

Table of Contents

- ■ Importing Libraries
 - Importing the dataset
 - Distribution of Data - Target
 - Cleaning the data
 - Pre-process Text
 - Word Cloud
 - Split Data
- Vectorization
 - TF-IDF
 - CountVectorizer
- Word Embedding using Word2Vec
- Modeling
 - Inteprete Results
 - ○ Please see part 2 (deep learning) in the colab notebook on GitHub.

TWEETER SENTIMENT ANALYSIS | NLP

Part 1

Importing Libraries

In [2]:

```
# Import Sklearn libraries to build models
from sklearn.feature_extraction.text import TfidfVectorizer # TF-IDF to vect
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
# from sklearn.metrics import plot_confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
# Import Libraries to perform computation and do visualization.
import pandas as pd
import numpy as np
np.random.seed(0)
import seaborn as sns
```

```
import matplotlib.pyplot as plt
import string
import time
# Import nltk to check english lexicon.
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet, stopwords
from nltk import word_tokenize, FreqDist
from nltk import pos_tag # for Parts of Speech tagging
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
# Generate wordcloud for word distribution visualization.
from wordcloud import WordCloud

# Generating random numbers.
import random

from xgboost import XGBClassifier

# Transforms text to a fixed-length vector of integers.
import warnings
warnings.filterwarnings(action='ignore', category=UserWarning,
                        module='gensim')
import gensim
import os

import textdistance
#Efficient functions to search in strings.
import re as re

# Import images for world cloud.
from PIL import Image, ImageDraw, ImageFont

from yellowbrick.text import FreqDistVisualizer
from yellowbrick.datasets import load_hobbies

from tqdm.notebook import tqdm

from os import path
from os import environ
```

Importing the dataset

```
In [5]: # Importing the dataset
DATASET_COLUMNS = ["sentiment", "ids", "date", "flag", "user", "tweet"]
DATASET_ENCODING = "ISO-8859-1"
ta_df = pd.read_csv('training.1600000.processed.noemoticon.csv',
                     encoding=DATASET_ENCODING, names=DATASET_COLUMNS)

ta_df.head()
```

Out[5]:

	sentiment	ids	date	flag	user	tweet
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 [6]: # Sampling 30,000 tweets randomly
ta_df = ta_df.sample(30000, random_state=42)
```

```
In [7]: # Checking the size of the data
ta_df.shape
```

Out[7]: (30000, 6)

```
In [8]: # Looking into the type of data. Noting the data does not seem to have any missing values
ta_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30000 entries, 541200 to 1027373
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   sentiment    30000 non-null   int64  
 1   ids          30000 non-null   int64  
 2   date         30000 non-null   object  
 3   flag         30000 non-null   object  
 4   user         30000 non-null   object  
 5   tweet        30000 non-null   object  
dtypes: int64(2), object(4)
memory usage: 1.6+ MB
```

In [9]: `# Checking the percentage of null values
ta_df.isna().mean()*100`

Out[9]:

	sentiment	ids	date	flag	user	tweet
	0.0	0.0	0.0	0.0	0.0	0.0

dtype: float64

In [10]: `# Checking to see that 'flag' column only has one value 'NP_QUERY'
ta_df['flag'].unique()`

Out[10]: array(['NO_QUERY'], dtype=object)

In [11]: `# Dropping 'flag' and 'ids' columns
ta_df.drop(['flag','ids'], axis=1, inplace = True)`

In [12]: `# Checking that the changes took place
ta_df.head()`

Out[12]:

	sentiment	date	user	tweet
541200	0	Tue Jun 16 18:18:12 PDT 2009	LaLaLindsey0609	@chrishasboobs AHHH I HOPE YOUR OK!!!
750	0	Mon Apr 06 23:11:14 PDT 2009	sexygrneyes	@misstoriblack cool , i have no tweet apps fo...
766711	0	Tue Jun 23 13:40:11 PDT 2009	sammydearr	@TiannaChaos i know just family drama. its la...
285055	0	Mon Jun 01 10:26:07 PDT 2009	Lamb_Leanne	School email won't open and I have geography ...
705995	0	Sat Jun 20 12:56:51 PDT 2009	yogicerdito	upper airways problem

CLEANING THE DATA

```
In [13]: # Checking length
print('length of data is', len(ta_df))
```

length of data is 30000

Distribution of Data - Target

```
In [14]: # Check unique values in the sentiment column
ta_df['sentiment'].unique()
```

Out[14]: array([0, 4])

The value_counts method is used to count the number of occurrences of each unique value in the sentiment column, and the "normalize=True" argument is used to calculate the percentages instead of the counts.

```
In [16]: # Checking distribution
ta_df['sentiment'].value_counts()
```

Out[16]:

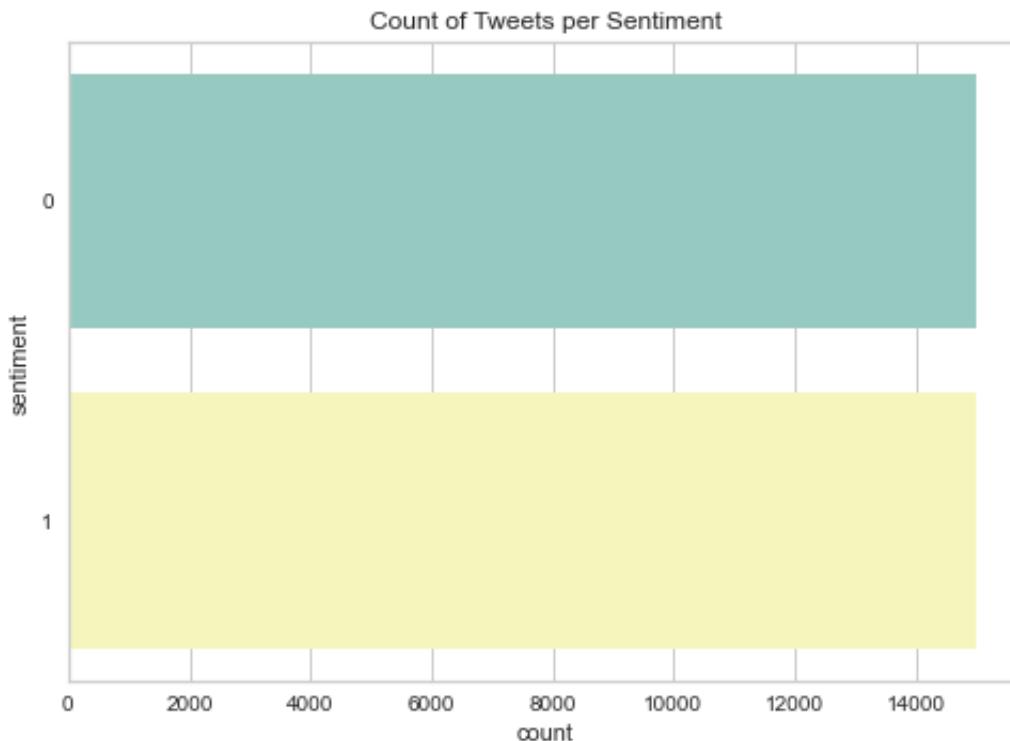
4	15001
0	14999
Name: sentiment, dtype: int64	

```
In [13]: # Replacing the value 4 -->1 for ease of understanding.
# 0 = negative, 1 = positive
ta_df['sentiment'] = ta_df['sentiment'].replace(4,1)
ta_df.head()
```

	sentiment	date	user	tweet
541200	0	Tue Jun 16 18:18:12 PDT 2009	LaLaLindsey0609	@chrishasboobs AHHH I HOPE YOUR OK!!!
750	0	Mon Apr 06 23:11:14 PDT 2009	sexygrneyes	@misstoriblack cool , i have no tweet apps fo...
766711	0	Tue Jun 23 13:40:11 PDT 2009	sammydearr	@TiannaChaos i know just family drama. its la...
285055	0	Mon Jun 01 10:26:07 PDT 2009	Lamb_Leanne	School email won't open and I have geography ...
705995	0	Sat Jun 20 12:56:51 PDT 2009	yogicerdito	upper airways problem

```
In [14]: # Plot the count plot for the target labels.  
  
# Setting p to plot of Emotion  
  
p = sns.countplot(data = ta_df, y = 'sentiment', palette="Set3")  
p.set(xlabel = 'count') #Labling X  
p.set(ylabel = 'sentiment') #Labling Y  
p.set(title = "Count of Tweets per Sentiment")
```

```
Out[14]: [Text(0.5, 1.0, 'Count of Tweets per Sentiment')]
```



```
In [15]: # Displays the count of columns and rows in the DataFrame.
```

```
print('Count of columns in the data is: ', len(ta_df.columns))  
print('Count of rows in the data is: ', len(ta_df))
```

```
Count of columns in the data is: 4  
Count of rows in the data is: 30000
```

Cleaning the data

- Checking and handling NaN values
- Drop duplicates

```
In [16]: # Dropping NaN values
```

```
ta_df.dropna(inplace=True)
```

```
In [17]: len(ta_df.duplicated(keep='last'))
```

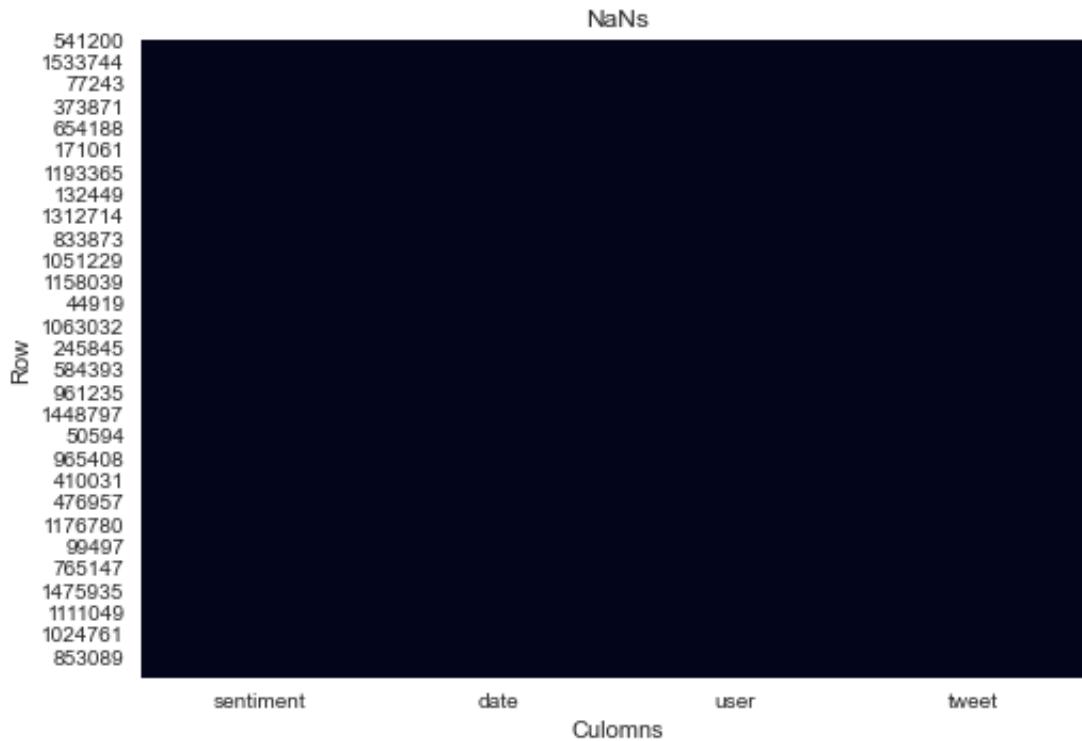
```
Out[17]: 30000
```

```
In [18]: # Dropping duplicates
ta_df.drop_duplicates()
ta_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30000 entries, 541200 to 1027373
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  --          -----          ----  
 0   sentiment   30000 non-null   int64  
 1   date        30000 non-null   object 
 2   user        30000 non-null   object 
 3   tweet       30000 non-null   object 
dtypes: int64(1), object(3)
memory usage: 1.1+ MB
```

```
In [19]: # Checking for Null values. We use the heatmap code which shows the contrast
```

```
sns.heatmap(ta_df.isnull(), cbar=False)
plt.title("NaNs")
plt.xlabel('Culomns')
plt.ylabel('Row')
plt.show()
```



Now that our data is clean and without duplicates, we can proceed to preprocess the tweets for our machine learning models.

```
In [20]: # Creating an independent copy  
ta_df_copy = ta_df.copy()
```

Pre-process Text

We will use text processing to allow the data to be more digestible for model use later in this project. This is an integral step in Natural Language Processing (NLP).

The Preprocessing steps taken are:

- Converting to lower case letters: Each text will be transferred to a lower case letter.
- Removing stop words: Remove common words like "a", "an", "the", etc., which don't carry much meaning and can be safely removed without affecting the model's performance.
- Removing Words with 2 letters: Words with length less than 2 are removed.
- Replacing http with space: Links starting with "http" or "https" or "www" are replaced by " ".
- Stemming to reduce words to their root form to remove any variations of the same word.

```
In [21]: # Defining a preprocessing function for cleaning and preprocessing text data.

tqdm.pandas() # Adding the progress bar
ps = PorterStemmer() # Defining stemming words with ps

# Removes URLs and non-alphanumeric
TEXT_CLEANING_RE = "@\s+|https?:\s+|http?:\s|[^A-Za-z0-9]+"

# Replace 3 or more consecutive letters by 2 letter.
sequencePattern = r"(.)\1\1+"
seqReplacePattern = r"\1\1"

stop_words = set(stopwords.words("english")) # Creating a set of English stop

# Defining the preprocessing function

def preprocess(text,apply_stem=True):

    # Remove link,user and special characters
    text = re.sub(TEXT_CLEANING_RE, ' ', str(text).lower()).strip()
    text = re.sub(sequencePattern, seqReplacePattern, text)

    tokens = []
    for token in text.split():
        if token not in stop_words: # Checking if the word is not a stopword
            if apply_stem:
                # Stemming to the word using the PorterStemmer if True
                tokens.append(ps.stem(token))
            else:
                # Adding the original word if stemming is not applied (for words like 'ain't)
                tokens.append(token)
    # Joining and returning the list of tokenized words into a single string
    return " ".join(tokens)
```

```
In [22]: # Creating a new column apply a function to each row

ta_df['clean_tweet'] = ta_df['tweet'].progress_apply(lambda x: preprocess(x,True))
```

```
In [23]: ta_df['clean_tweet'].head()
```

```
Out[23]: 541200                               ah hope ok
750          cool tweet app razr 2
766711      know famili drama lame hey next time u hang ki...
285055      school email open geographi stuff revis stupid...
705995          upper airway problem
Name: clean_tweet, dtype: object
```

```
In [24]: # Creating a new column apply a function to each row
```

```
ta_df['clean_tweet_wt_stem'] = ta_df['tweet'].progress_apply(lambda x:preproc
```

```
In [25]: # ta_df.drop('clean_tweets',axis=1,inplace=True)
```

```
In [26]: # Checking that the change has taken place.  
ta_df.head()
```

Out[26]:

		sentiment	date	user	tweet	clean_tweet	clean_tweet_wt_s
541200	0		Tue Jun 16 PDT 2009	LaLaLindsey0609	@chrishasboobs AHHH I HOPE YOUR OK!!!	ahh hope ok	ahh hop
750	0		Mon Apr 06 PDT 2009	sexygrneyes	@misstoriblack cool , i have no tweet apps fo...	cool tweet app razr 2	cool tweet apps r
766711	0		Tue Jun 23 PDT 2009	sammydearr	@TiannaChaos i know just family drama. its la...	know famili drama lame hey next time u hang ki...	know family dr lame hey next tir hang
285055	0		Mon Jun 01 PDT 2009	Lamb_Leanne	School email won't open and I have geography ...	school email open geographi stuff revis stupid...	school email c geography stuff re stu
705995	0		Sat Jun 20 PDT 2009	yogicerdito	upper airways problem	upper airway problem	upper airways prot

EXPLORATORY DATA ANALYSIS

```
In [27]: ta_df['clean_tweet_wt_stem']
```

```
Out[27]: 541200                                ah hope ok
          750                                     cool tweet apps razr 2
          766711      know family drama lame hey next time u hang ki...
          285055      school email open geography stuff revise stupi...
          705995                               upper airways problem
          ...
          263962      bout go eat eat sushi phoboy86 house hopefully...
          1540392           guess bus ride tour bus good job
          1549636           hmm might go bed
          706868      sounds suspiciously like year million dreams h...
          1027373           problem
Name: clean_tweet_wt_stem, Length: 30000, dtype: object
```

Word Cloud

We are creating a wordcloud with masking which allows us to visualize words in a specific shape of thumbs up and down.

