**Madhuri Dilliker**

**Ruid :191004207**

**Capstone Project Topic: News Article analysis using NLP and Sentimental Analysis**

**Table of Contents**

[Introduction 2](#_Toc39525906)

[Dataset 3](#_Toc39525907)

[Problem Statement 3](#_Toc39525908)

[Most common and uncommon words 5](#_Toc39525909)

[Methodology 6](#_Toc39525910)

[Feature Engineering 6](#_Toc39525911)

[**1.Text cleaning** 7](#_Toc39525912)

[**2.Text processing** 7](#_Toc39525913)

[**3.Text Vectorization** 8](#_Toc39525914)

[**4.Normalization** 8](#_Toc39525915)

[**5.N Gram Language model** 9](#_Toc39525916)

[**6.TF-IDF** 10](#_Toc39525917)

[Sentimental Analysis 12](#_Toc39525918)

[**Text Blob** 12](#_Toc39525919)

[**Polarity** 12](#_Toc39525920)

[**Subjectivity** 12](#_Toc39525921)

[Extracting Sentiment Score 13](#_Toc39525922)

[**Vader Sentiment Analyzer** 14](#_Toc39525923)

[**Sentiment Category** 14](#_Toc39525924)

[Model 15](#_Toc39525925)

[**Multinomial Naïve Bayes** 15](#_Toc39525926)

[**SVM:** 15](#_Toc39525927)

[**LSTM Modeling** 15](#_Toc39525928)

[Result 16](#_Toc39525929)

[Conclusion 18](#_Toc39525930)

[Future Works 18](#_Toc39525931)

[References 18](#_Toc39525932)

[Appendices 19](#_Toc39525933)

# **Introduction**

All around the world there can be events that are both good as well as bad, and we are aware of only those which we see or read on social media platform. Media has played a very important contribution in spreading these to majority people, its media’s primary responsibly to make people aware of what is happening around the globe. As bigger responsibility this sounds, media houses is the way in which they express the content of events happening to the people. Media house content should be unbiased, original and free of exaggeration as the same story can be interpreted in many different ways by its readers. It is very well known to us that, we become who we are by what we read and what we say.

If a person reads a positive story which is filled with positive words makes that person feel positive and vise versa. It is clear that words used in the content plays an important role as that of original content. Humans are able to classify any content or article into positive or negative subconsciously. For example “That women is very generous”, will give us a positive sentiment. We can surely say this as words used in this sentence are positive, this results in overall sentiment to be positive.

We can now use sentimental analysis to predict what would a person feel or think about particular news article. Term sentimental analysis calculates the sentiment score along with the combined power of natural language processing and text analysis to classify if the sentiment is ‘positive’, ‘neutral’ or ‘negative’. Sentiment analysis will combine the human language semantics and symbolic representation, this will be input for the algorithm, algorithm will be trained with news text to predict if the input news text sentiment is positive or negative or neutral. This is supervised learning approach where we will train the model and predict its sentiment class.

For prediction and classification, we make use of machine learning process and Natural Language processing technique to understand how the text data is and characteristics of this text data and classify the sentiment. We need a computational model that can classify the test into sentiment category. With increasing in information and development of online news forum, we need a effective tool to accurately classify this information into category. In this way we could easily filter, search and store useful information. For news agencies this will be a accurate tool as they receive lot of articles in a day.

With emerging demand in the field of digital news, it is observed that there are vast developments in deep learning for Natural Language processing. In current times deep learning model can be used for NLP task which involves machine translation, speech recognition, sentiment analysis and classification problems. We need a system for online news which would associate with problems faced by journalism to classify if they are spreading a news which has a negative impact to the world.

The scope of this project is to use the new article data which is in the form of raw textual data format and interpret if the news represent a positive and negative sentiment.

# **Dataset**

**Dataset**: News Articles

**Data source**: Kaggle

**Description**: This dataset has data collected from various media houses home page to see which News media shares/writes articles with less gory words. This dataset stands as sample to find out which media house conveys the NEWS in more optimistic way. Data is collected from multiple news websites from October 2017 to Nov 2017. The dataset consist of 155 null values and 1100 rows. The dataset has following attributes:

1. **Title** – Title of the news article

2. **Summary** – Gist of the article

3. **Text** – Full text inside an article

4. **Url** – url of the article

5. **Keywords** – important words used in the article

In this Kaggle data source we have dataset for all the popular house media. We will move ahead with cnn dataset for this project.

