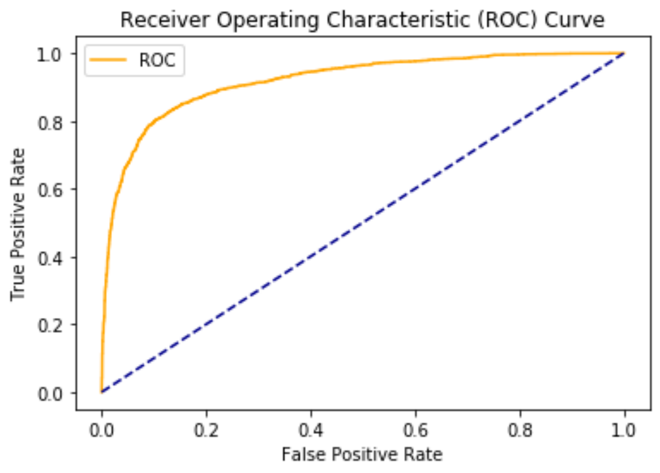


Fig: 1.13 ROC Curve

6. Summary

All in all, we have observed how different numerical metrics could be used to evaluate performance of a binary text classifier. We managed building an end to end pipeline for critical NLP processes, right from data extraction to model generation. To briefly summarize the proceedings, we have obtained a fairly accurate statistical model to predict the sentiment of the tweets.

7. Appendix:

7.1 Data Source: https://www.kaggle.com/vkrahul/twitter-hate-speech

Refered links,

- https://medium.com/natural-language-processing-machine-learning/nlp-for-beginners-how-simple-machine-learning-model-compete-with-the-complex-neural-network-on-b9f7f93c79e6
- https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
- https://medium.com/@annabiancajones/sentiment-analysis-on-reviews-feature-extraction-and-logistic-regression-43a29635cc81
- https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

7.2 Code:

7.2.1. Read data in:

```
import pandas as pd
df=pd.read_csv("/Users/snehavarghese/Downloads/twitter-hate-
speech/train_E6oV3lV.csv",sep=",",engine='python')
```

Sentiment Analysis using Vader

```
features = df.iloc[:,2].values
features
import statistics
import numpy as np
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
positive=list()
negative=list()
neutral=list()
main_list=list()
index_product=list()
dic={}
for i in range(0,len(features),1):
        score = analyser.polarity_scores(features[i])
        #df['Sentiment'] = np.where(score['neg']>0.25,
        'BlackList','Average/Good')
        positive.append(score['pos'])
        negative.append(score['neg'])
        neutral.append(score['neu'])
        #df['Sentiment'] = np.where(score['pos']>=50, 'Good')
        #print("Product:", df.iloc[i,1] , "Score:", score)
        main_list.append(score)
main_list
negative1=list()
positive1=list()
neutral1=list()
for i in range(0,len(main_list),1):
         negative1.append(main_list[i].get("neg"))
        positive1.append(main_list[i].get("pos"))
        neutral1.append(main_list[i].get("neu"))
print("Positive: ",statistics.mean(positive1))
print("Negative: ",statistics.mean(negative1))
print("Neutral: ",statistics.mean(neutral1))
index_list_positive=list()
for r in range(0,len(main_list),1):
         if main_list[r]['pos']>=0.1322:
                index_list_positive.append(r)
index_list_negative=list()
for x in range(0,len(main_list),1):
         if main_list[x]['neg']>=0.0538125586634128:
                index_list_negative.append(x)
for i in index_list_positive:
        df.loc[df.index[i], 'Sentiment'] = 0
for g in index_list_negative:
         df.loc[df.index[g], 'Sentiment'] = 1
df.isna().sum()
df = df.dropna()
df= df.reset_index(drop=True)
```

```
l1=list()
val=0
for x in range(0,len(df),1):
        if df['label'][x]==df['Sentiment'][x]:
                val=val+1
print("Vader Accuracy compared to existing labels: ",val/len(df))
val
len(df)
df.head()
```

7.2.2. Data Cleaning

```
  # check for duplicate values
df.drop_duplicates(inplace = True)

# check for missing values in tweet and label columns
print(df['tweet'].isna().sum())
print(df['label'].isna().sum())
#Data doesn't contain either duplicate values or missing values
df.head()
# remove @user
df['cleaned_tweet'] = df.tweet.apply(lambda x: ' '.join([word for word in x.split() if not
word.startswith('@')]))

# converting everything to lower-case
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x:''.join([x.lower() for word in
x.split()]))
import unidecode
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x : '
'.join([unidecode.unidecode(word) for word in x.split()]))

# Remove punctuation
import string
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x:''.join([i for i in x if i not in
string.punctuation]))
```

```
# removing the symbols
df["cleaned_tweet"] = df['cleaned_tweet'].str.replace('[?+-/]',' ')

# removing numbers
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x : ' '.join([tweet for tweet in x.split()
if not tweet.isdigit()]))

#Removing punctuations
df["cleaned_tweet"] = df ['cleaned_tweet'].str.replace('[~`!$%^&*_+-=|\:"]',")

# word-count
df['word_count'] =df['cleaned_tweet'].apply(lambda x: len(str(x).split(" ")))
```