```
In [28]: def red_color_func(word, font_size, position, orientation, random_state=None,
                      return tuple(Reds_9.colors[random.randint(6,8)])
```

```
In [29]: # Function to help us generate wordcloud
def red_color_func(word, font_size, position, orientation, random_state=None,
                    return tuple(Reds_9.colors[random.randint(6,8)])
```

```
In [30]: # Storing clean tweets
tweets = ta_df['clean_tweet_wt_stem']
```

With these Series objects containing the preprocessed text, you can perform further analysis or visualization on the negative and positive tweets separately. We will be able to create separate word clouds for each sentiment group or calculate the most frequent words in each group.

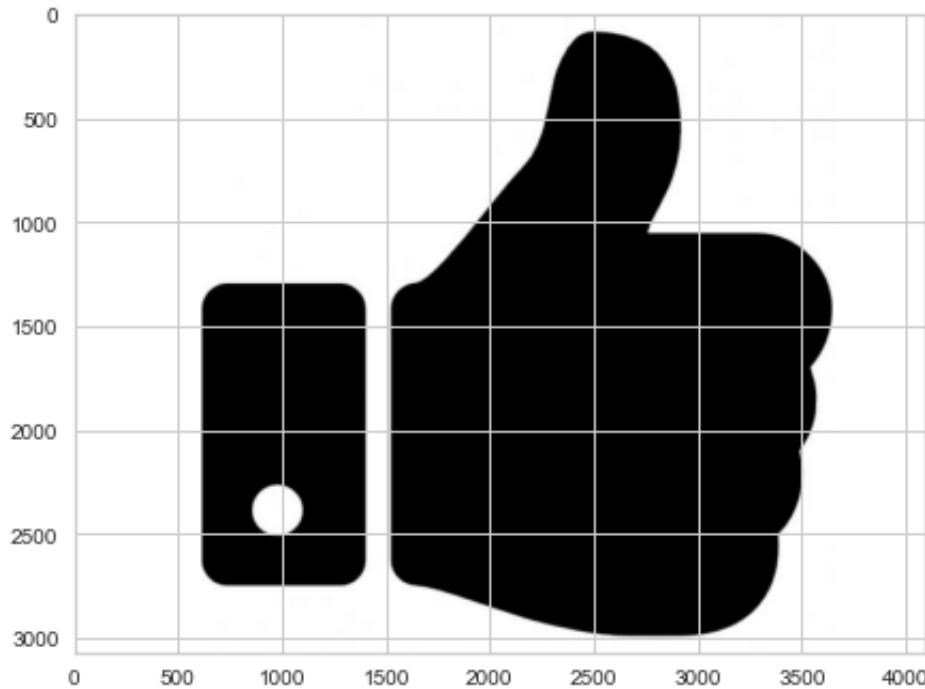
```
In [31]: # Storing the rows with a sentiment value of 0, which represents negative sen
tweet_negative = ta_df[ta_df['sentiment']==0]

# Storing the rows with a sentiment value of 1, which represents positive sen
tweet_positive = ta_df[ta_df['sentiment']==1]
```

```
In [32]: # Storing the preprocessed text of the tweets with negative sentiment
tweet_negative = tweet_negative['clean_tweet_wt_stem']
# Storing the preprocessed text of the tweets with positive sentiment
tweet_positive = tweet_positive['clean_tweet_wt_stem']
```

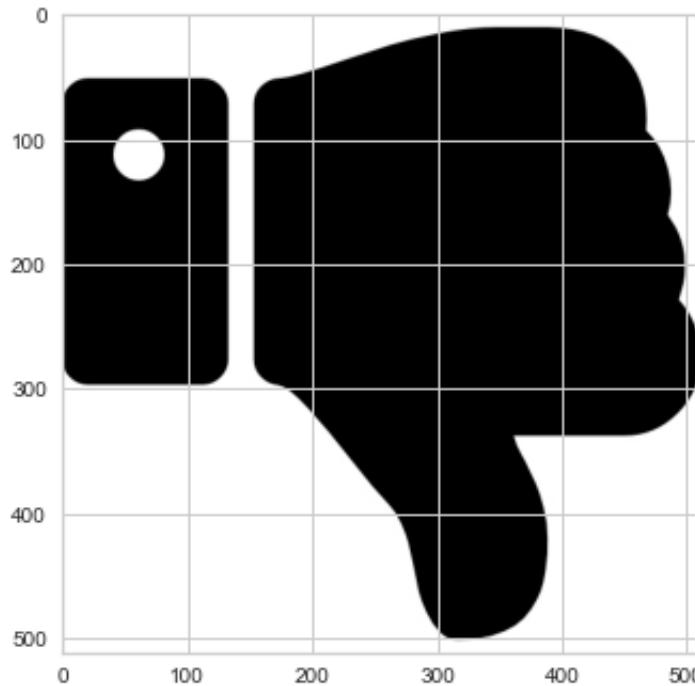
```
In [33]: # Resizing it to a 20x20 size using the Image.ANTIALIAS filter
img = Image.open('upvote.png')
img.resize((20, 20), Image.ANTIALIAS)
plt.imshow(img) # show output
```

```
Out[33]: <matplotlib.image.AxesImage at 0x7fb256aff280>
```



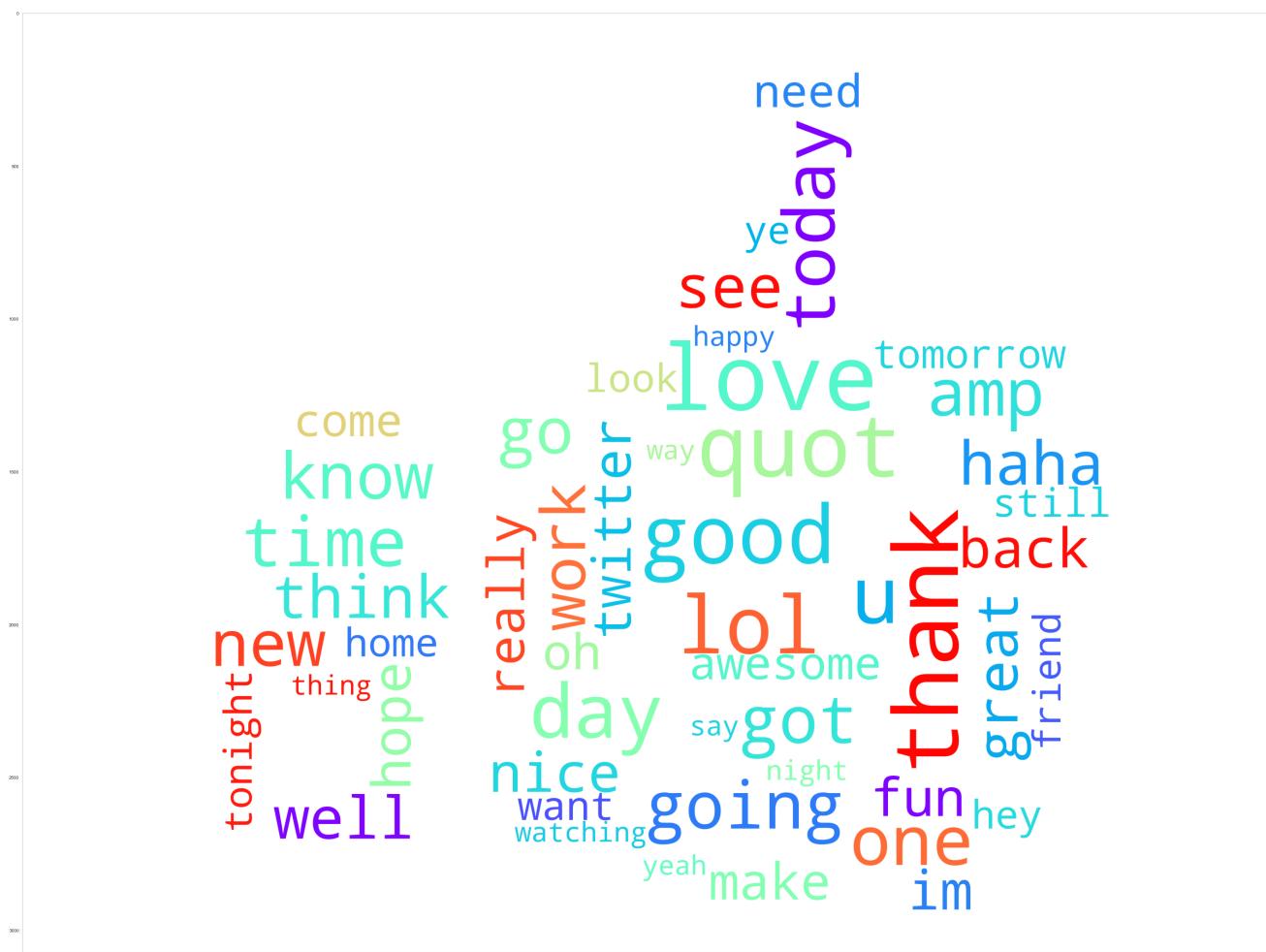
```
In [34]: # Resizing it to a 20x20 size using the Image.ANTIALIAS filter  
mask_d = Image.open('final.png')  
mask_d.resize((20, 20), Image.ANTIALIAS)  
plt.imshow(mask_d) # show output
```

```
Out[34]: <matplotlib.image.AxesImage at 0x7fb255a0f2e0>
```

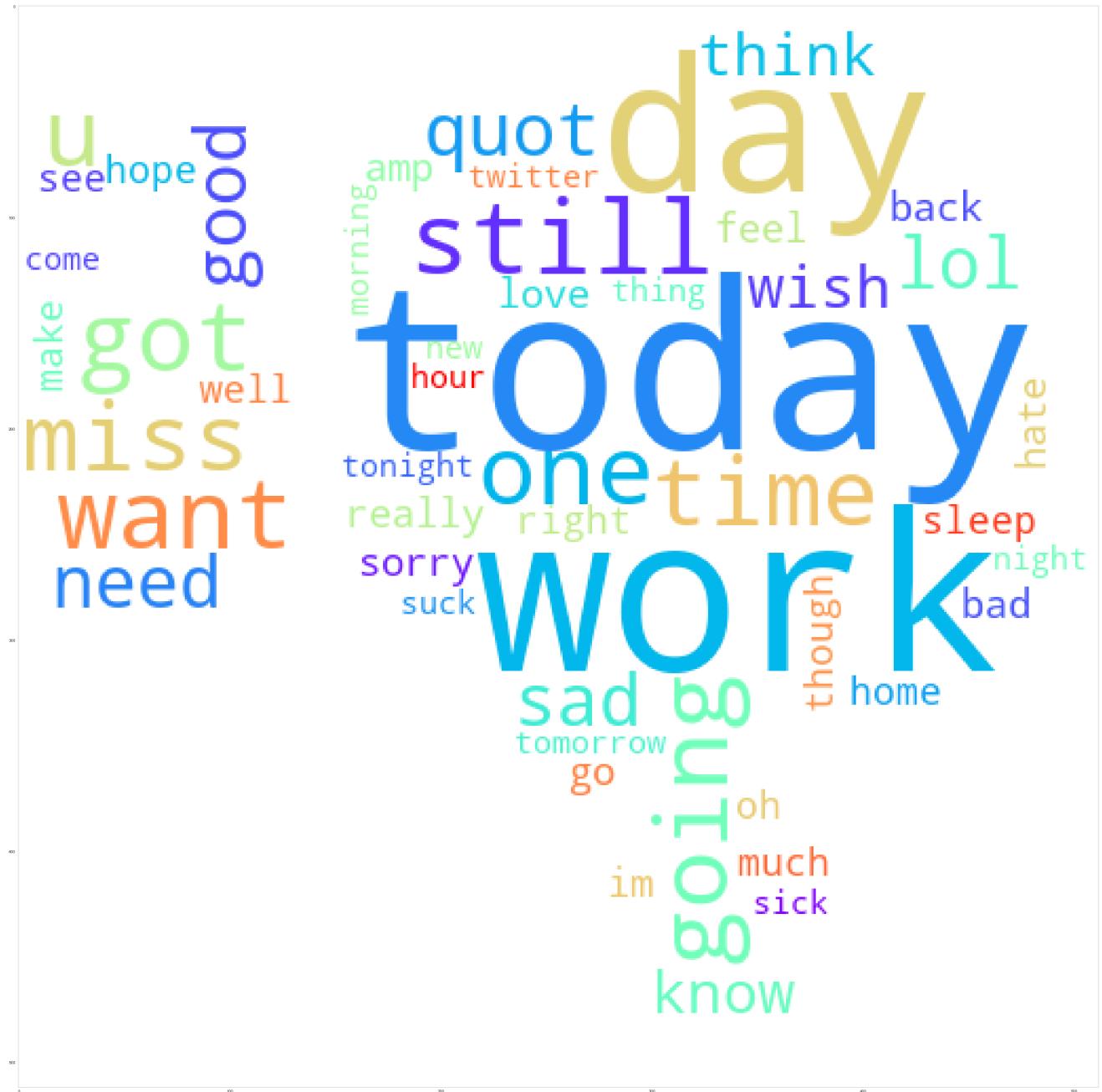


```
In [35]: def wc(data, mask, fname):  
  
    plt.figure(figsize = (50,50))  
  
    wc = WordCloud(width = 300, height = 200,background_color='white',  
                   max_words = 50, max_font_size =300,  
                   colormap='rainbow',mask=mask)  
  
    wc.generate(' '.join(data))  
  
    plt.imshow(wc)  
    plt.grid(visible=False)  
    plt.savefig(f'{fname}.png',dpi=200)  
#    plt.axis('off')
```

```
In [36]: mask_pos = np.array(Image.open('upvote.png'))  
wc(tweet_positive, mask_pos, 'positive')
```



```
In [37]: mask_neg = np.array(Image.open('final.png'))
wc(tweet negative, mask neg, 'negative')
```



```
In [38]: pos_df = ta_df['clean_tweet'].loc[ta_df['sentiment'] == 1].progress_apply(lambda
```

```
In [39]: neg_df = ta_df['clean_tweet'].loc[ta_df['sentiment'] == 0].progress_apply(lambda
```

```
In [40]: tokens = [] #Instantiating total token list
tokens_pos = [] #Instantiating positive token list
tokens_neg = [] #Instantiating negative token list

for row in pos_df:
    tokens.extend(row) #Populating token list from dataframe
for row in pos_df:
    tokens_pos.extend(row) #Populating token list from dataframe
for row in neg_df:
    tokens_neg.extend(row) #Populating token list from dataframe

print(f'Total Corpus Tokens: {len(tokens)}')
# Print total number of tokens
print(f'Number of Positive Tokens: {len(tokens_pos)}')
# Print number of positive tokens
print(f'Number of Negative Tokens: {len(tokens_neg)}')
# Print number of negative tokens
```

```
Total Corpus Tokens: 106790
Number of Positive Tokens: 106790
Number of Negative Tokens: 109861
```

The function creates two bar plots. The first plot shows the top 10 most frequent positive n-grams, and the second plot shows the top 10 most frequent negative n-grams.

This will create and display the bar plots for unigrams (1-grams) of positive and negative tokens. You can change the input integer to display n-grams of different sizes: bigrams (2-grams) or trigrams (3-grams).

In [42]: # Defining function

```
def make_ngram(i, tokens_pos = tokens_pos, tokens_neg = tokens_neg):
    #Setting up positive ngram
    n_gram_pos = (pd.Series(nltk.ngrams(tokens_pos, i)).value_counts()[:10])
    #Setting up negative ngram
    n_gram_neg = (pd.Series(nltk.ngrams(tokens_neg, i)).value_counts()[:10])

    n_gram_df_pos = pd.DataFrame(n_gram_pos) #Creating positive ngram datafra
    n_gram_df_neg = pd.DataFrame(n_gram_neg) #Creating negative ngram datafra

    n_gram_df_pos = n_gram_df_pos.reset_index() #Resetting index
    n_gram_df_neg = n_gram_df_neg.reset_index() #Resetting index

    #Renaming positive plot
    n_gram_df_pos = n_gram_df_pos.rename(columns = {'index': 'Phrase', 0: 'Co'
    #Renaming negative plot
    n_gram_df_neg = n_gram_df_neg.rename(columns = {'index': 'Phrase', 0: 'Co