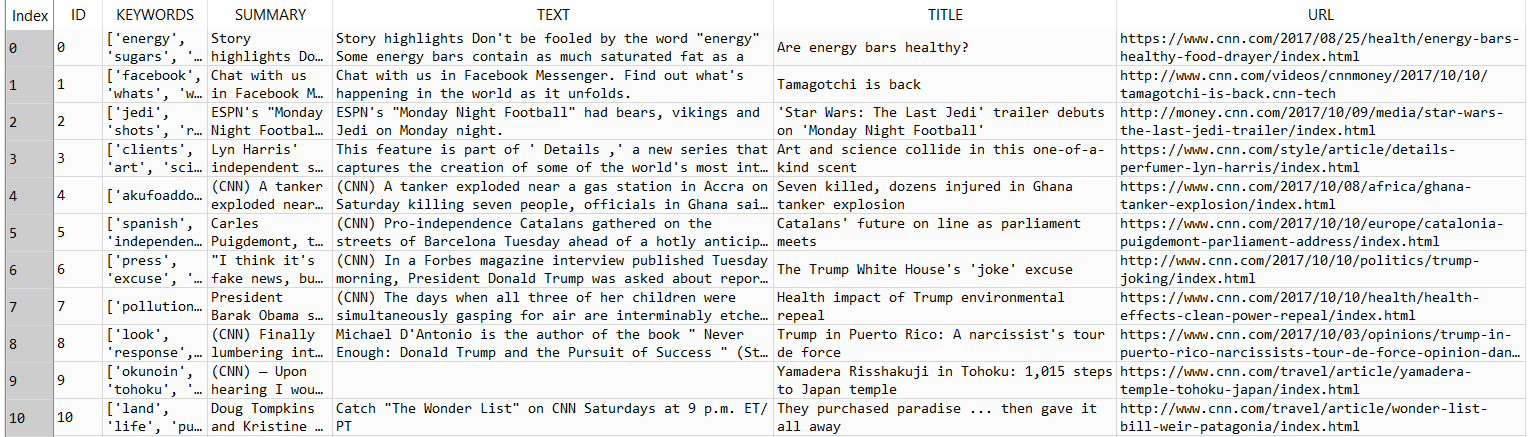


Figure 1:Sample Data

# **Problem Statement**

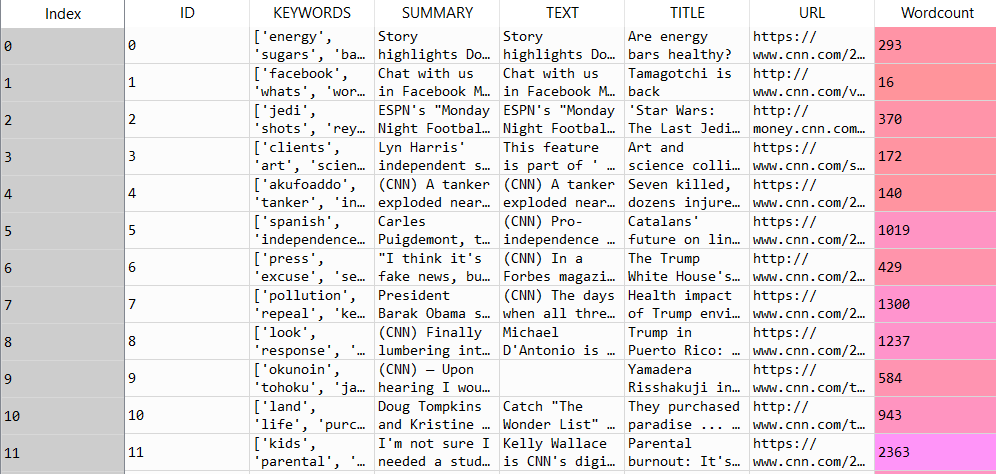
In this digital world people tend to consume news from social media, online news paper, online media house, popular search engines. And it becomes difficult to identify if the content behind the news. To classify if news article has positive or negative sentiment. Sentimental scores are extracted for each row in the dataset and classified if it belong to “positive”, “negative” or “neutral” sentiment.

**Exploratory Data Analysis**

**Word Count**

Gaining some insights from the new articles data. For now we just have our new text data so let’s find the word count for each news article and news length distribution for each article.

We can see that mean of the distribution for word count lies near 800 words per abstract. The minimum and maximum word count ranges from 2 to 6000. This gives us insight into word count range for the data set we are handling and how it is varied for each rows. To have number of word count for each row, we can create a new column “Wordcount” using lambda function which will describe word count for each row.



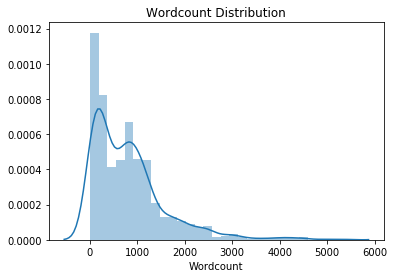
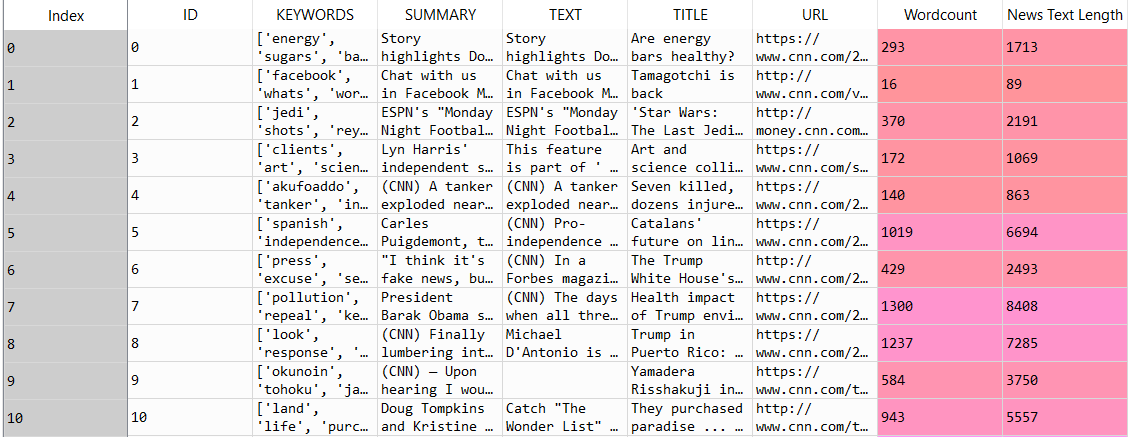


Figure 2: Word count distribution

**News Length**

For news length distribution the mean length lies around 5000 characters for this CNN dataset. The minimum and maximum text character ranges from 6 to 33000. This gives us insight into text character count range for the data set we are handling and how it is varied for each rows. To have text length for each row, we can create a new column “News Length” using lambda function which will describe text length for each row.