7.2.3. WordCloud

```
# positive hashtags
import nltk
#Select all words from positive tweet
normal_words = ' '.join([word for word in df['cleaned_tweet'][df['Sentiment'] == 0]])
#Collect all hashtags
pos_htag = [htag for htag in normal_words.split() if htag.startswith('#')]
#Remove hashtag symbol (#)
pos_htag = [pos_htag[i][1:] for i in range(len(pos_htag))]
freq1 = pd.Series(' '.join(df['cleaned_tweet']).split()).value_counts()[:10]
freq1
import nltk
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
import spacy
spacy.load('en')
```

```
stop_words = stopwords.words('english')
Tweets_wordcloud =
WordCloud(stopwords=stop_words,max_font_size=100).generate(str(df.cleaned_tweet))
#Plotting word clouds for all words in tweets
plt.figure(figsize=(16,5))
plt.imshow(Tweets_wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

Neg_wordcloud=WordCloud(stopwords=stop_words,max_font_size=100).generate(str(df[df.
Sentiment==1].cleaned_tweet))
#Plotting word clouds for all words in hate tweets
plt.figure(figsize=(16,5))
plt.imshow(Neg_wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()


Pos_wordcloud=WordCloud(stopwords=stop_words,max_font_size=100).generate(str(df[df.
Sentiment==0].cleaned_tweet))
#Plotting word clouds for all words in good tweets
plt.figure(figsize=(16,5))
plt.imshow(Pos_wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

7.2.4. Tokenization and Stemming

```
from nltk.stem import LancasterStemmer
lancaster=LancasterStemmer()
stop_words = stopwords.words('english')
tokenizer = RegexpTokenizer("[\w]+")
ef identify_tokens(row):
    Cleaned_Tweet = row['cleaned_tweet']
    tokens = tokenizer.tokenize(Cleaned_Tweet)
    # taken only words (not punctuation)
    token_words = [w for w in tokens if w.isalpha()]
    return token_words
#Applying the function and output column is tokens
df['Tokenized_words'] = df.apply(identify_tokens, axis=1)

from nltk.corpus import stopwords
stops = set(stopwords.words("english"))
#Creating function
def remove_stops(row):
    my_list = row['Tokenized_words']
    meaningful_words = [w for w in my_list if not w in stops]
    return (meaningful_words)
#Applying the funtion and output column is stem_meaningful words in dataframe
df['Meaningful_Words'] = df.apply(remove_stops, axis=1)

from nltk.stem import LancasterStemmer
stemming=LancasterStemmer()
def stem_list(row):
    my_list = row['Meaningful_Words']
    stemmed_list = [stemming.stem(word) for word in my_list]
    return (stemmed_list)
#Applying the funtion and output column is stemmend words in dataframe
df['stemmed_words'] = df.apply(stem_list, axis=1)

def rejoin_words(row):
    my_list = row['stemmed_words']
    joined_words = ( " ".join(my_list))
    return joined_words
#Reforming the sentence with tokenized words
df['processed'] = df.apply(rejoin_words, axis=1)
```

7.2.5. Sampling to balance data

```
um_0 = len(df[df['Sentiment']==0])
num_1 = len(df[df['Sentiment']==1])
undersampled_data = pd.concat([ df[df['Sentiment']==0].sample(num_1) ,
df[df['Sentiment']==1] ])
print(len(undersampled_data))

#Creating model data for analysing
Model = undersampled_data[['processed','Sentiment']]
#Checking the balance of the data (Fairly balanced)
import seaborn as sns
sns.countplot(x='Sentiment', data=df)
```

7.2.6. Feature Engineering

```
#Tfidf Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
# initalising the vectoriser
tvec = TfidfVectorizer()
#Transforming features of whole data of processed tweets to understand the
#to understand the operation of TFIDF
TFIDF_Matrix = tvec.fit_transform(Model['processed'])
#vec = tvec.named_steps['vec']
features = tvec.get_feature_names()
Label = Model['Sentiment']

#Creating dataframe of TFIDF Sparse matrix
#From the frequency vectors created  for eacgh row of tweet
TFIDF_df = pd.DataFrame(TFIDF_Matrix.toarray(), columns=features)
TFIDF_df
TFIDF_df.columns
#Selecting few negative words from top positive words and seeing their sparse matrix
Positive = TFIDF_df[['love','thank','posit']]
Positive.head(16)
#Selecting few negative words from top negative words and seeing their sparse matrix
```

```
Negative = TFIDF_df[['father','want','sad','hate']]
Negative.tail(16)
#Defining a function to get top n tfidf values with their feature names
def top_tfidf_feats(row, features, top_n=5):
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df_x = pd.DataFrame(top_feats)
    df_x.columns = ['feature', 'tfidf']
    return df_x
#Defining a function to get top n tfidf values with their feature specific to each row
def top_feats_in_doc(TFIDF_Matrix, features, row_id, top_n=5):
    row = np.squeeze(TFIDF_Matrix[row_id].toarray())
    #get top n tfidf values with their feature names
    return top_tfidf_feats(row, features, top_n)