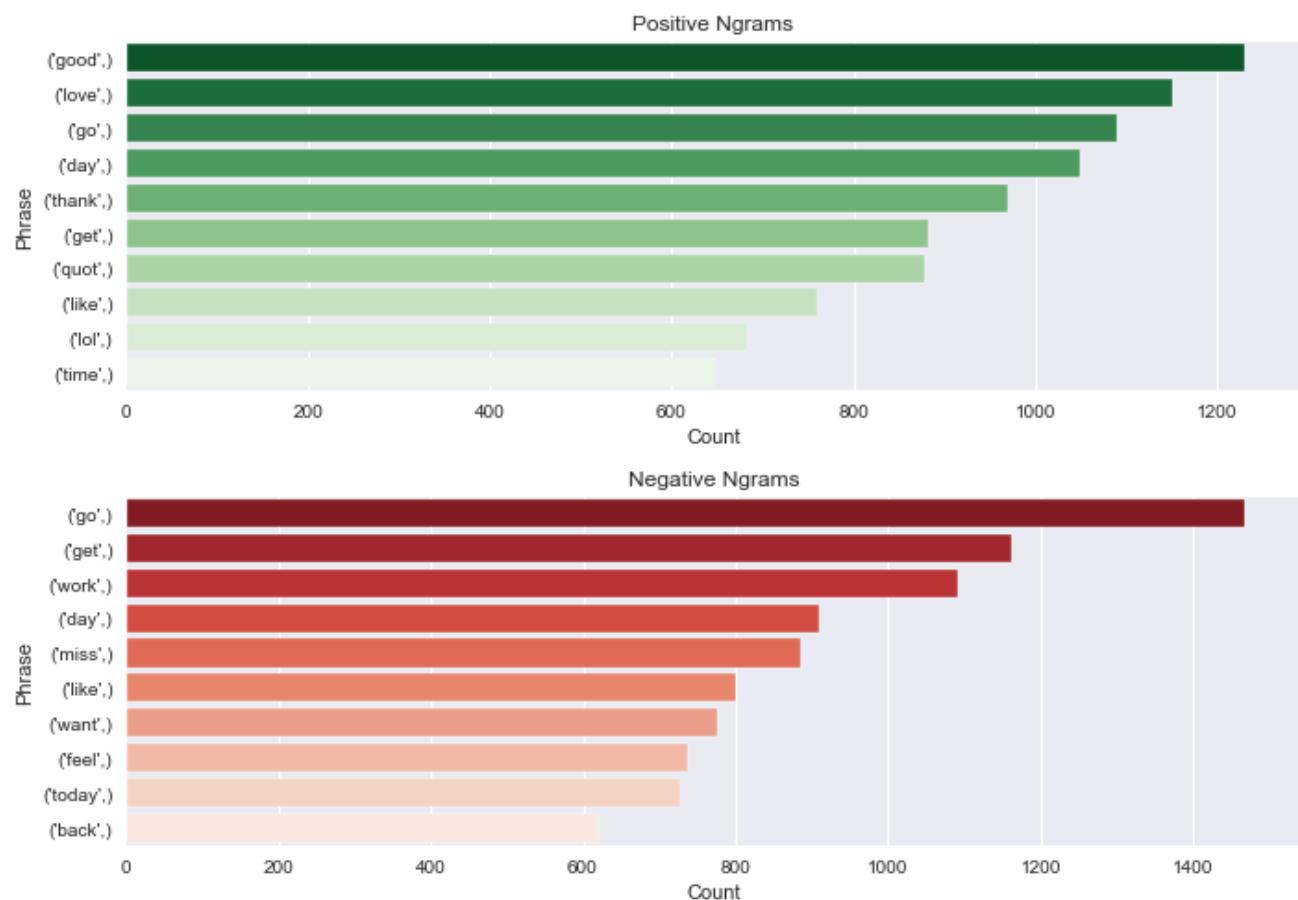
    with sns.axes_style('darkgrid'): #Setting seaborn to darkgrid style

        fig = plt.figure(figsize = (10, 10)) #Setting figsize
        ax1 = fig.add_subplot(311) #Stacking first figure
        ax2 = fig.add_subplot(312) #Stacking second figure

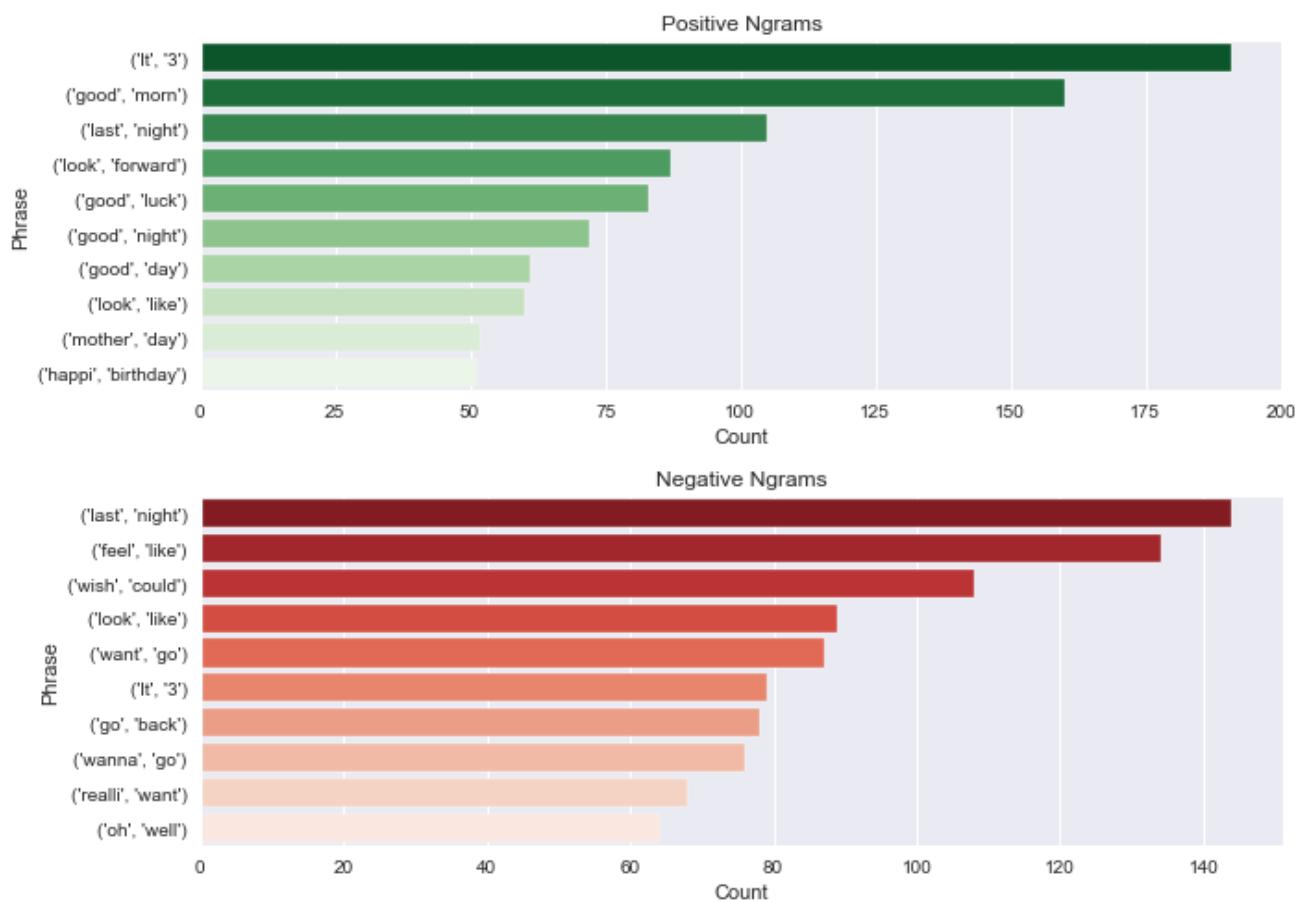
        sns.barplot(ax = ax1, x = 'Count', y = 'Phrase', data = n_gram_df_pos
                    # Assigning barplot to positive ngrams
                    palette = 'Greens_r').set(title = 'Positive Ngrams')
        sns.barplot(ax = ax2, x = 'Count', y = 'Phrase', data = n_gram_df_neg
                    # Assigning barplot to negative ngrams
                    palette = 'Reds_r').set(title = 'Negative Ngrams')

        plt.tight_layout() #Make plot layouts tight

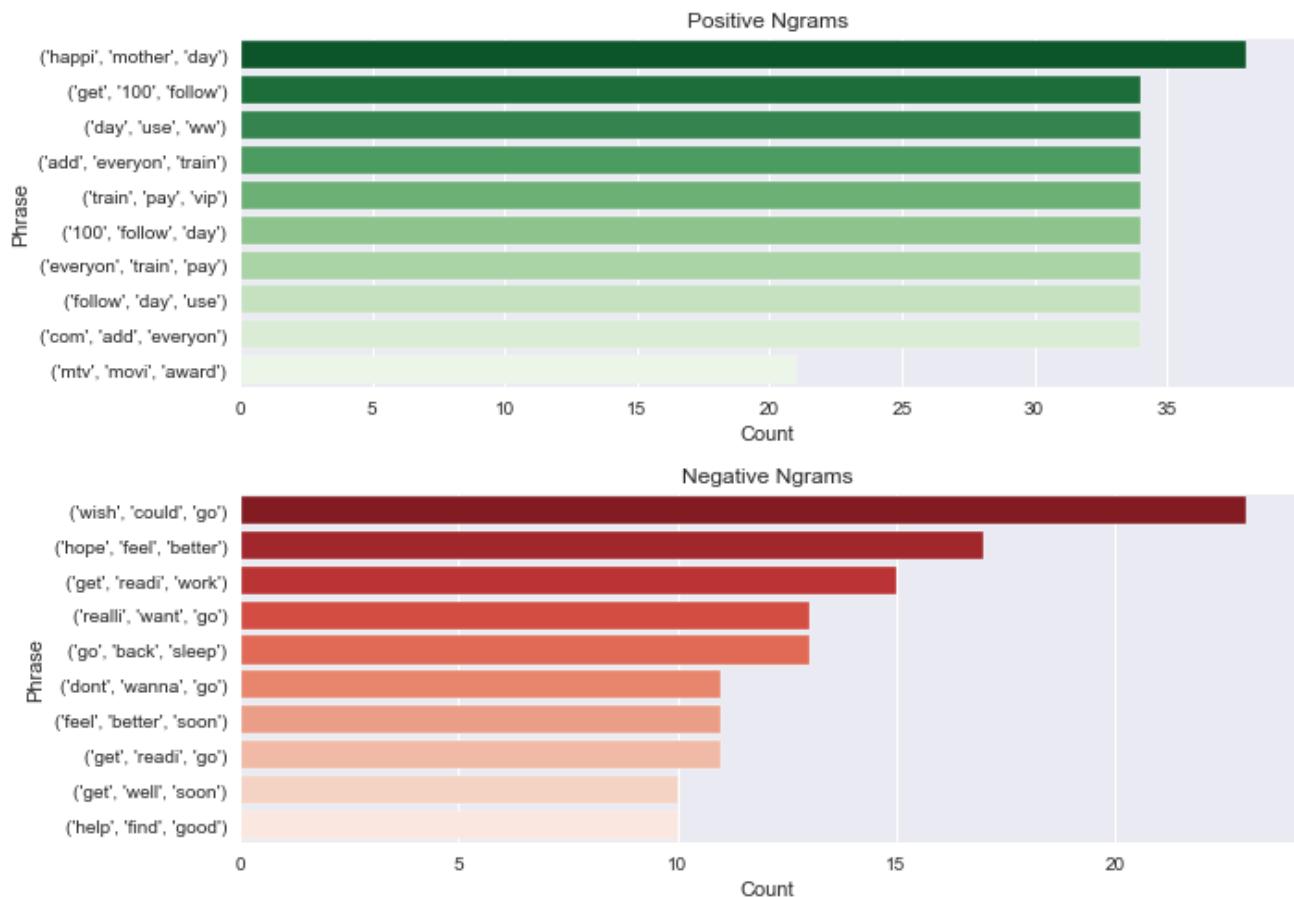
make_ngram(1) #Plot 1 word ngrams
```



```
In [43]: make_ngram(2) #Plot 2 word ngrams
```



```
In [44]: make_ngram(3) #Plot 3 word ngrams
```



MODELING

Split Data

```
In [45]: # Splitting data target and clean tweets
x = ta_df['clean_tweet'] # 1,000,000
y = ta_df['sentiment'] # 1,000,000
```

```
In [46]: # List of unique sentiment values
list(set(ta_df['sentiment']))
```

```
Out[46]: [0, 1]
```

```
In [47]: # View independent
y.head()
```

```
Out[47]: 541200      0
         750       0
        766711      0
        285055      0
       705995      0
Name: sentiment, dtype: int64
```

```
In [48]: # Count values
ta_df['sentiment'].value_counts()
```

```
Out[48]: 1      15001
0      14999
Name: sentiment, dtype: int64
```

```
In [49]: # Splitting your dataset into training and testing sets.
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.25,
                                                    random_state = 43)
```

```
In [50]: # Checking shape of train and test data
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape , y_test.shape)
```

```
(22500,) (22500,)
(7500,) (7500,)
```

Vectorization

We will utilize three different vectorization approaches: TF-IDF, Vector Count, and Word2Vec. We will examine which vector approach has worked best among the machine learning models.

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that reflects the importance of a word to a document in a corpus. It assigns more weight to words that are less frequent in the entire corpus and less weight to words that are more common. The calculation involves both: Term Frequency (TF) and Inverse Document Frequency (IDF).

IDF: $IDF = \log(\text{count of documents}) / (\text{count of documents containing the word})$

TF: $TF = (\text{count of the frequency of a word in a document}) / (\text{count of words in a document})$
<https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/>

```
In [75]: # Initializing TFIDF
vectoriser_tfidf = TfidfVectorizer()
```

```
In [76]: # Fit the data  
vectoriser_tfidf.fit(X_train)  
  
print('Number of feature words:', len(vectoriser_tfidf.get_feature_names_out())  
Number of feature words: 18078  
  
In [77]: # Converting the tweets in train and test data into TFIDF features  
X_train_tfidf = vectoriser_tfidf.transform(X_train)  
X_test_tfidf = vectoriser_tfidf.transform(X_test)
```

CountVectorizer

Count vectorization is a simple technique to vectorize words by counting their frequency in a document. It creates a matrix in which each word is represented by its count in the corpus.

```
In [78]: # Initializing countvector  
vectorizer_vc = CountVectorizer()  
# Fitting countvector  
vectorizer_vc.fit(X_train)  
  
Out[78]: ▾ CountVectorizer  
CountVectorizer()
```

```
In [79]: # Converting tweets in train and test data into countvector features  
  
X_train_vc = vectorizer_vc.transform(X_train)  
X_test_vc = vectorizer_vc.transform(X_test)
```

Word Embedding using Word2Vec

Word2Vec is a neural network-based technique that learns a vector representations of words from large amounts of text data. It represents each word as a vector in a multi dimensional space - and the distance between vectors(words) reflects the semantic similarity between words.