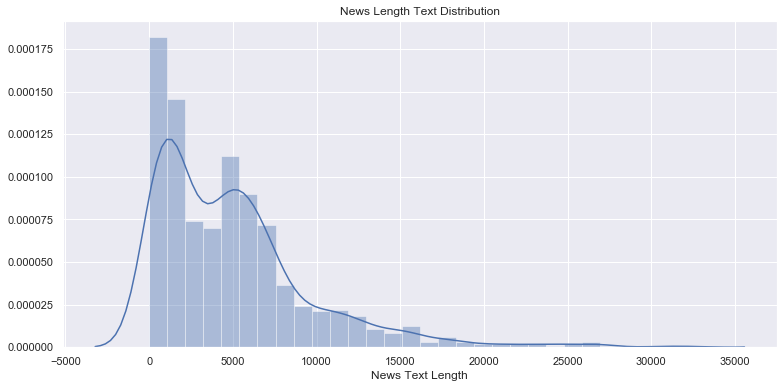
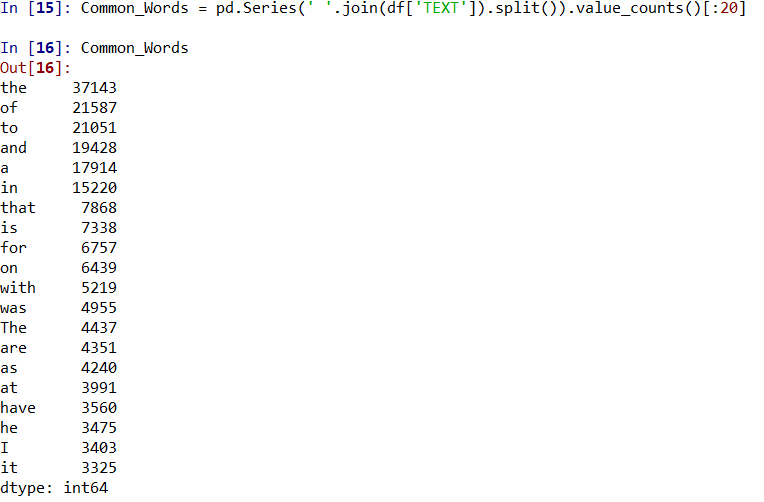


Figure 3. News Length Text Distribution

# **Most common and uncommon words**

Getting the list of common and uncommon words will give us the words which are used frequently which can be later added to custom list of stop words. Following is the list of common words with their value count, we can use this list later to add these frequently used words in custom stop word list



# **Methodology**

**Phase 1**: Cleaning the data set

**Phase 2:** Use rules of NLP and NTLK for text vectorization for preparing text data for classification and text pre processing

**Phase 3:** Extracting Sentiment Scores

**Phase 4:** Use the effective classification technique such as Naïve Bayes, SVM and LSTM model.

**Phase 5:** Check the classification accuracy and represent the result graphically

# **Feature Engineering**

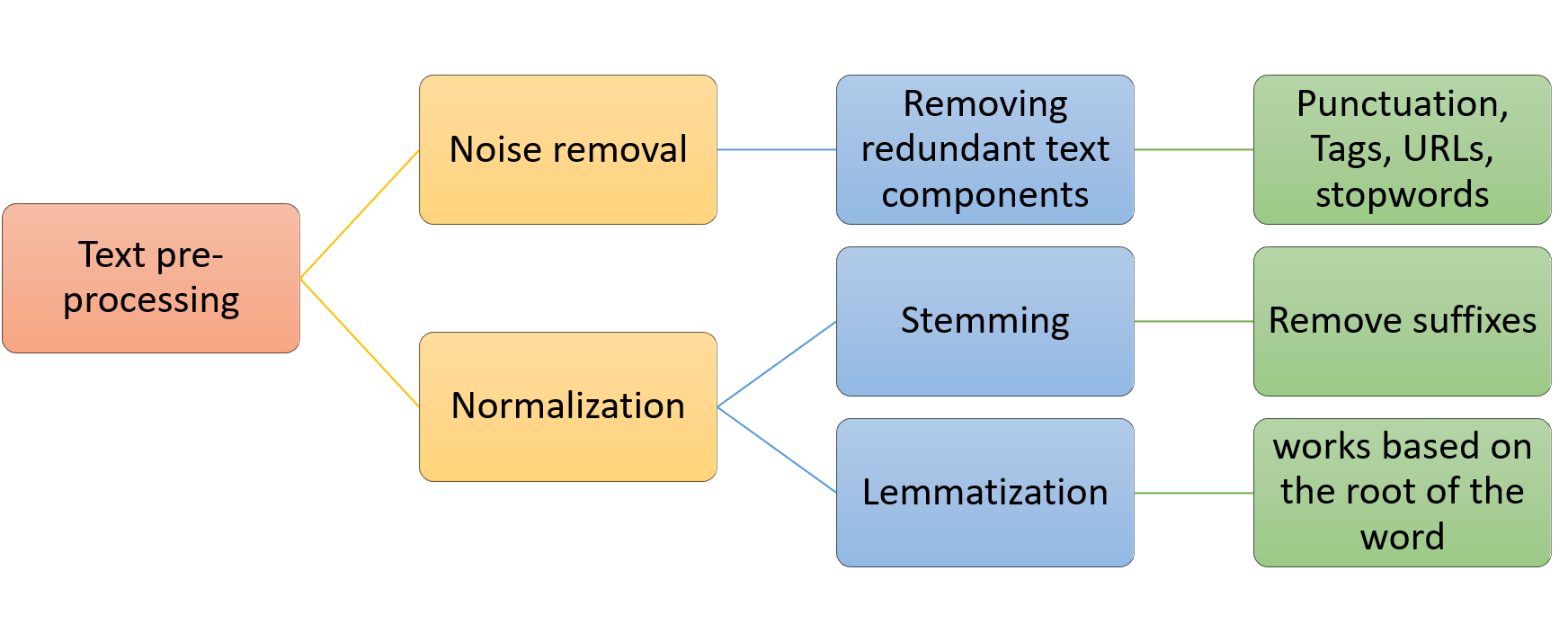


Figure 4: Text pre-processing

Here we will transform our raw news data into feature which can be used as input for classification models.