#Top five feature names for row 100
top_feats_in_doc(TFIDF_Matrix, features, row_id = 100, top_n=5)
#Defining a function to Return the top n features that on average
#are most important amongst rows identified by indices in grp_ids.
def top_mean_feats(TFIDF_Matrix, features, grp_ids=None, min_tfidf=0.1, top_n=5):

    if grp_ids:
        D = TFIDF_Matrix[grp_ids].toarray()
    else:
        D = TFIDF_Matrix.toarray()

    D[D < min_tfidf] = 0
    tfidf_means = np.mean(D, axis=0)
    return top_tfidf_feats(tfidf_means, features, top_n)

top_mean_feats(TFIDF_Matrix, features, grp_ids=None, min_tfidf=0.1, top_n=5)

#Defining a function to return a list of dfs,
#where each df holds top_n features and their mean tfidf value
```

```
#calculated across rows with the same class label.
def top_feats_by_class(TFIDF_Matrix, Label, features, min_tfidf=0.1, top_n=5):
    dfs = []
    labels = np.unique(Label)
    for label in labels:
        ids = np.where(Label==label)
        feats_df = top_mean_feats(TFIDF_Matrix, features, ids, min_tfidf=min_tfidf,
top_n=top_n)
        feats_df.label = label
        dfs.append(feats_df)
    return dfs

top_feats_by_class(TFIDF_Matrix, Label, features, min_tfidf=0.1, top_n=5)

#Plotting the data frames returned by the function plot_tfidf_classfeats
def plot_tfidf_classfeats_h(dfs):
    fig = plt.figure(figsize=(5, 5), facecolor="w")
    x = np.arange(len(dfs[0]))
    for i, DF in enumerate(dfs):
        ax = fig.add_subplot(1, len(dfs), i+1)
        ax.spines["top"].set_visible(False)
        ax.spines["right"].set_visible(False)
        ax.set_frame_on(False)
        ax.get_xaxis().tick_bottom()
        ax.get_yaxis().tick_left()
        ax.set_xlabel("Mean Tf-Idf Score", labelpad=30, fontsize=20)
        ax.set_title("Label = " + str(DF.label), fontsize=20)
        ax.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
        ax.barh(x, DF.tfidf, align='center', color='#3F5D7D')
        ax.set_yticks(x)
        ax.set_ylim([-1, x[-1]+1])
        yticks = ax.set_yticklabels(DF.feature)
        plt.subplots_adjust(bottom=0.09, right=0.97, left=0.15, top=0.95, wspace=1)
    plt.show()
```

```
plot_tfidf_classfeats_h(top_feats_by_class(TFIDF_Matrix, Label, features, min_tfidf=0.1,
top_n=5))
```

7.2.7. Split data into test and train data

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(Model['processed'], Model['Sentiment'],
train_size=int(Model.shape[0]*.70),random_state=1)
#Tfidf Vectorizer

# fit the data for vectorization
X_train_tvec = tvec.fit(X_train)
X_train_tvec = tvec.transform(X_train)
```

7.2.8. Build Model and train data

```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
log_reg = LogisticRegression(solver='lbfgs')
log_reg.fit(X_train_tvec,Y_train )
X_test_tvec = tvec.transform(X_test)
predictions = log_reg.predict(X_test_tvec)
```

7.2.9. Model Evaluation

```
rom sklearn.metrics import accuracy_score, precision_score, recall_score
print('Accuracy score: ', accuracy_score(Y_test, predictions))
print('Precision score: ', precision_score(Y_test, predictions))
print('Recall score: ', recall_score(Y_test, predictions))

#F1 Score
from sklearn.metrics import f1_score
f1_score(Y_test,predictions)

#Evaluation report of the model
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print(confusion_matrix(Y_test,predictions))
print(classification_report(Y_test,predictions))
print(accuracy_score(Y_test,predictions))

from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

def plot_roc_curve(fpr, tpr):
   plt.plot(fpr, tpr, color='orange', label='ROC')
   plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
   plt.xlabel('False Positive Rate')
   plt.ylabel('True Positive Rate')
   plt.title('Receiver Operating Characteristic (ROC) Curve')
   plt.legend()
   plt.show()

#Taking probabilities of only positive class
Positive_probs = probs[:, 1]

#Taking probabilities of only positive class
Negative_probs = probs[:, 0]
#Calculating AUC value for label =1(hate tweets)
pos_auc = roc_auc_score(Y_test, Positive_probs)
print('AUC:',pos_auc)

#Calculating AUC value for label =1(good tweets)
neg_auc = roc_auc_score(Y_test, Negative_probs)
print('AUC' ,neg_auc)

   #Considering false positive rate and true positive rates for ROC
   fpr, tpr, thresholds = roc_curve(Y_test, Positive_probs)

   #Plotting ROC
   plot_roc_curve(fpr, tpr)
```

```
plt.figure(figsize=(23,5))
ax = sns.barplot( x= ['Accuracy,'], y = "Count")
ax.set(ylabel = 'Count')
plt.show()
```