```
In [58]: # Initializing corpus using funciton progress_apply to go though each row and  
corpus = X.progress_apply(lambda line:line.split())  
  
In [59]: # View corpus  
corpus
```

```
Out[59]: 541200 [ahh, hope, ok]
750 [cool, tweet, app, razr, 2]
766711 [know, famili, drama, lame, hey, next, time, u...
285055 [school, email, open, geographi, stuff, revis, ...
705995 [upper, airway, problem]
...
263962 [bout, go, eat, eat, sushi, phoboy86, hous, ho...
1540392 [guess, bu, ride, tour, bu, good, job]
1549636 [hmm, might, go, bed]
706868 [sound, suspici, like, year, million, dream, h...
1027373 [problem]
Name: clean_tweet, Length: 30000, dtype: object
```

```
In [60]: # Train Word2Vec Model on corpus
import warnings

warnings.filterwarnings(action = 'ignore')

model = Word2Vec(corpus,
                  min_count=1,           # specifying the minimum frequency cou...
                  vector_size=200,        # specifying the dimensionality of the ...
                  workers=os.cpu_count(), # Number of processors
                  sg=1                   # 1 for skip-gram
)
```

```
In [62]: # Creating a new column of tokenized tweets
ta_df['tokens'] = ta_df['clean_tweet'].progress_apply(lambda line:line.split(
```

```
In [63]: # Checking that change took place
ta_df.head()
```

Out[63]:

		sentiment	date	user	tweet	clean_tweet	clean_tweet_wt_s
			Tue Jun 16 0 PDT 2009		@chrishasboobs AHHH I HOPE YOUR OK!!!	ahh hope ok	ahh hop
541200	0	18:18:12	LaLaLindsey0609				
750	0	23:11:14	sexygrneyes	@misstoriblack cool , i have no tweet apps fo...	cool tweet app razr 2	cool tweet apps re	
766711	0	13:40:11	sammydearr	@TiannaChaos i know just family drama. its la...	know famili drama lame hey next time u hang ki...	know family dr lame hey next tir hang	
285055	0	10:26:07	Lamb_Leanne	School email won't open and I have geography ...	school email open geographi stuff revis stupid...	school email c geography stuff re stu	
705995	0	12:56:51	yogicerdito	upper airways problem	upper airway problem	upper airways prot	
		Sat Jun 20 2009					

In [64]:

```
# Filtering out any rows where the tokens column is an empty list.
ta_df = ta_df[ta_df['tokens'].apply(lambda line: True if len(line) else False)]
```

In [66]:

```
# View tokens
ta_df['tokens']
```

Out[66]:

```
541200          [ahh, hope, ok]
750            [cool, tweet, app, razr, 2]
766711        [know, famili, drama, lame, hey, next, time, u...
285055        [school, email, open, geographi, stuff, revis, ...
705995          [upper, airway, problem]
...
263962        [bout, go, eat, eat, sushi, phoboy86, hous, ho...
1540392        [guess, bu, ride, tour, bu, good, job]
1549636          [hmm, might, go, bed]
706868        [sound, suspici, like, year, million, dream, h...
1027373          [problem]
Name: tokens, Length: 29855, dtype: object
```

```
In [68]: # Storing the tokenized text data and corresponding sentiment labels
X_w2v = ta_df['tokens']
y_w2v = ta_df['sentiment']
```

```
In [69]: # Length of the embedding
len(X_w2v)
```

Out[69]: 29855

```
In [70]: # model.wv.get_vector('there')
```

```
In [71]: # Creating a set of all words in the vocabulary of a trained Word2Vec model
all_word2vec_vocab = set(model.wv.key_to_index)
```

```
In [81]: # Averaging the embedding of a list of tokens
def get_embed(token_list):
    n = len(token_list) # Calculating the length
    embed = np.zeros((200)) # Initializing array of zeros with length 200
    for token in token_list: # Iterating over each token
        if token in all_word2vec_vocab: # Checking if the token is present in
            embed = embed + model.wv.get_vector(token) # If so, embedding vec
        else: # if not in the vocabulary of Word2Vec
            # Using levenshtein distance between the token and takes 2nd elem
            temp = sorted([(textdistance.levenshtein.normalized_distance(e, t,
                e) for e in all_word2vec_vocab], reverse=False)[0]
            embed = embed + model.wv.get_vector(temp) # Adding the selected
    return embed/n # Averaging of the word embeddings

# https://radimrehurek.com/gensim/models/keyedvectors.html
```

```
In [82]: # Applying the function to each row (200 dimensional embedding)
X_w2v_embed = X_w2v.progress_apply(get_embed)
```

```
In [83]: # View embedding
X_w2v_embed
```

```
Out[83]: 541200      [0.06851391276965539, -0.09154102951288223, -0...
750       [0.034500939678400754, -0.0822992317378521, -0...
766711      [0.06930903100261562, -0.07717739801706844, -0...
285055      [0.018783764928230084, -0.12192256841808558, -...
705995      [0.01074135295736293, -0.06879128531242411, 0....
...
263962      [0.02220468013204963, -0.1353233720668975, -0....
1540392     [0.07094248304409641, -0.08008804558111089, -0...
1549636     [0.04116647597402334, -0.10027825180441141, -0...
706868      [0.027969976741587743, -0.08249159273691475, -...
1027373     [0.0009594860021024942, -0.1414019614458084, -...
Name: tokens, Length: 29855, dtype: object
```

```
In [85]: # Creating a dataframe
df_new = pd.DataFrame(X_w2v_embed)
```

```
In [86]: # Resetting index
df_new.reset_index(drop=True, inplace=True)
```

```
In [ ]: # View Dataframe
df_new.head()
```

```
In [88]: # Adding column numbers corresponding to the integers in the list (converted
df_new[list(range(200))] = pd.DataFrame(df_new.tokens.tolist(), index=df_new.index)
```

```
In [89]: # Dropping tokens
df_new.drop('tokens', axis=1, inplace=True)
```

```
In [90]: # View data frame
df_new.head()
```

Out[90]:

	0	1	2	3	4	5	6	7
0	0.068514	-0.091541	-0.103803	0.324743	0.097741	-0.216004	0.068874	0.436102
1	0.034501	-0.082299	-0.048294	0.174695	0.182607	-0.286788	0.030019	0.316800
2	0.069309	-0.077177	-0.095453	0.218893	0.107716	-0.263946	0.037270	0.386705
3	0.018784	-0.121923	-0.020300	0.237091	0.148512	-0.247636	0.019359	0.403607
4	0.010741	-0.068791	0.008459	0.073478	0.079290	-0.127433	0.000541	0.197276
...
29850	0.022205	-0.135323	-0.011604	0.220139	0.107597	-0.204771	-0.014877	0.404414
29851	0.070942	-0.080088	-0.034401	0.194365	0.149497	-0.248896	0.053803	0.404973
29852	0.041166	-0.100278	-0.016394	0.245068	0.108878	-0.250209	0.065530	0.470274
29853	0.027970	-0.082492	-0.038959	0.136888	0.159569	-0.186139	-0.029539	0.321192
29854	0.000959	-0.141402	-0.001341	0.146052	0.151828	-0.251664	0.022471	0.382959

29855 rows × 200 columns

```
In [91]: # Splitting the embedding data into training and testing sets
X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v = train_test_split(df_new,
y_w2v, test_size=0.2, random_state=42)
```

Modeling

The 'Train_Test_Scores' function returns the evaluation metrics (train_acc, test_acc, precision, recall, f1) as output. We will use these metrics can to evaluate and compare different machine learning models.

```
In [80]: # Building a function that will fit the model and
# then fit it to produce predicted values.

def Train_Test_Scores(model,X_train,y_train,X_test,y_test,display=False):

    model.fit(X_train,y_train)
    y_preds = model.predict(X_test)

    # Store the score for later evaluation of the model.

    train_acc = model.score(X_train,y_train)
    test_acc = model.score(X_test,y_test)
    precision = precision_score(y_test,y_preds)
    recall = recall_score(y_test,y_preds)
    f1 = f1_score(y_test,y_preds)

    # Allowing the display to switch off for later on when I just need to fun
    # for other purposes like creating a data frame of the scoring.

    cm = confusion_matrix(y_test,y_preds)
    if display:
        print('Training_Accuracy:', train_acc)
        print('Test_Accuracy:', test_acc)

        print('Precision:', precision)
        print('Recall:', recall)
        print('F1_Score:', f1)

        disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=mode
        disp.plot()
        plt.show()

    return train_acc,test_acc,precision,recall,f1
```

We built a loop we store the results of three machine learning models (GaussianNB RandomForest XGboost) applied to three vectorization techniques (TF-IDF, CountVector, and Word2Vec). WE create a new DataFrame that has columns for the Vectorizer used, the Model used, and various evaluation metrics such as Train Accuracy, Test Accuracy, Precision, Recall, and F1 Score.

In [92]:

```

%%time
# Creating a data frame to collect all the results
# and evaluate them

models_DataFrame = pd.DataFrame(columns=['Vectorizer', 'Model', 'Train_Accuracy',
                                         'Precision', 'Recall', 'F1_score'])

list_models = [GaussianNB(),
               RandomForestClassifier(n_jobs=-1),
               XGBClassifier(n_jobs=-1)
]

model_names = 'GaussianNB RandomForest XGboost'.split()

from tqdm import tqdm
x_probs = []
predictions = []
for vector in ('TF-IDF', 'CountVector', 'Word2Vec'):
    print(vector)
    if vector == "TF-IDF":
        x_train_temp = X_train_tfidf.toarray()
        x_test_temp = X_test_tfidf.toarray()

    elif vector == 'CountVector':
        x_train_temp = X_train_vc.toarray()
        x_test_temp = X_test_vc.toarray()

    elif vector == 'word2vec':
        x_train_temp = X_train_w2v.toarray()
        x_test_temp = X_test_w2v.toarray()

    for model, model_name in tqdm(zip(list_models, model_names)):
        print(model)
        train_acc, test_acc, precision, recall, f1 = Train_Test_Scores(model, x_tr,
                                                                     y_train, x_
                                                                     y_test)

        #     x_probs.append(x_prob)
        #     pipe_models.append(pipe_model)
        models_DataFrame.loc[len(models_DataFrame)] = [vector, model_name, train_
                                                       test_acc, precision, rec

```

TF-IDF

0it [00:00, ?it/s]
GaussianNB()
1it [08:56, 536.25s/it]
RandomForestClassifier(n_jobs=-1)
2it [17:02, 507.05s/it]

```
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=Non
e,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=-1, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
[16:35:30] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691
-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Sta
rting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_m
etric if you'd like to restore the old behavior.

3it [1:51:06, 2222.06s/it]
CountVectorizer
0it [00:00, ?it/s]
GaussianNB()

1it [02:47, 167.58s/it]
RandomForestClassifier(n_jobs=-1)

2it [47:03, 1631.06s/it]
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()','
              n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0
              ,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
[18:11:28] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691
-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Sta
rting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_m
etric if you'd like to restore the old behavior.

3it [1:08:16, 1365.50s/it]
0it [00:00, ?it/s]
Word2Vec
GaussianNB()

1it [00:41, 41.56s/it]
RandomForestClassifier(n_jobs=-1)

2it [02:27, 79.16s/it]
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0
            ,
            reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
            tree_method='exact', validate_parameters=1, verbosity=None)
[18:49:39] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691
-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Sta
rting in XGBoost 1.3.0, the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_m
etric if you'd like to restore the old behavior.
3it [38:10, 763.66s/it]
CPU times: user 2h 38min 6s, sys: 4h 45min 57s, total: 7h 24min 3s
Wall time: 3h 37min 35s
```

In [93]: `models_DataFrame`

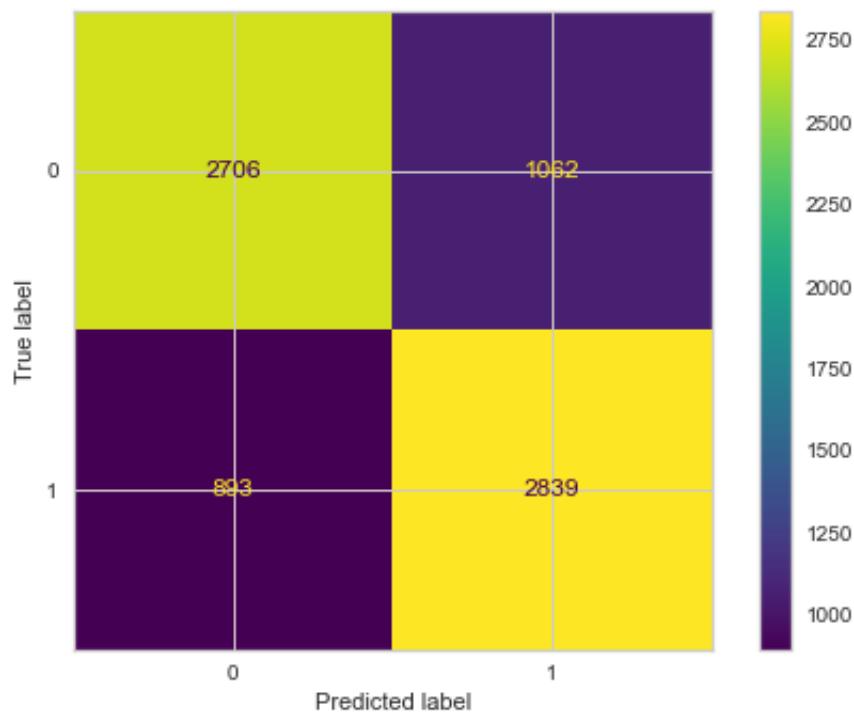
	Vectorizer	Model	Train_Accuracy	Test_Accuracy	Precision	Recall	F1_score
0	TF-IDF	GaussianNB	0.754800	0.534400	0.587083	0.216774	0.316634
1	TF-IDF	RandomForest	0.994978	0.738400	0.727040	0.759378	0.742857
2	TF-IDF	XGboost	0.782800	0.722000	0.687799	0.808146	0.743132
3	CountVector	GaussianNB	0.733111	0.527733	0.587477	0.170954	0.264840
4	CountVector	RandomForest	0.995111	0.735867	0.732413	0.739282	0.735831
5	CountVector	XGboost	0.765022	0.730533	0.693159	0.822615	0.752359
6	Word2Vec	GaussianNB	0.733111	0.527733	0.587477	0.170954	0.264840
7	Word2Vec	RandomForest	0.995111	0.735600	0.729950	0.743837	0.736828
8	Word2Vec	XGboost	0.765022	0.730533	0.693159	0.822615	0.752359

Interpreting results of the best performing model

The models in general performed well on the training data and did not do as well on the testing data. They mostly were all overfitting. It is possible that if we increase the data the model will do better or if we use a more advanced deep learning approach which we will attempt both in the part 2. In the meantime, the top performer is Random Forest using the TF-IDF method for vectorization.

In [97]: `Train_Test_Scores(RandomForestClassifier(n_jobs=-1),x_train_tfidf,y_train,
 x_test_tfidf,y_test,True)`

```
Training_Accuracy: 0.9949777777777777
Test_Accuracy: 0.7393333333333333
Precision: 0.7277621122789029
Recall: 0.7607181136120043
F1_Score: 0.7438752783964365
```



```
Out[97]: (0.9949777777777777,
0.7393333333333333,
0.7277621122789029,
0.7607181136120043,
0.7438752783964365)
```

```
In [106...]: # Define classifier and fit it
clf = RandomForestClassifier(n_jobs=-1)
clf.fit(X_train_tfidf, y_train)
```

```
Out[106...]: ▼      RandomForestClassifier
RandomForestClassifier(n_jobs=-1)
```

```
In [118...]: # Get predictors from Random Forest using TF-IDF
y_pred = clf.predict(X_test_tfidf)
```

```
In [119...]: # Classification report
target_names = ['Negative', 'Positive']
print(classification_report(y_test, y_pred, target_names = target_names))
```

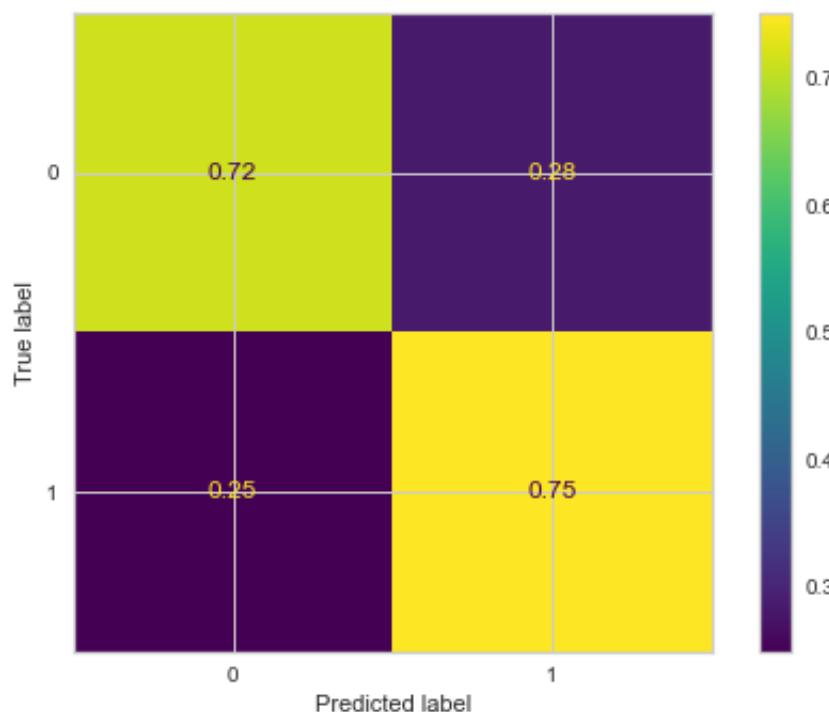
	precision	recall	f1-score	support
Negative	0.74	0.72	0.73	3768
Positive	0.72	0.75	0.74	3732
accuracy			0.73	7500
macro avg	0.73	0.73	0.73	7500
weighted avg	0.73	0.73	0.73	7500

In [120...]