## **1.Text cleaning**

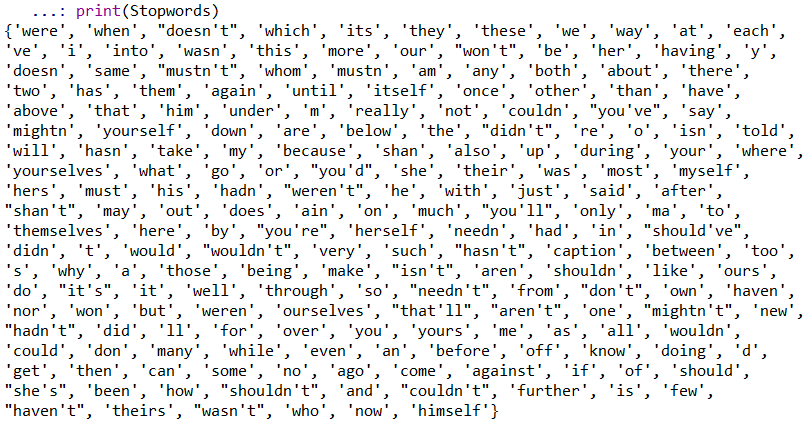
Following steps are used to perform cleaning of data

* Removing special character and tags: Tags like ‘&lt’ and special characters such as “\\d” which do not add value to text data are needed to be removed from the text.
* Change text to lower case: Example the word price with P in caps and one with all lower case “price” and “Price” all will be treated as different entity, so as to treat these words as equal converting the text to lower case would solve this problem.
* Punctuation signs: Characters in the text which includes ‘?’, ‘!’ or ‘,’should be removed
* Possessive Pronouns: The words with apostrophe s i.e ‘John’ or ‘John’s’ should be treated as same so we need to remove this from the given text

## **2.Text processing**

**NTKL**: NTKL is Natural Language Toolkit is text processing library which is used in building python programs which involves work with human language data for Natural language processing.

**Stopwords:** A stop word is list of frequently used words which can be ignored to derive better results. NTKL package gives downloaded list of common words. We can modify this list and add some new words which are common in the news text and delete it from the text data. Stop words is mostly referred as noise data which should be eliminated to generate better results



**Word Cloud**

Word cloud is another way to which represents frequency of a word in given text. If the size of word is larger this means the word is commonly used in the given text or represents importance of each word. Below is the word cloud for our text column in dataset.



Figure 5: Word Cloud

## **3.Text Vectorization**

When words are converted to numbers this process is called vectorization. Word vectorization is a method which maps words from vocabulary to corresponding vectors of real numbers which can later be used for predictions or word semantics. We convert group of sentences into tokens, this is also called process of segmentation. We split the data into smallest possible chunk. Each word is converted to its numerical vector format. In simple words tokenization process will break the stream of text in meaningful entities called tokens. Tokenization can be performed on paragraph to break it into meaningful sentences and sentence can be further broken down to words. We are using countVectorizer function to map each word to its feature, which can later be transformed into a sparse matrix

## **4.Normalization**

Normalization is the process of converting the list of words to uniform sequence. This will aid in processing the text for training. This is achieved using stemming and lemmatization . Stemming and lemmatization is normalization of words in the text data, i.e. reducing word to it original root form.

**Stemming**: Stemming is the process of reducing derived words to their word stem, root or root form. Words which are spelled differently because of their tense can be deduced to one central word, if they have the same meaning. For example, cook, cooking and cooked will be deduced to one word ‘cook’. Stemming is applied to single word .**Lemmatization** is the process of reducing a word to its lemma ie resolving words to their dictionary form and it requires to know the part of speech for the word. Lemmatization needs to know the structure of language

## **5.N Gram Language model**

N gram is sequence of words or N tokens. N gram language model can predict probability of input N-gram with any sequence of words. We use this language model to predict the probability of seeing word x, if we know which all words are used previously ie y where previous words contain n-1 list of words. Before moving forward to use Term frequency and Inverse document frequency for our text column. Lets visualize unigram, bigram and trigram for our text dataset Using count vectorization.

**Unigram**

Sequence of that contains one word at a time. Following is the visualization for top 20 words in our text data set.

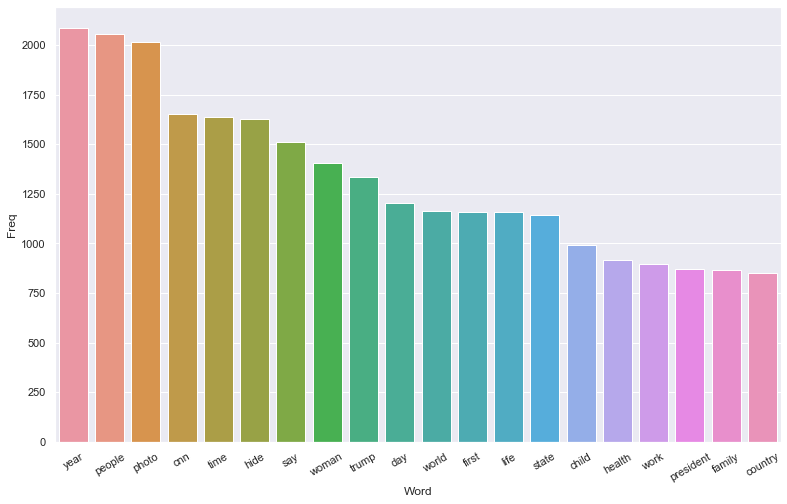


Figure 6: Unigram Visualization

**Bigram**

Bigram is when sequence contains 2 words at a time, following is bar plot visualization of bigram for our data set