```
#cm = confusion_matrix(y_test,y_pred)
cm1 = confusion_matrix(y_test,y_pred,normalize='true')

#disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
displ = ConfusionMatrixDisplay(confusion_matrix=cm1,display_labels=[0,1])
#disp.plot()
displ.plot()
plt.show()

# https://scikit-learn.org/stable/modules/generated/sklearn.
# metrics.ConfusionMatrixDisplay.html
```



Random Forest shows accuracy of 73%, which means that it correctly classified 73% of the samples in the test set both negative and positive tweets correctly.

Please see part 2 (deep learning) in the colab notebook on GitHub.

Table of Contents

- Preparing to model the data
 - Saving the model
 - Loading the model
 - Results
 - LIME
 - Testing a Review with LIME
- Testing on Product Reviews
- Topic Modeling
 - LIME

Table of Contents

- Preparing to model the data
 - Saving the model
 - Loading the model
- Results
 - Testing a Review with LIME
- Testing on Product Reviews

TWEETER SENTIMENT ANALYSIS | NLP

Part 2 Deep Learning

To upload data to Google Colab, follow these steps:

```
In [1]: # Create a new folder
!mkdir .kaggle
## upload.json
# ! chmod 600 .kaggle/kaggle.json
```



```
In [2]: # Changing the permissions of the kaggle.json
! chmod 600 .kaggle/kaggle.json
```



```
In [4]: # Move to the root path
!mv .kaggle /root/
```



```
In [6]: # download data from kaggle
!kaggle datasets download -d kazanova/sentiment140
!kaggle datasets download -d crowdflower/twitter-airline-sentiment
```

sentiment140.zip: Skipping, found more recently modified local copy (use --force to force download)
 twitter-airline-sentiment.zip: Skipping, found more recently modified local copy (use --force to force download)


```
In [7]: ! unzip sentiment140.zip
! unzip twitter-airline-sentiment.zip
```

Archive: sentiment140.zip
 inflating: training.1600000.processed.noemoticon.csv
 Archive: twitter-airline-sentiment.zip
 inflating: Tweets.csv
 inflating: database.sqlite

Table of Contents

- Preparing to model the data
 - Saving the model
 - Loading the model
 - Results
 - LIME
 - Testing a Review with LIME
- Testing on Product Reviews
- Topic Modeling
 - LIME

Table of Contents

- Preparing to model the data
 - Saving the model
 - Loading the model
- Results
 - Testing a Review with LIME
- Testing on Product Reviews

TWEETER SENTIMENT ANALYSIS | NLP

Part 2 Deep Learning

To upload data to Google Colab, follow these steps:

```
In [1]: # Create a new folder
!mkdir .kaggle
## upload.json
# ! chmod 600 .kaggle/kaggle.json
```



```
In [2]: # Changing the permissions of the kaggle.json
! chmod 600 .kaggle/kaggle.json
```



```
In [4]: # Move to the root path
!mv .kaggle /root/
```



```
In [6]: # download data from kaggle
!kaggle datasets download -d kazanova/sentiment140
!kaggle datasets download -d crowdflower/twitter-airline-sentiment
```

sentiment140.zip: Skipping, found more recently modified local copy (use --force to force download)
 twitter-airline-sentiment.zip: Skipping, found more recently modified local copy (use --force to force download)


```
In [7]: ! unzip sentiment140.zip
! unzip twitter-airline-sentiment.zip
```

Archive: sentiment140.zip
 inflating: training.1600000.processed.noemoticon.csv
 Archive: twitter-airline-sentiment.zip
 inflating: Tweets.csv
 inflating: database.sqlite

In [8]:

```
# remove all zip file
!rm product-tweets-dataset.zip
!rm sentiment140.zip
!rm twitter-airline-sentiment.zip
```

rm: cannot remove 'product-tweets-dataset.zip': No such file or directory

In [9]:

```
# Import Sklearn libraries to build models
from sklearn.feature_extraction.text import TfidfVectorizer # vectorize words
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC

from sklearn.metrics import ConfusionMatrixDisplay
# from sklearn.metrics import plot_confusion_matrix
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import GridSearchCV
# Import Libraries to perform computation and do visualization.
import pandas as pd
import numpy as np
np.random.seed(0)
import seaborn as sns
import matplotlib.pyplot as plt
import string

# Import nltk to check english lexicon.
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet, stopwords
from nltk import word_tokenize, FreqDist
from nltk import pos_tag # for Parts of Speech tagging
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
# Generate wordcloud for word distribution visualization.
from wordcloud import WordCloud

# Generating random numbers.
import random

from xgboost import XGBClassifier

# Transforms text to a fixed-length vector of integers.
from gensim.models import Word2Vec

import os

# !pip install -q textdistance
# import textdistance
# Efficient functions to search in strings.
import re as re

# Import images for world cloud.
from PIL import Image, ImageDraw, ImageFont

from tqdm.notebook import tqdm

from os import path
from os import environ
```

In [10]:

```
import tensorflow as tf
```

In [11]:

```
# !pip install tensorflow
```

In [12]:

```
# Uploading data
DATASET_COLUMNS=[ "sentiment", "ids", "date", "flag", "user", "tweet"]
```

```

DATASET_ENCODING = "ISO-8859-1"
ta_df = pd.read_csv('training.1600000.processed.noemoticon.csv',
                     encoding=DATASET_ENCODING, names=DATASET_COLUMNS)

ta_df.head() #View Tweets Data Frame

```

Out[12]:

	sentiment	ids	date	flag	user	tweet
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 [13]:

```
# Generating a random sample of 1,000,000 tweets from a 1.6 million database
ta_df = ta_df.sample(1000000,random_state=42)
```

In [14]:

```
# Checking shape
ta_df.shape
```

Out[14]: (1000000, 6)

In [15]:

```
# Dropping 'flag' and 'ids'
ta_df.drop(['flag','ids'], axis=1, inplace = True)
```

In [16]:

```
# Checking the the data is balanced
ta_df['sentiment'].value_counts()
```

Out[16]:

0	500167
4	499833
Name: sentiment, dtype: int64	

In [17]:

```
# The variable target labeling in the data set is binary: 0 and 4,
# where 0 represents the tweets that
# are negative and 4 represents the tweets that are positive.
ta_df['sentiment'] = ta_df['sentiment'].replace(4,1)
ta_df.head()
```

Out[17]:

	sentiment	date	user	tweet
541200	0	Tue Jun 16 18:18:12 PDT 2009	LaLaLindsey0609	@chrishasboobs AHHH I HOPE YOUR OK!!!
750	0	Mon Apr 06 23:11:14 PDT 2009	sexygrneyes	@misstoriblack cool , i have no tweet apps fo...
766711	0	Tue Jun 23 13:40:11 PDT 2009	sammydearr	@TiannaChaos i know just family drama. its la...
285055	0	Mon Jun 01 10:26:07 PDT 2009	Lamb_Leanne	School email won't open and I have geography ...
705995	0	Sat Jun 20 12:56:51 PDT 2009	yogicerdit0	upper airways problem

CLEANING THE DATA

We repeat from Part 1 the preprocessing and cleaning the tweets data using the preprocess() function.

```
In [18]: tqdm.pandas() # Monitor completion
ps = PorterStemmer() # defining and applying stemming to words in the text.

# text patterns that we want to remove from the text
TEXT_CLEANING_RE = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"

# Replace 3 or more consecutive letters by 2 letter.
sequencePattern = r"(.)\1\1"
seqReplacePattern = r"\1\1"
nltk.download('stopwords')
# Listst of common words we will removed from text.

stop_words = set(stopwords.words("english"))

# The preprocess() function takes in the tweet and
# applies various text cleaning techniques to it.

def preprocess(text,apply_stem=True):

    # Remove link,user and special characters
    text = re.sub(TEXT_CLEANING_RE, ' ', str(text).lower()).strip()
    text = re.sub(sequencePattern, seqReplacePattern, text)

    tokens = [] # Initializing an empty list to store tokenized words.
    for token in text.split():
        if token not in stop_words: # Checking if the word is not a stopword

            if apply_stem:
                # Stemming to the word using the PorterStemmer if True
                tokens.append(ps.stem(token))
            else:
                # Adding the original word if stemming is not applied.
                tokens.append(token)
    # Joining and retuning the list of tokenized words into a single string
    return " ".join(tokens)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]  Unzipping corpora/stopwords.zip.
```

```
In [19]: # Creating a new column in the DataFrame
# Using lambda function applying the 'preprocess()'
# function to each row in the 'tweet' column.
ta_df['clean_tweet'] = ta_df['tweet'].progress_apply(lambda x:preprocess(x,True))
```

```
In [20]: # Creating a new column in the DataFrame without stemming for wordcloud.
# Using lambda applying the 'preprocess()' function to each row in the 'tweet'
# column but adding 'false'.
ta_df['clean_tweet_wt_stem'] = ta_df['tweet'].progress_apply(lambda
                                                               x:preprocess(x, False))
```

```
In [21]: # Splitting data target and clean tweets

x = ta_df['clean_tweet'][:1000000]
y = ta_df['sentiment'][:1000000]
```

```
In [22]: # Spliting your dataset into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.25,
                                                    random_state = 43)
```

```
In [23]: # Applying tokenizing to the corpus using lambda
corpus = X.progress_apply(lambda line:line.split())
```

```
In [24]: # View corpus
corpus
```

```
Out[24]: 541200          [ahh, hope, ok]
750           [cool, tweet, app, razr, 2]
766711        [know, famili, drama, lame, hey, next, time, u...
285055        [school, email, open, geographi, stuff, revis, ...
705995        [upper, airway, problem]
...
21340         [bad, time, lap, top, broken, good, time, foun...
80620           [yeah, fun, think, movi]
411912        [ministri, agricultur, forest, keep, take, cat...
798815         [sad, news, farrah, fawcett, pass, away]
1406049        [indo, pro, shop, trocar, meu, celular, quando...
Name: clean_tweet, Length: 1000000, dtype: object
```

```
In [25]: # Train Word2Vec Model on corpus
WORD2VEC_MODEL = Word2Vec(corpus,
                           min_count=1,           # word frequency
                           vector_size=300,       # dimension of word embeddings
                           workers=os.cpu_count(), # Number of processors
                           sg=0 # default, 1 for skipgram and 0 for cbow
                           )
```

```
In [26]: # help(Word2Vec)
```

```
In [27]: # Adding a new column and applying token to each row.
ta_df['tokens'] = ta_df['clean_tweet'].progress_apply(lambda line: line.split())
```

When the DataFrame is filtered using this True if len(line) else False, only the rows with at least one token in the 'tokens' column will be retained.

```
In [28]: # Removing rows from ta_df if the 'tokens' column contains an empty list
ta_df = ta_df[ta_df['tokens'].apply(lambda line: True if len(line) else False)]
```

```
In [29]: # all_word2vec_vocab = model.wv.vocab.keys()
```

```
In [31]: # Max character size for a tweet
W2V_SIZE = 300 # its vector dimension
```

Preparing to model the data

Given that the results were not great from our machine learning models, we will implement the Long Short-Term Memory ("LSTM") model, which works better with a larger data set. Therefore we increased our data to 1,000,000 tweets.

We are utilizing the deep model LSTM to achieve better results. LSTM is a type of Recurrent Neural Network (RNN) a type of artificial neural network that is used to process sequential data.

```
In [34]: # Importing Keras libraries
from keras.preprocessing.text import Tokenizer
# from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, Embedding, Flatten
from keras.layers import Conv1D, MaxPooling1D, LSTM
from keras import utils
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
# from keras_preprocessing.sequence import pad_sequences
# from keras_preprocessing.sequence import pad_sequences
```

```
In [35]: # from keras.preprocessing.sequence import pad_sequences
# Downloading tensorflow library
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
In [36]: %%time
tokenizer = Tokenizer() # Initializing a new object
tokenizer.fit_on_texts(X_train) # Tokenizer will learn the vocabulary of the text
# data and assign a unique index to each word

vocab_size = len(tokenizer.word_index) + 1 # Vocabulary size (tokenizer reserves
print("Total words", vocab_size)
```

Total words 162955
CPU times: user 8.64 s, sys: 55.1 ms, total: 8.7 s
Wall time: 8.67 s

```
In [37]: # tokenizer.word_index
```

```
In [38]: # KERAS
# Maximum length of the tweet.
SEQUENCE_LENGTH = 300

#Number of times it passes through the training data.
EPOCHS = 10

# Number of training examples/samples to process at once during each iteration.
BATCH_SIZE = 1024

# https://medium.com/@dclengacher/keras-lstm-recurrent-neural-networks-c1f5febde
```

```
In [39]: max(X_train.apply(lambda x: len(x))) # the maximum length of the text
```

Out[39]: 174

Converting the tweets in training and testing to sequences of word indices using the tokenizer then padding them to have a fixed length of 300.

```
In [40]: # Converting training and testing data to sequences of word
# indices to prepare for the model.
x_train = pad_sequences(tokenizer.texts_to_sequences(X_train),
maxlen=SEQUENCE_LENGTH)
x_test = pad_sequences(tokenizer.texts_to_sequences(X_test),
maxlen=SEQUENCE_LENGTH)
```

Word2Vec

Embedding matrix using a pre-trained Word2Vec model for words in the tweet learned by the tokenizer. The embedding matrix will be used to establish the weights of an embedding layer in a neural network.

```
In [41]: embedding_matrix = np.zeros((vocab_size, W2V_SIZE)) # Initializing an empty
# embedding matrix with the shape
for word, i in tokenizer.word_index.items(): # Looping over each word in tweet
    if word in WORD2VEC_MODEL.wv: # Checking if the word is in Word2Vec model
        embedding_matrix[i] = WORD2VEC_MODEL.wv[word] # If yes, it assigns a vector
print(embedding_matrix.shape) # Printing shape
```

(162955, 300)

```
In [42]: # embedding_matrix[95]
```

Creating an embedding layer using the Embedding class from Keras.

```
In [43]: # Initializing embedding layers for our model
embedding_layer = Embedding(vocab_size, W2V_SIZE, weights=[embedding_matrix],
input_length=SEQUENCE_LENGTH, trainable=False)
```

Layers to LSTM model

```
In [44]: KERAS_MODEL = Sequential() # Creating a new sequential model
KERAS_MODEL.add(embedding_layer) # Adding embedding layer to the model.

# Dropout is a fraction of the neurons will drop in a layer during training.
KERAS_MODEL.add(Dropout(0.5)) # Adding a dropout layer
# Adding LSTM hidden layer with 100 neurons

KERAS_MODEL.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
# Adding a dense output layer to the model with a one one output and
# a sigmoid activation function
KERAS_MODEL.add(Dense(1, activation='sigmoid'))

KERAS_MODEL.summary() # Print summary

# https://heartbeat.comet.ml/using-a-keras-long-shortterm-memory-
# lstm-model-to-predict-stock-prices-a08c9f69aa74
```

WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 300, 300)	48886500
dropout (Dropout)	(None, 300, 300)	0
lstm (LSTM)	(None, 100)	160400
dense (Dense)	(None, 1)	101

Total params:
49,047,001
Trainable params: 160,501
Non-trainable params: 48,886,500

Compiling the model, specifying the binary cross-entropy loss function, the Adam optimizer, and the accuracy metric.

```
In [45]: # Compiling the model

KERAS_MODEL.compile(loss='binary_crossentropy',
                      optimizer="adam",
                      metrics=['accuracy'])
```

Learning the rate when the validation loss stops improving, and EarlyStopping stops training when the validation accuracy stops improving.

```
In [46]: # Initializing callback
callbacks = [ ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown=0),
              EarlyStopping(monitor='val_accuracy', min_delta=1e-4, patience=5) ]
```

```
In [47]: x_train.shape
```

```
Out[47]: (750000, 300)
```

```
In [48]: y_train.shape
```

```
Out[48]: (750000,)
```

```
In [49]: history = KERAS_MODEL.fit(x_train, y_train,
                                batch_size=BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_split=0.1,
                                verbose=1,
                                callbacks=callbacks)
```

```

Epoch 1/10
660/660 [=====] - 591s 884ms/step - loss: 0.5105 - accuracy: 0.7455 - val_loss: 0.4719 - val_accuracy: 0.7728 - lr: 0.0010
Epoch 2/10
660/660 [=====] - 572s 866ms/step - loss: 0.4881 - accuracy: 0.7614 - val_loss: 0.4689 - val_accuracy: 0.7754 - lr: 0.0010
Epoch 3/10
660/660 [=====] - 571s 865ms/step - loss: 0.4800 - accuracy: 0.7668 - val_loss: 0.4583 - val_accuracy: 0.7797 - lr: 0.0010
Epoch 4/10
660/660 [=====] - 569s 862ms/step - loss: 0.4746 - accuracy: 0.7701 - val_loss: 0.4535 - val_accuracy: 0.7829 - lr: 0.0010
Epoch 5/10
660/660 [=====] - 584s 885ms/step - loss: 0.4706 - accuracy: 0.7731 - val_loss: 0.4527 - val_accuracy: 0.7838 - lr: 0.0010
Epoch 6/10
660/660 [=====] - 574s 869ms/step - loss: 0.4686 - accuracy: 0.7742 - val_loss: 0.4535 - val_accuracy: 0.7850 - lr: 0.0010
Epoch 7/10
660/660 [=====] - 574s 870ms/step - loss: 0.4661 - accuracy: 0.7756 - val_loss: 0.4492 - val_accuracy: 0.7866 - lr: 0.0010
Epoch 8/10
660/660 [=====] - 576s 873ms/step - loss: 0.4649 - accuracy: 0.7763 - val_loss: 0.4493 - val_accuracy: 0.7876 - lr: 0.0010
Epoch 9/10
660/660 [=====] - 574s 870ms/step - loss: 0.4632 - accuracy: 0.7775 - val_loss: 0.4479 - val_accuracy: 0.7873 - lr: 0.0010
Epoch 10/10
660/660 [=====] - 574s 870ms/step - loss: 0.4621 - accuracy: 0.7782 - val_loss: 0.4464 - val_accuracy: 0.7884 - lr: 0.0010

```

Saving the model

```
In [50]: # Saving the model so that we don't need to keep training it.
KERAS_MODEL.save('keras_model.h5') # Saving LTS
WORD2VEC_MODEL.save("model.w2v") # Saving Word2Vec
```

Loading the model

```
In [51]: from tensorflow.keras.models import load_model # Import to load the model
# Loading model
keras_model = load_model('keras_model.h5')

# summarize model.
# keras_model.summary()
```

WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.

Results

```
In [52]: # Showing Accuracy and Loss
score = keras_model.evaluate(x_test, y_test, batch_size=BATCH_SIZE)
print()
print("ACCURACY:", score[1])
print("LOSS:", score[0])
```

```
245/245 [=====] - 26s 104ms/step - loss: 0.4517 - accuracy: 0.7864
```

```
ACCURACY: 0.7863839864730835
LOSS: 0.45171108841896057
```

We are plotting the accuracy and loss values over the epochs of the model training, so we can get an idea of how the model is performing over time of the iteration in the neural network setting.

```
In [53]: # Defining variables

# Returning when calling the fit() method on a Keras LSTM model.
val_acc = history.history['val_accuracy']
acc = history.history['accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
# it will give the epoch it stopped at after training or 0 if it didn't early stop
n = callbacks[1].stopped_epoch

# get X axis range
n = range(1,EPOCHS+1) if n==0 else range(1,n+2)

# epochs = range(EPOCHS) # range object 10

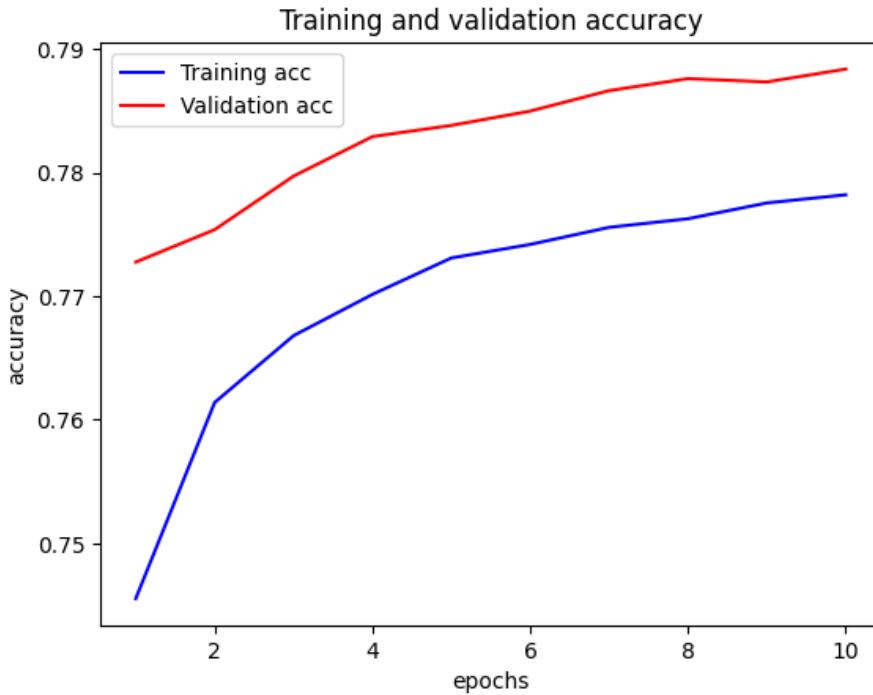
# Plotting training accuracy over the epochs in blue.
plt.plot(n, acc, 'b', label='Training acc')
plt.plot(n, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel("epochs") # Labeling
plt.ylabel("accuracy") # Labeling
plt.legend()

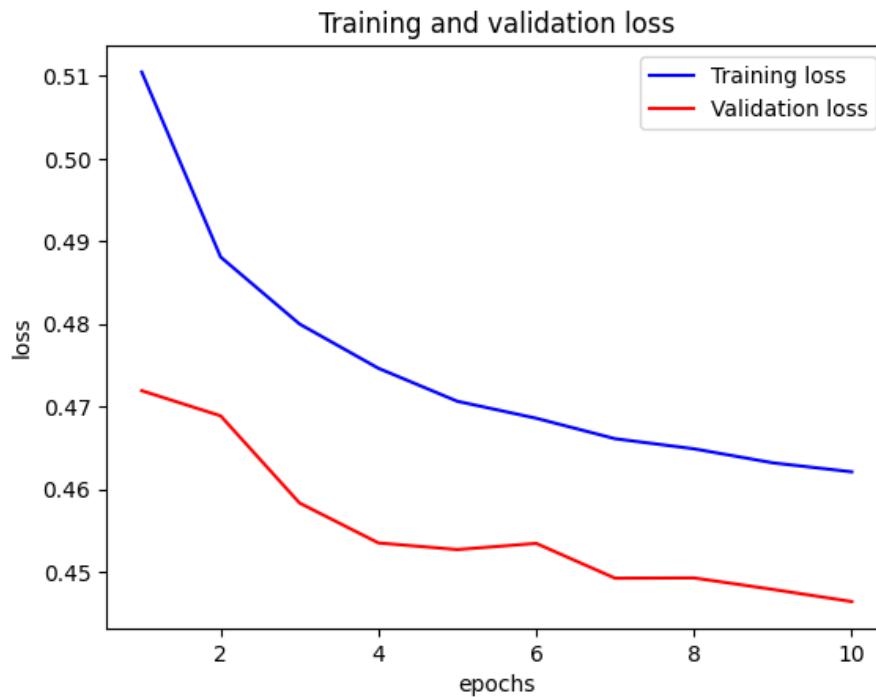
plt.figure()

# Plotting training loss over the epochs in red.
plt.plot(n, loss, 'b', label='Training loss')
plt.plot(n, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel("epochs") # Labeling
plt.ylabel("loss") # Labeling
plt.legend()

plt.show()

# https://stackoverflow.com/questions/41908379/keras-plot-training-validation-and-test-set-accuracy
```

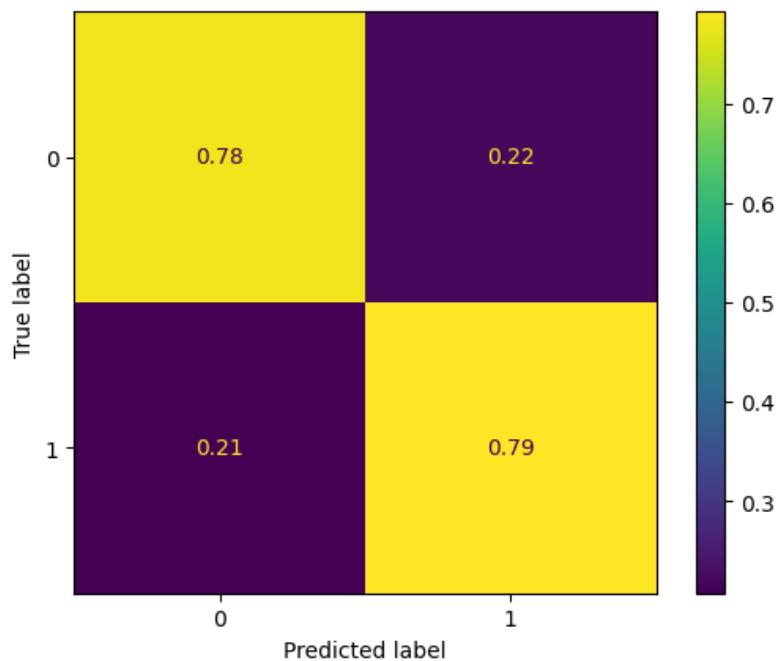
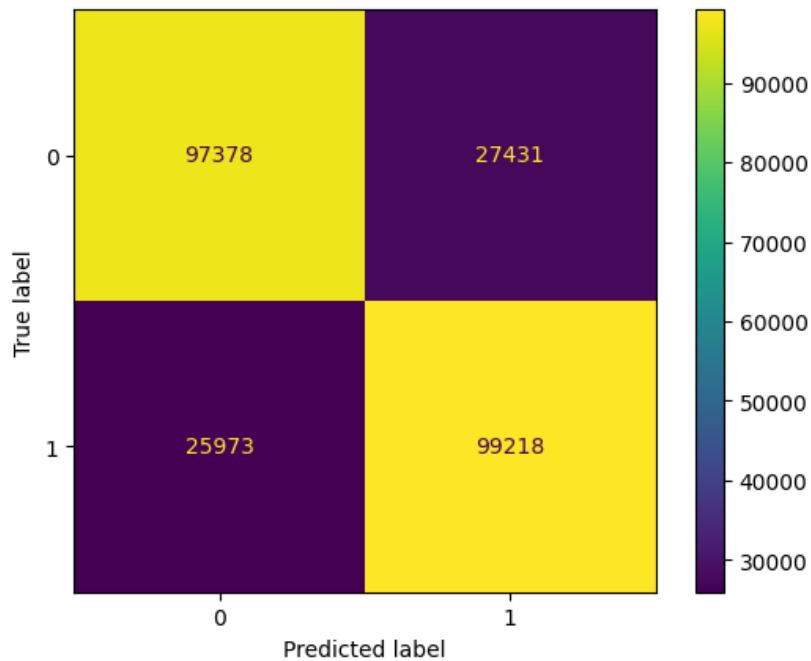




The training accuracy is increasing, and the validation accuracy is also increasing. We are seeing the same pattern with the training loss graph. Loss is decreasing, and the validation loss as well. This suggests that the model is learning the patterns in the data and generalizing well to new data. In both graphs, when the epochs pass 6, the training and the validation are starting to get a little closer to one another. With higher than 10 epochs the model might be overfitting to the training data and not teaching the new data.

```
In [54]:  
y_test_1d = list(y_test)  
scores = keras_model.predict(x_test, verbose=1, batch_size=1000)  
y_pred_1d = [0 if score[0]<0.5 else 1 for score in scores ]  
  
250/250 [=====] - 25s 100ms/step
```

```
In [55]:  
cm = confusion_matrix(y_test_1d,y_pred_1d)  
cm1 = confusion_matrix(y_test_1d,y_pred_1d,normalize='true')  
  
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])  
disp1 = ConfusionMatrixDisplay(confusion_matrix=cm1,display_labels=[0,1])  
disp.plot()  
disp1.plot()  
plt.show()
```



```
In [56]: print(classification_report(y_test_1d,y_pred_1d))
```

	precision	recall	f1-score	support
0	0.79	0.78	0.78	124809
1	0.78	0.79	0.79	125191
accuracy			0.79	250000
macro avg	0.79	0.79	0.79	250000
weighted avg	0.79	0.79	0.79	250000

Accuracy: The accuracy is the proportion of correctly labeled class as negative or positive tweet.
In this case, the accuracy is 79%. the LSTM model correctly classified 79% of the time both negative and positive tweets.

LIME

LIME (Local Interpretable Model-agnostic Explanations) provides an explanation for the prediction made by our LSTM on a specific tweet from the testing data.

```
In [57]: !pip install -q lime
_____
Preparing metadata (setup.py) ... done
Building wheel for lime (setup.py) ... done
275.7/275.7 KB 5.8 MB/s eta 0:00:00
```

We define a predict_proba function that takes a list of tweet texts as input, preprocesses and tokenizes the texts, and then returns the predicted probabilities for each class (negative and positive) using the LSTM model.

```
In [58]: # Import to explain text classifiers.
from lime.lime_text import LimeTextExplainer
class_names=[0,1]

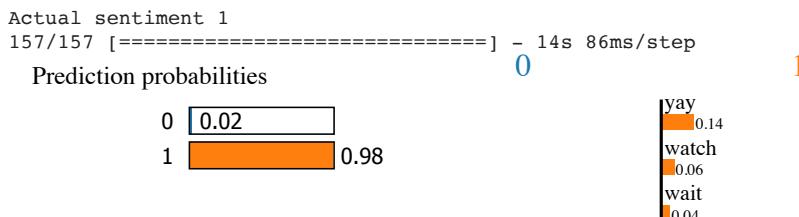
def predict_proba(tweet_text):
    example = pad_sequences(tokenizer.texts_to_sequences(list(tweet_text)),
                            maxlen=SEQUENCE_LENGTH) # Tokenizes the tweets then pads
    # Training LSTM predict sentiment probabilities
    pred=keras_model.predict(example)
    returnable=[] # initializing list to store predicted probabilities.
    for i in pred:
        temp=i[0]
        # Predictes for each class in all tweets
        returnable.append(np.array([1-temp,temp]))
    return np.array(returnable)
# https://towardsdatascience.com/interpreting-an-lstm-through-lime-e294e6ed3a03
# print(pred)
# temp = pred[0][0]
# return np.array([1-temp,temp]).reshape(1,2)
```

```
In [59]: # View test date
X_test
```

```
Out[59]: 956610                      spongebob isnt
          483432                      rain
          1523842                     yay wait watch
          48041      morn mother natur left monthli present fun fun...
          366878                     mess huh
          ...
          560578                      ack happen
          916703                      happy mother day everyon
          631662      sick cough fever bodi ach headache runni nose g...
          549361                     hold anymorc act like fine
          283913      lt got call earli work b c dude work call mill...
Name: clean_tweet, Length: 250000, dtype: object
```

```
In [60]: print("Actual sentiment",y_test[1523842])
# positive (orange), Negative (blue)

# Initializing a LimeTextExplainer object to explain the tweet.
explainer= LimeTextExplainer(class_names=class_names)
# Returning the predicted probabilities for each class.
exp = explainer.explain_instance(X_test[1523842],
                                  predict_proba).show_in_notebook(text=True)
```

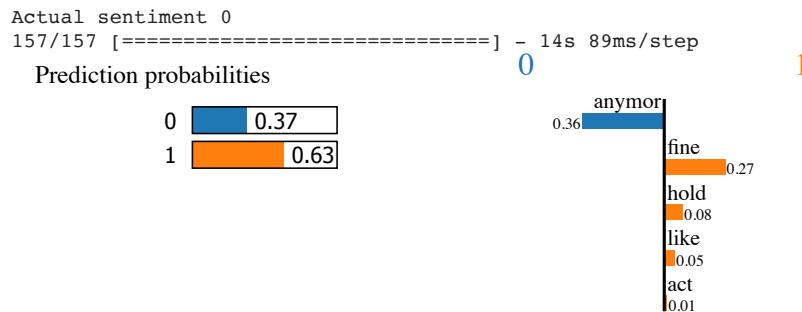


Text with highlighted words

yay wait watch

The model predicts probabilities of 65% for the negative class and 35% for a positive class. This means that the model is more confident that the tweet is labeled as negative.

```
In [61]: print("Actual sentiment",y_test[549361])
# positive (orange), Negative (blue)
# Initializing a LimeTextExplainer object to explain the tweet.
explainer= LimeTextExplainer(class_names=class_names)
exp = explainer.explain_instance(X_test[549361],
                                predict_proba).show_in_notebook(text=True)
```



Text with highlighted words

hold anymor act like fine

The model is very confident that the tweet belongs to the positive class, as the probability is 100%.

Testing a Review with LIME

```
In [62]: # print("Actual sentiment",y_test[339425])
# positive (orange), Negative (blue)
# explainer= LimeTextExplainer(class_names=class_names)
# Returning the predicted probabilities for each class.
exp = explainer.explain_instance("I wish the interface was easier to navigate.",
                                predict_proba).show_in_notebook(text=True)
```



Text with highlighted words

I wish the interface was easier to navigate.