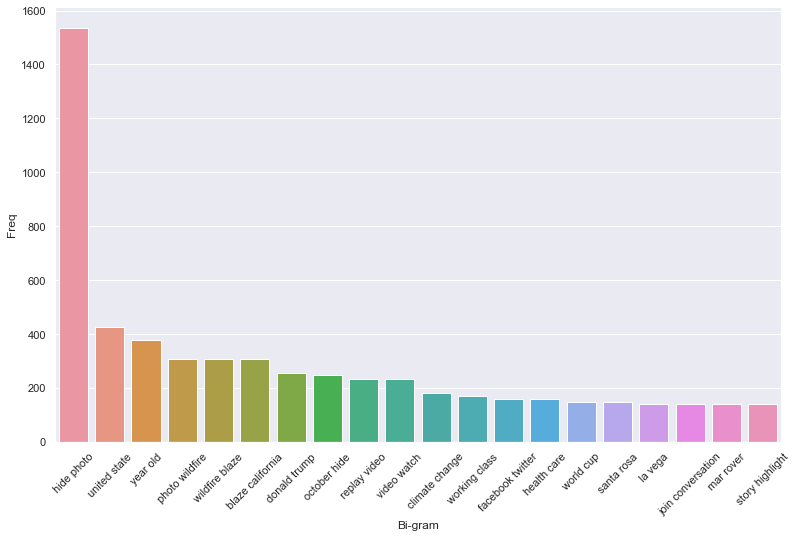


Figure 7- Bi-gram Visualization

**Trigram:** Tri-gram is when sequence contains 3 words at a time, following is bar plot visualization of bigram for our data set

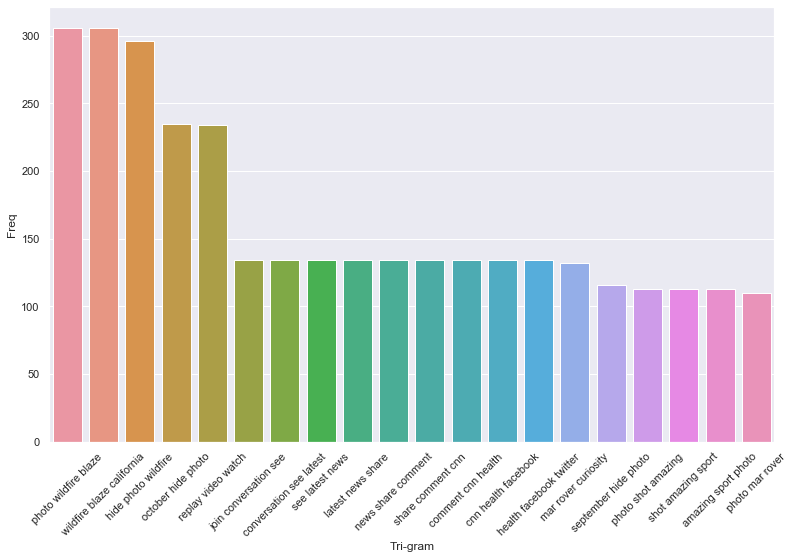
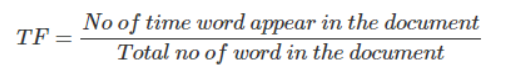


Figure 8: Trigram Visualization

## **6.TF-IDF**

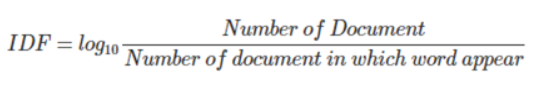
**Term- Frequency:**

Term Frequency can be defined as how frequently a particular word appears in the document. In a document each sentence will be of different length and it is a least possibility that a sentence would be of same length, so there is a slight possibility a word will appear in long sentence more than one time as compared to sorter sentences. Formula to calculate term frequency is as follows:



**Inverse Document Frequency:**

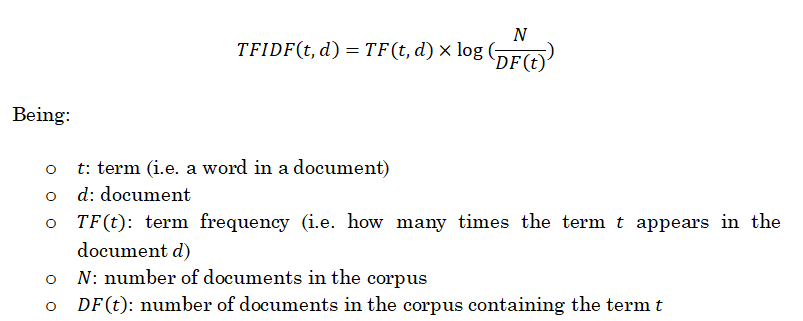
This is another way of finding the importance of a word. It is based on the fact that less frequent used word are more important and informative than common words in the document. IDS formula is given below:



**TF-IDF**

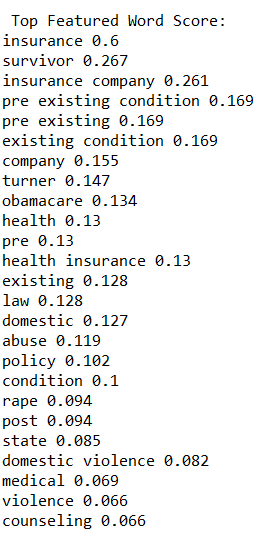
When we multiply the values of Term Frequency (TF) and Inverse Document Frequency (IDF) we get TF-IDF score.TF-IDF score will represent relative importance of a term/word in the entire document. TF-IDF will reduce value of the common word that are used in text document.

TF- is term frequency which will represent frequency of a term in given document and IDF – is Inverse Document Frequent. The formula for TF-IDF and algorithm is given below



TF-IDF value will increase proportionally with number of time a particular word appears in the text document and is offset by the number of document in the corpus that contain the word, which would help in adjusting to the fact that some words appear more frequently generally. TF-IDF feature creation process is fast and we can avoided overfitting by fine tuning the features extracted using this process.

**Top Feature score extracted from our text column**



# **Sentimental Analysis**

## **Text Blob**

Text blob in python is a library for processing textual data. It provides a simple interface for dividing into the natural language processing task like classification, sentiment analysis. **Textblob,sentiments** function will calculate polarity and sentiment for given textual data.We can also classify sentiment into positive and negative sentiment using **NaiveBayesAnalyzer** which will classify whole text data into positive and negative score.

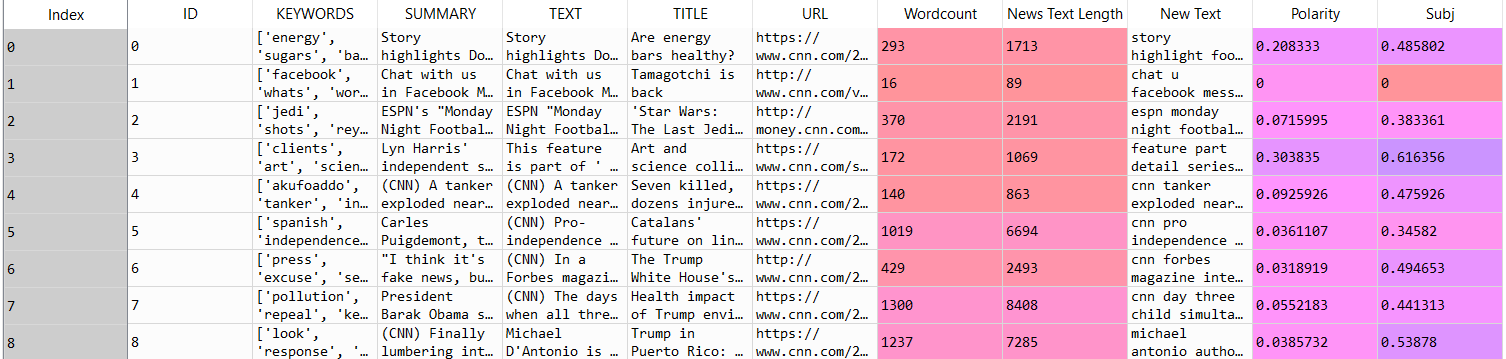
## **Polarity**

When the floating point sentiment value lies within the range of -1.0 to 1.0 where 0 represents neutral sentiment, +1 represents more positive sentiment and -1 represents more negative sentiment.

## **Subjectivity**

When a floating point sentiment value lies within range of 0.0 to 1.0 where 0.0 tends to more objective and 1.0 tends to more subjective. Objective sentence represent sentence are factual and subjective sentence refer to emotion, feelings, believe and views.

We have calculated subjectivity and polarity for each row in our dataset.



Below is the sentiment score for our overall text data set



Below is the subjectivity and polarity line graph for our overall text data

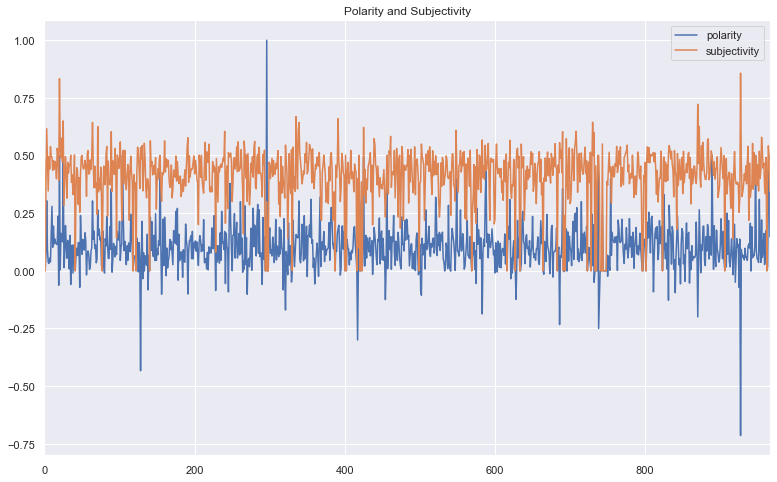


Figure 9: Sentiment line plot

# **Extracting Sentiment Score**

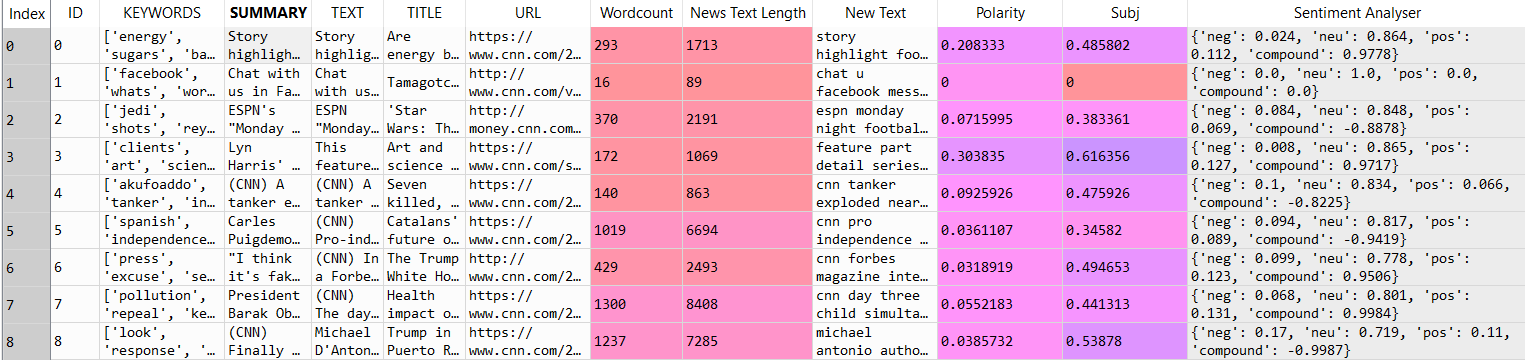
In order to perform sentiment analysis we do not have sentiment score in our dataset. We need a score so that we can apply classification model to our text dataset, to classify if the news text represent positive or negative news.

## **Vader Sentiment Analyzer**

Vander stands for valence aware dictionary and sentiment reasoner. It is a sentiment analysis library is built in NLTK which is lexicon and rule based which is specifically designed for sentiments expressed in online or social media. It uses a combination of sentiment lexicon, which is a special list of lexical feature which are labelled as positive, neutral or negative score. **polarity \_scores** method is used to calculate the sentiment score. All these score should be add up to 1. Vander generates positive and negative scores and tell us how positive and negative a sentiment is. This library is fast enough and doesn’t suffer from speed performance. It not only gives the scores for positive and negative but also proves a compound score, **compound** score is calculates sum of all lexicon ratings and is normalized between -1(extremely negative) and 1(extremely positive).