```
In [63]: # https://www.reviews.io/company-reviews/store/tshirtfella
```

Testing on Product Reviews

<https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment>

Airline Reviews on Twitter

For the business presentation, we will be conducting an analysis on a dataset containing actual customer feedback on a specific airline. This exercise primarily aims to demonstrate the

potential insights that can be gained through data-driven approaches in enhancing our understanding of customer feedback.

```
In [64]: # https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment?select=Tweets.csv

# Reading and viewing data set (feedback tweets on airlines)
df_airline = pd.read_csv('Tweets.csv')
df_airline.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negative
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

We are gathering equal amount of positive and negative tweets, resulting in a balanced dataset. Each sentiment category will have 2,363 tweets.

```
In [65]: # Store only positive emotions
neg_al = df_airline[df_airline['airline_sentiment'].isin(['negative'])].sample(2363, random_state=42)
```

```
In [66]: # Store only positive emotions
pos_al = df_airline[df_airline['airline_sentiment'].isin(['positive'])]
```

```
In [67]: #Merging the postive and negative tweets
df_airline = pd.concat([pos_al, neg_al])
```

```
In [68]: # View dataframe
df_airline.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negative
1	570301130888122368	positive	0.3486	NaN	
6	570300616901320704	positive	0.6745	NaN	
8	570299953286942721	positive	0.6559	NaN	
9	570295459631263746	positive	1.0000	NaN	

tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negative
11	positive	1.0000	NaN	

```
In [69]: # View the tweets
df_airline['text']
```

```
Out[69]: 1      @VirginAmerica plus you've added commercials t...
6      @VirginAmerica yes, nearly every time I fly VX...
8      @virginamerica Well, I didn't...but NOW I DO! :-D
9      @VirginAmerica it was amazing, and arrived an ...
11     @VirginAmerica I &lt;3 pretty graphics. so muc...
...
894     @united where's my damn bag??
9875    @USAirways 5 hours on the phone waiting for a ...
10003   @USAirways flight 838 now leaving after 6 hour...
11674   @USAirways flt 5302 CLT to DAY supposed to dep...
9786    @USAirways over two hours on hold to talk to a...
Name: text, Length: 4726, dtype: object
```

```
In [70]: # Downloading Spacy
```

```
!python -m spacy download en_core_web_lg
```

```
2023-04-03 15:39:39.988248: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting en-core-web-lg==3.5.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_lg-3.5.0/en_core_web_lg-3.5.0-py3-none-any.whl (587.7 MB)
  ━━━━━━━━━━━━━━━━ 587.7/587.7 MB 2.6 MB/s eta 0:00:00
Requirement already satisfied: spacy<3.6.0,>=3.5.0 in /usr/local/lib/python3.9/dist-packages (from en-core-web-lg==3.5.0) (3.5.1)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (6.3.0)
Requirement already satisfied: typer<0.8.0,>=0.3.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (0.7.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (2.27.1)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (1.0.9)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (2.0.8)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (2.0.7)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (3.0.8)
Requirement already satisfied: thinc<8.2.0,>=8.1.8 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (8.1.9)
Requirement already satisfied: srslly<3.0.0,>=2.4.3 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (2.4.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (67.6.1)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.11.0,>=1.7.4 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (1.10.7)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (1.22.4)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (3.1.2)
Requirement already satisfied: pathy>=0.10.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (0.10.1)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (1.0.4)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (3.3.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (3.0.12)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (23.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (4.65.0)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.9/
```

```

dist-packages (from spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (1.1.1)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python
3.9/dist-packages (from pydantic!=1.8,!1.8.1,<1.11.0,>=1.7.4->spacy<3.6.0,>=3.
5.0->en-core-web-lg==3.5.0) (4.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/di
st-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->en-core-web-
lg==3.5.0) (2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/pytho
n3.9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->en-core-w
eb-lg==3.5.0) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-pac
kages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.
0) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.
9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->en-core-web-
lg==3.5.0) (1.26.15)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.9/di
st-packages (from thinc<8.2.0,>=8.1.8->spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.
0) (0.7.9)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python
3.9/dist-packages (from thinc<8.2.0,>=8.1.8->spacy<3.6.0,>=3.5.0->en-core-web-lg
==3.5.0) (0.0.4)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.9/d
ist-packages (from typer<0.8.0,>=0.3.0->spacy<3.6.0,>=3.5.0->en-core-web-lg==3.
5.0) (8.1.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/di
st-packages (from jinja2->spacy<3.6.0,>=3.5.0->en-core-web-lg==3.5.0) (2.1.2)
Installing collected packages: en-core-web-lg
Successfully installed en-core-web-lg-3.5.0
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_lg')

get_entity function passes the tweets and stores them in the list 'entity'. It extracts
organizations & products in the text and adds them to the list.

```

In [71]:

```

def get_entity(x):
    entity = [] # Initialize list
    doc = nlp(x) # #xtract entities
    for ent in doc.ents: # loops and checks for org. and product
        if ent.label_ in ('ORG', 'PRODUCT'):
            entity.append((ent.text, ent.label_)) # Adding to the list
    return entity # Returns filtered data

```

Topic Modeling

Filtering through only the negative tweets and look for problems that people might be experiencing,

In [133...]

```

import pandas as pd
import os
from tqdm.notebook import tqdm
tqdm.pandas()
import re
import spacy
import numpy as np
import seaborn as sns
from spacy.language import Language
from spacy import displacy
from gensim.models.ldamodel import LdaModel
from gensim.corpora.dictionary import Dictionary
from gensim.models.coherencemodel import CoherenceModel
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation

import warnings
warnings.filterwarnings('ignore')
# result = pd.DataFrame([],columns=[ 'Corpus' , 'No_of_Comments' , 'No_of_Tokens' ])

```

In [79]:

```

# loads the language model of the spaCy library.
nlp = spacy.load('en_core_web_lg')

```

In [139...]

```

# Creating doc using nlp language model named entitie
#doc = nlp(text)

```

```
In [78]: # Creating a new column in the DataFrame called 'clean_tweet',
# Which stores the cleaned tweets ,using'preprocess()' function without stemming.
df_airline['clean_tweet'] = df_airline['text'].progress_apply(lambda x:preproces
```

We need to convert the cleaned tweets (`x_test`) into padded sequences of fixed length (SEQUENCE_LENGTH :300) using the tokenizer that we have previously defined. This step is essential for preparing the input data for a deep learning model, LTSM.

```
In [ ]: # Importing the Spacy library
```

We will be deploying topic modeling, which allow us to identify the topics presented in the tweets reviews.

The function returns the matrix as well as the `tfidf_featurizer` object, which can be used later to transform new text data into the same format as the training data.

- `max_df` = threshold for ignoring words that appear too frequently
- `min_df` = parameter specifies the threshold for ignoring words that appear too infrequently.

The function returns the matrix as well as the `tfidf_featurizer` object, which can be used later to transform new text data into the same format as the training data.

```
In [81]: def analysis(docs,vocab_size):

    # Sets up features for TF-IDF
    tfidf_featurizer = TfidfVectorizer(max_features=vocab_size,
                                        max_df=0.95, stop_words='english')

    # count_text_vectors = count_text_vectorizer.fit_transform(docs)

    X_tfidf = tfidf_featurizer.fit_transform(docs) # Transform to TF-IDF

    # feature_names = tfidf_featurizer.get_feature_names_out()
    # X = X_tfidf.toarray()

    # return X,feature_names
    return X_tfidf,tfidf_featurizer

# https://python.hotexamples.com/examples/sklearn.
# feature_extraction.text/TfidfVectorizer/get_feature_names/
# python-tfidfvectorizer-get_feature_names-method-examples.html
```

This function "plot_top_words" can be used to visualize the top words for each topic in a topic model and help interpret the results of the model.

- `k_topic`: number of topics
- `model`: our trained to vectorize
- `n_top_words`: top words to plot for each topic

```
In [111]: # Plotting the top words/topics
def plot_top_words(k_topic,model, feature_names, n_top_words,title=''):
    plt.clf()
    cols = 3
    rows = int(np.ceil(k_topic/cols))
    fig, axes = plt.subplots(rows, cols, figsize=(4 * cols, 4 * rows), sharex=True)
    axes = axes.flatten()

    # Looping to iterate over the topics in a trained topic
    # model and creates a horizontal bar chart for each topic.
    for topic_idx, topic in enumerate(model.components_):
        top_features_ind = topic.argsort()[:-1][:-n_top_words]
        top_features = [feature_names[i] for i in top_features_ind]
        weights = topic[top_features_ind]
        ax = axes[topic_idx]
        ax.bartop_features, weights, height=0.7)
        ax.set_title(f'Topic {topic_idx + 1}', fontdict={'fontsize': 15})
        ax.invert_yaxis()
```

```

ax.tick_params(axis='both', which='major', labelsize=20)

# Formatting and display the plot created in the plot_top_function

for i in 'top right left'.split():
    ax.spines[i].set_visible(False)
fig.suptitle(title, fontsize=15)
plt.subplots_adjust(top=0.90, bottom=0.05, wspace=0.90, hspace=0.3)
fig.tight_layout()
# fig.savefig(f'figure\{fname}.png')

plt.show()

# https://github.com/scikit-learn/scikit-learn/blob/main/examples/
# applications/plot_topics_extraction_with_nmf_lda.py

```

wordcloud_topics creating a wordcloud for each topic in the model by first iterating over the topics in the model. For each topic, it calculates the top no_top_words features based on the topic model coefficients and builds a dictionary that maps each feature to its respective weight.

```

In [112... # Wordcloud function
def wordcloud_topics(model, features, no_top_words=50):
    for topic, words in enumerate(model.components_):
        size = {}
        largest = words.argsort()[-no_top_words:] # invert sort order
        for i in range(0, no_top_words):
            size[features[largest[i]]] = abs(words[largest[i]])
        wc = WordCloud(background_color="white", stopwords=stopwords,
                        max_words=100,
                        width=960, height=540)
        wc.generate_from_frequencies(size)
        plt.figure(figsize=(7,7))
        #
        # print(f'TOPIC - {topic}')
        plt.imshow(wc, interpolation='bilinear')
        plt.title(f'TOPIC - {topic}: top {no_top_words} words')

```

This function can be used to perform topic modeling on a collection of text documents and visualize the results in a meaningful way.

```

In [113... # Topic modeling function
def get_topic_plot(data):
    docs = [] # Initializing list
    for tweet in tqdm(data): # loop through tweets
        docs.append(tweet) # add to the list

    # Merge the words into a single string
    vocab_size = len(set(" ".join(docs).split(" ")))

    # Store outcome of the analysis function
    X_tfidf, tfidf_vectorizer = analysis(docs, vocab_size)

    X = X_tfidf.toarray() # Converting into a matrix for the model

    # Storing a list of the most important words for each topic.
    feature_names = tfidf_vectorizer.get_feature_names_out()

    # Converting into tokens
    tokens = [text.split() for text in docs]
    # Storing using class to map words to integers ids
    id2word = Dictionary(tokens)
    # Storing into corpus using method to generate no. of times word appears
    corpus = [id2word.doc2bow(text) for text in tokens]

    coh = []
    for i in tqdm(range(5,30)):
        model = LdaModel(corpus, i, id2word, random_state=42)
        cm = CoherenceModel(model=model, corpus=corpus, coherence='u_mass')
        coherence = cm.get_coherence() # get coherence value
        coh.append(coherence)

    plt.plot(range(5,30),coh)
    plt.title('Coherence Score Vs Number of Topics (k)')
    plt.xlabel('Number of Topics (k)')
    plt.ylabel('Coherence Score')

```

```

plt.show()

k_topic = np.array(coh).argmax() + 5

print('No of Topic Selected by Coherence Score:', k_topic)
# LDA Model
lda_model = LatentDirichletAllocation(n_components=k_topic, random_state=42)
lda_model.fit(X)
plot_top_words(k_topic, lda_model, feature_names, 10)
wordcloud_topics(lda_model, feature_names)

```

In [132...]

```

# import locale
# locale.getpreferredencoding = lambda: "UTF-8"

```

We built a function arguments passing x_test and y_test, converting the test data into padded sequences using tokenizer and sequence length.

The function then generates predictions on the x_test which is the independent variables taken from the US airline data set - using a pre-trained LSTM model.

The predicted scores are thresholded at 0.5 to obtain predictions (0 = negative 1 = positive). We obtain the results using a confusion matrix and classification report. Finally, the function returns the binary predictions as a list for positive and negative tweets.

In [115...]

```

# Building function to predict
def get_sentiment_prediction(x_test,y_test):
    x_test = pad_sequences(tokenizer.texts_to_sequences(x_test), maxlen=SEQUENCE_L
    # Converting into list
    y_test_1d = list(y_test)
    # Generating predictions on the x_test
    scores = keras_model.predict(x_test, verbose=1, batch_size=50)

    # Setting up hreshold
    y_pred_1d = [0 if score[0]<0.5 else 1 for score in scores]

    # Producing Results
    cm = confusion_matrix(y_test_1d,y_pred_1d)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
    print(classification_report(y_test_1d,y_pred_1d))

    # Plotting
    disp.plot()
    plt.show()

    return y_pred_1d

```

Deploying our pre-trained LTSM Keras model and then converting the predictions into a binary format.

In [116...]

```

# Setting up variables
x_test_al = df_airline['clean_tweet']
y_test_al = df_airline['airline_sentiment'] # Target variable

```

In [117...]

```

# Replacing pos with 1 and neg with 0.
y_test_al = y_test_al.replace(['positive','negative'],[1,0])

```

In [118...]

```

# Checking that sentiments classes are balanced
df_airline['airline_sentiment'].value_counts()

```

Out[118...]

```

positive    2363
negative    2363
Name: airline_sentiment, dtype: int64

```

In [119...]

```

# Viewing scores: classification report and classification report
y_pred_al = get_sentiment_prediction(x_test_al,y_test_al)

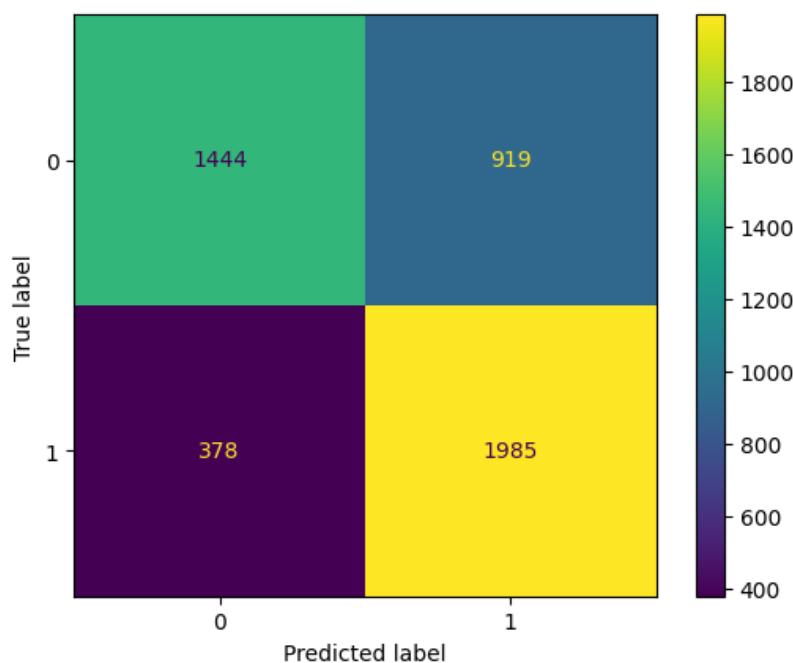
```

```

95/95 [=====] - 8s 80ms/step
      precision    recall   f1-score   support

```

0	0.79	0.61	0.69	2363
1	0.68	0.84	0.75	2363
accuracy			0.73	4726
macro avg	0.74	0.73	0.72	4726
weighted avg	0.74	0.73	0.72	4726



```
In [120... # Storing the predicted values in a new column
      df_airline['pred_sentiment'] = y_pred_al

In [121... # Check unique values
      df_airline['airline'].unique()

Out[121... array(['Virgin America', 'United', 'Southwest', 'Delta', 'US Airways',
      'American'], dtype=object)

In [122... # Filtering only for US Airways airline to check for negative tweets
      df_us_al = df_airline[df_airline["airline"].str.contains('US Airways', case=False)

In [123... # Checking filter
      df_us_al['airline'].unique()