* When compound score is greater than **0.05** then sentiment is **positive.**
* When compound score lies between the range which is **greater than -0.05 and less than 0.05** then sentiment is **neutral.**
* When compound score is less than **-0.05** then sentiment is **negative**

Below is the snippet of our data set after applying **polarity \_scores** method in vander sentiment analyzer library



## **Sentiment Category**

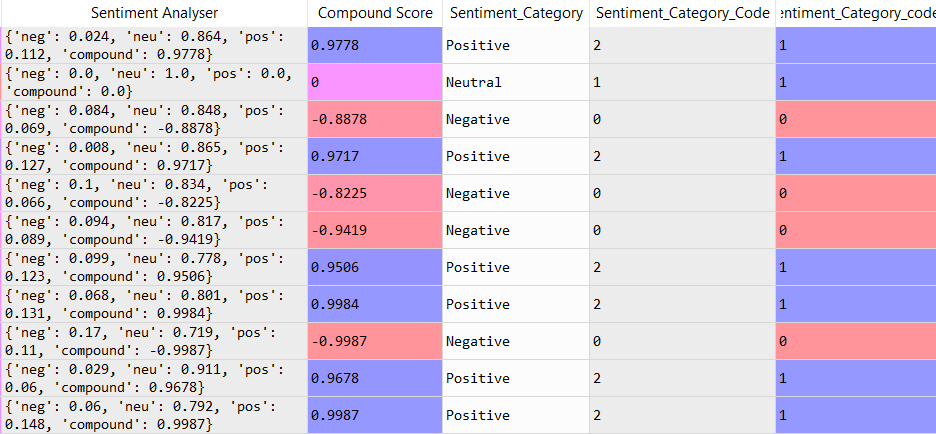
Now we have compound score we can follow following steps to extract the sentiment category from this score

1. Extract only **compound score** into a new column in pandas dataframe
2. Classifying the score as positive negative and neutral in new column called ‘Sentiment category’
3. Converting column type to category type
4. Adding a column to have numerical columns to category
5. Treating neutral type as positive category

Following is the snippet of dataset were we have a column which indicate sentiment category

We now have following new columns in our dataframe.

* Compound score
* Sentiment category
* Sentiment Category code
* Sentiment Category code 1 (neutral is treated as positive)



# **Model**

## **Multinomial Naïve Bayes**

Naïve Bayes belongs to family of algorithms based on application of bayes theorem with a strong assumption, that every feature is independent of others, and to predict the category of a given input sample. This algorithm is probability based classifier which will calculate the Bayes theorem and category which represents the highest will be the output of classification. Naïve Bayes are commonly used in NLP problems. This algorithm is considered to be fast, reliable and accurate in NLP applications

## **SVM:**

It works by finding a hyperplane that separates the two classes with maximum margin. SVM algorithm for text classification problem can obtain into good results as its very effective in high dimension space

## **LSTM Modeling**

* Vectorizing text column by converting each text into a vector or integer sequence.
* Limiting and setting max features to 30000 words
* Truncating and padding input sequence so the length of all the sequence is same before they are processed
* The activation function layer used is soft max and model loss function is compiled using categorical crossentropy as it is multi class classification problem
* The output layer will create 2 output values one for each class.

# **Result**

|  |  |
| --- | --- |
| **Model** | **Accuracy** |
| Multinomial Naïve Bayes | 82 |
| Random Forest | 75 |
| Svm | 66 |
| LSTM | 97 |

We can see from above evaluation Multinomial Naïve Bayes and LSTM could classify news article text in with better accuracy as compared to SVM and Random Forest.

With these result, we can construct execution measurements that are valuable for a speedy evaluation on how well a classifier functions:

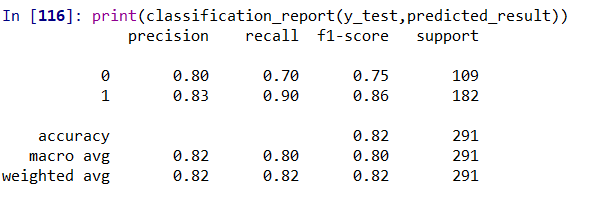
• **Accuracy**: gives level of writings that were anticipated with the right tag.

• **Precision**: gives level of models the classifier got directly out of the complete number of precedents that it anticipated for a given tag.

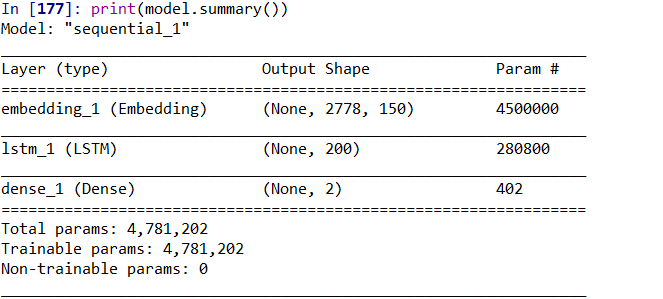
• **Recall**: gives level of precedents the classifier anticipated for a given tag out of the absolute number of models it ought to have anticipated for that given tag.

• **F1 Score**: gives the consonant mean of exactness and review

For Multinomial Naïve Bayes we have following classification report which indicates precision, recall and F1 score for positive (1) and negative sentiment. If we calculate the average of all these values it turns out to be around 0.82



We have divided our data into test and training split into 70/30 ratio. Below is the model summary for LSTM model.



Loss function is used in machine learning algorithm, to optimize it and loss is calculated on training and testing data. And this graph will provide the insight on how well model is performing in training and testing dataset. It reflects sum of errors made for each training and testing set. Loss value also indicates how well or bad model behaves after each iteration. The graph shows overfitting model during training as there is sudden rise in the start of training model and then value is decreased.