Out[123... array(['US Airways'], dtype=object)

In [124... # Filtering negative tweets to obtain topic
      df_us_al_neg = df_us_al[df_us_al['pred_sentiment']==0]['clean_tweet']

In [125... # Checking the number of negative tweets
      df_us_al_neg.shape

Out[125... (408,)

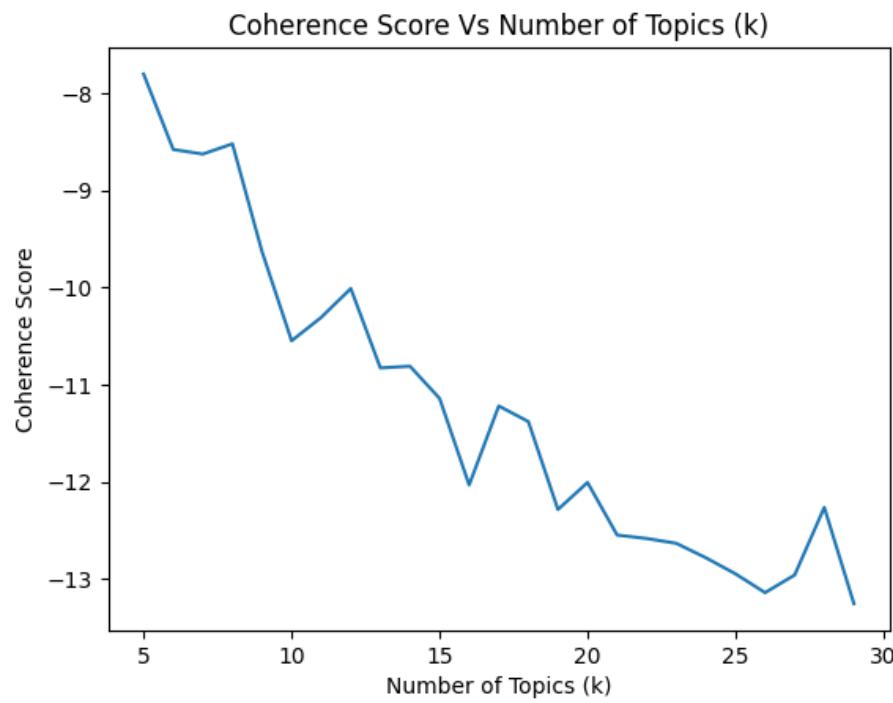
In [126... # View negative tweets
      df_us_al_neg

Out[126... 8985    well miss gate agents rebooked boarding pass w...
9024    great job today team challenging weather delay...
9105                      waiting flight right thanks
9222    worries flight attendant took care
9244    good work flight 1798 crew chairman recognitio...
```

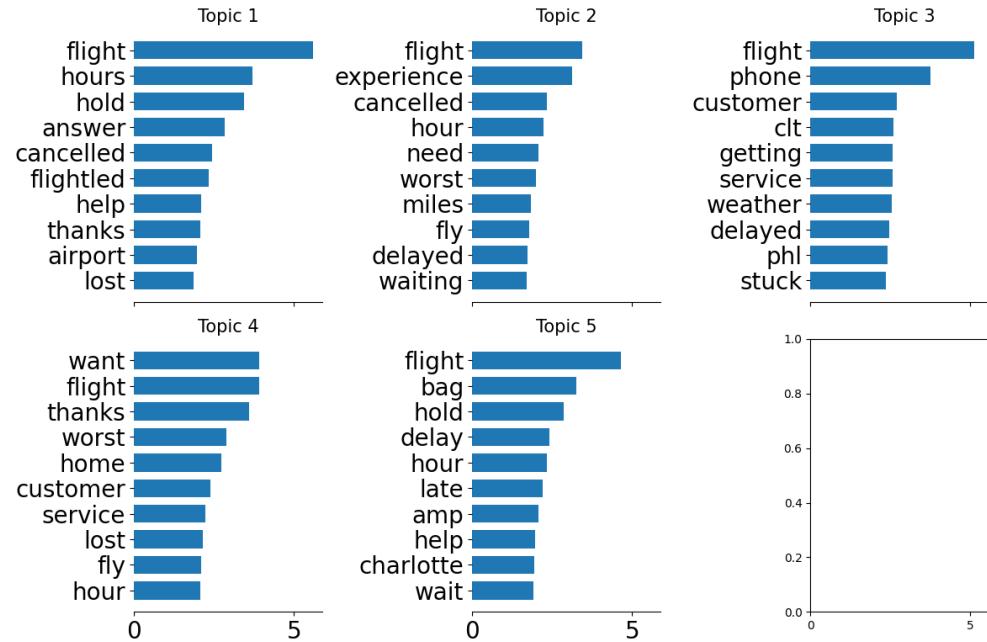
```
10271 tough night two 90 minute calls hold delayed p...
9875 5 hours phone waiting call back find luggage s...
10003 flight 838 leaving 6 hour mechanical delay nee...
11674 flt 5302 clt day supposed depart 5 51 6 20 st...
9786 two hours hold talk agent disconnects trying t...
Name: clean_tweet, Length: 408, dtype: object
```

We are going to apply the LDA model to analyze which topics were discussed the most among the negative tweets concerning US airlines.

```
In [127]: get_topic_plot(df_us_al_neg)
```



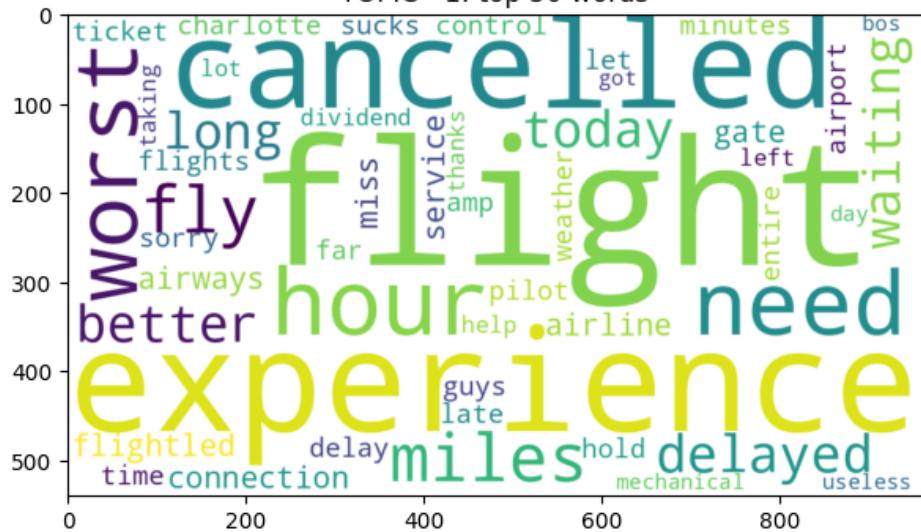
No of Topic Selected by Coherence Score: 5
<Figure size 640x480 with 0 Axes>



TOPIC - 0: top 50 words

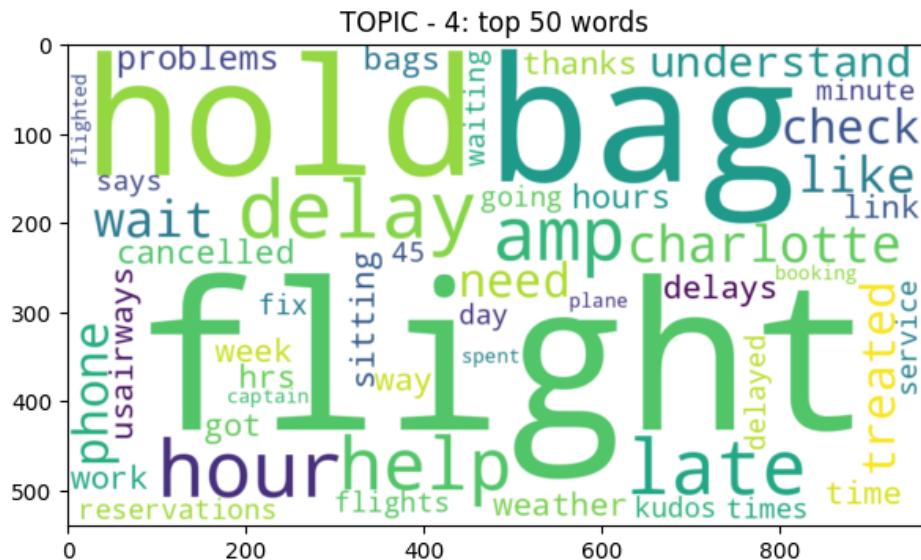
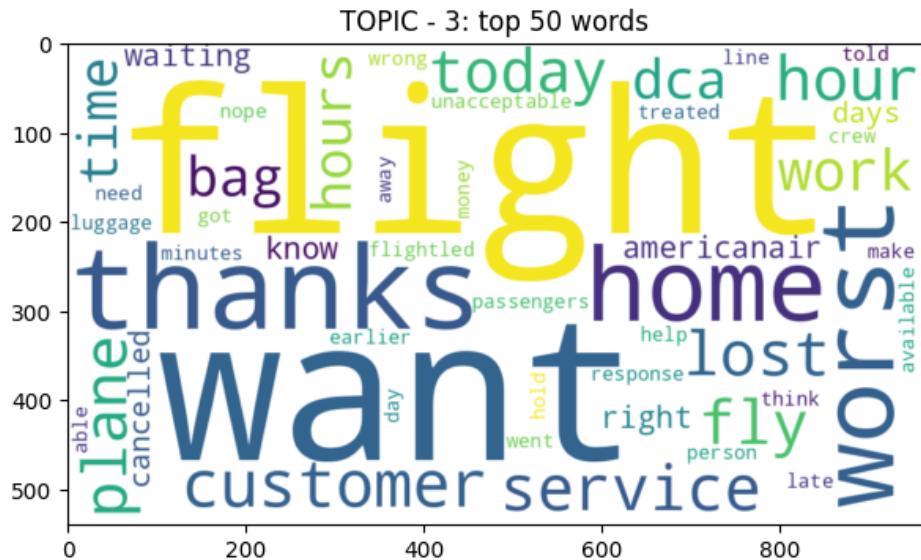


TOPIC - 1: top 50 words



TOPIC - 2: top 50 words





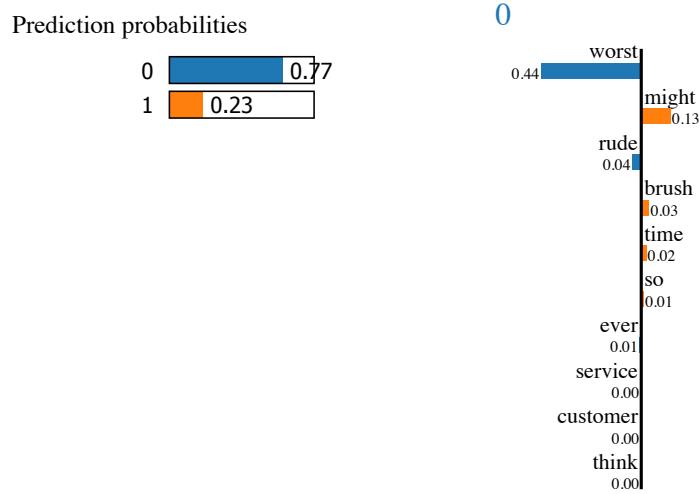
Using our model's tweet predictions, we filtered US Airlines' negative feedback and identified the main topics. Among them were customer service and flight delays. Based on these tweets, it seems that customers are dissatisfied with US Airlines' customer service and frequent delays. This valuable feedback can be used to address these issues.

LIME

LIME interprets the model in a simpler way. This allows us to gain insights into the contribution of each feature to the prediction, making it easier to understand the model's behavior.

```
In [135...]: # Use LIME to test a tweet  
exp = explainer.explain_instance("might possibly worst airline ever think time b  
predict_proba").show_in_notebook(text=True)
```

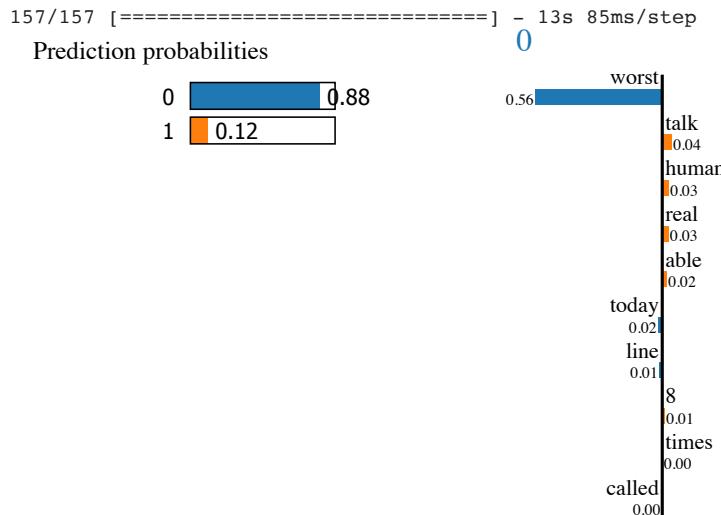
157/157 [=====] - 14s 87ms/step



Text with highlighted words

might possibly worst airline ever think time brush customer service so rude.

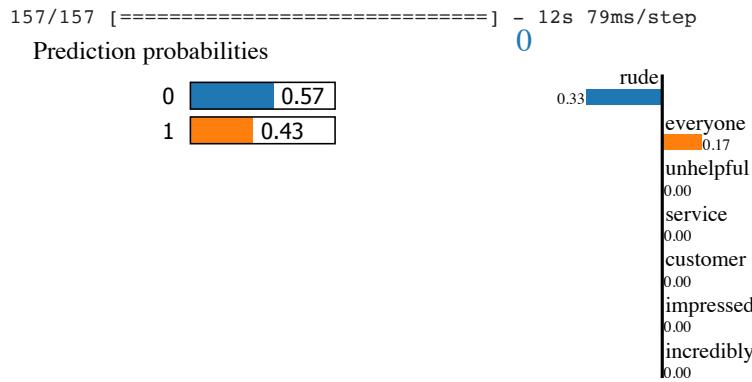
```
In [136]: # Use LIME to test a tweet  
exp = explainer.explain_instance("worst customer service line called 8 times tod
```



Text with highlighted words

worst customer service line called 8 times today able talk real human

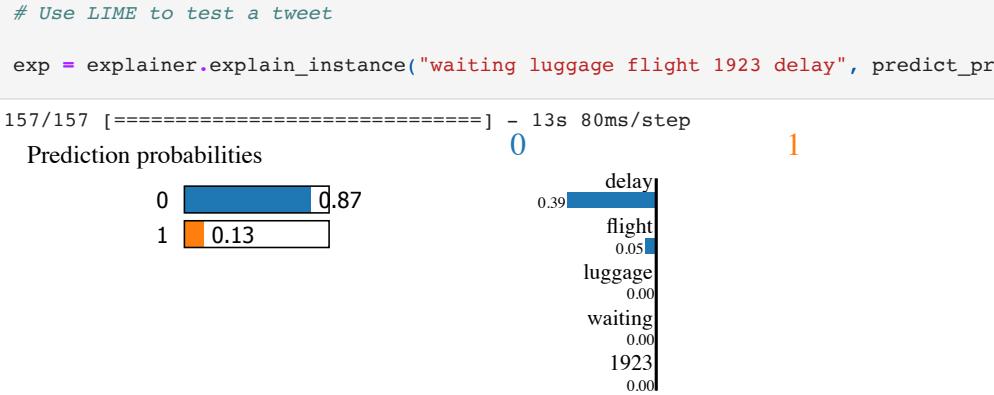
```
In [137...]: # Use LIME to test a tweet  
exp = explainer.explain_instance("impressed customer service everyone unhelpful")
```



Text with highlighted words

impressed customer service everyone unhelpful incredibly rude

In [138...]



Text with highlighted words

waiting luggage flight 1923 **delay**

Conclusion

In conclusion, we employed various models for sentiment analysis and found that the LSTM model outperformed the others. With training on one million tweets, the model achieved a prediction accuracy of 79%. To demonstrate its real-world application, we used the model to analyze airline feedback tweets, accurately predicting sentiments 71% of the time.

To gain further insights, we applied an LDA model for topic modeling, focusing specifically on negative tweets. The topics that emerged revealed that customers are predominantly dissatisfied with US Airlines' customer service and frequent delays. This valuable feedback can help airlines address these issues and make improvements in the areas that matter most to their customers. By continuously refining and applying such sentiment analysis models, businesses can gain actionable insights from customer feedback and enhance their products and services accordingly.