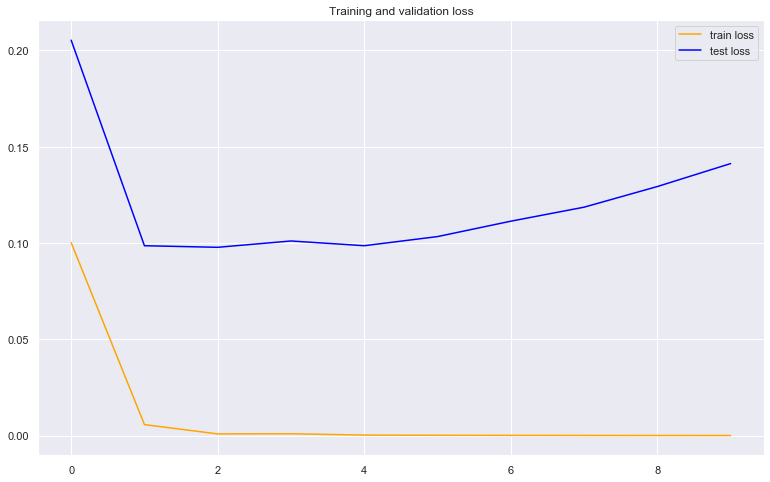


Figure 10: Training and Validation loss

An accuracy graph will provide the insight into how algorithm performs. The model accuracy is determined after model is trained it is measure of model prediction against its test or true data and calculated in percentage form. As we can see from the accuracy graph our accuracy is close to 97 percent

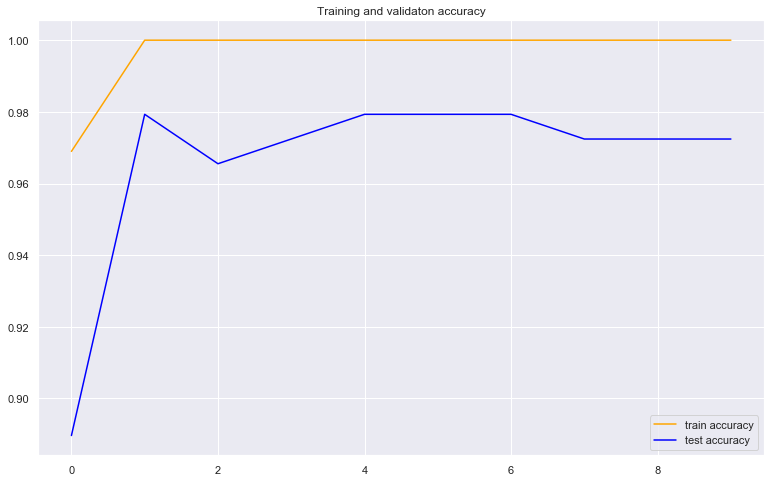


Figure 11: Training and validation Accuracy

# **Conclusion**

In this project we have used NLP text processing and vectorization models to pre process our text data so that we can use this text for classifying it into positive and negative sentiment. The model proposed is being tested for various performance measures which consist of accuracy, Precision, recall and F1 score. The accuracy ranges from 66 to 97 for all the training models. LSTM model with convoluted learning has outperformed all the other models

# **Future Works**

As we have seen the resultant curve for training loss and accuracy showed overfitting of data we can overcome this using cross validating algorithm. We can also implement hashing vectorization which would improve results drastically. We would perform further testing on datasets with larger values to see if we can further improve the results from this project.

# **References**

<http://www.uvm.edu/pdodds/files/papers/others/2007/godbole2007a.pdf>

<https://www.kaggle.com/harishcscode/all-news-articles-from-home-page-media-house>

# **Appendices**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Apr 1 13:09:37 2020

@author: madhuri

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import model\_selection, naive\_bayes, svm

#loading the dataset

df = pd.read\_csv('CNN\_news.csv')

#checking and cleaning the mising values

df.isnull().sum()

df= df.dropna()

#viewing the dataset

df.head(10)

df.columns

#Calculating word count for each row for text column

df['Wordcount'] = df['TEXT'].apply(lambda x: len(str(x).split(" ")))

df[['TEXT','Wordcount']].head()

#descriptive statistics for the word count and its visualization

df.Wordcount.describe()

plt.figure(figsize=(12.8,6))

sns.distplot(df['Wordcount']).set\_title('Wordcount Distribution');

#calculate news text length column and plot news text length

df['News Text Length'] = df['TEXT'].str.len()

plt.figure(figsize=(12.8,6))

sns.distplot(df['News Text Length']).set\_title('News Length Text Distribution');

df['News Text Length'].describe()

#find common words

Common\_Words = pd.Series(' '.join(df['TEXT']).split()).value\_counts()[:20]

Common\_Words

#find uncommom words

Uncommon\_words = pd.Series(' '.join(df ['TEXT']).split()).value\_counts()[-20:]

Uncommon\_words

# Libraries for text preprocessing

import re

import nltk

#nltk.download('stopwords')

from nltk.corpus import stopwords

#nltk.download('wordnet')

from nltk.stem.porter import PorterStemmer

from nltk.tokenize import RegexpTokenizer

from nltk.stem.wordnet import WordNetLemmatizer

#stop words

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

#adding more words to stop words

##Creating a list of custom stopwords

addNewStopWords = ['said','say','also','could','even','get','would','like','one','caption','may','much','go','make','come','take'

,'know','well','really','much','two','must','ago','new','many','say','way','told']

Stopwords = Stopwords.union(addNewStopWords)

print(Stopwords)

#remove possive pronoun

df['TEXT'] = df['TEXT'].str.replace("'s", "")

corpus = []

for i in range(0, 3847):

#Remove punctuations

text = re.sub('[^a-zA-Z]', ' ', df['TEXT'][i])

#Convert to lowercase

text = text.lower()

#remove tags

text=re.sub("&lt;/?.\*?&gt;"," &lt;&gt; ",text)

# remove special characters and digits

text=re.sub("(\\d|\\W)+"," ",text)

##Convert to list from string

text = text.split()

##Stemming

ps=PorterStemmer()

#Lemmatisation

lemmen = WordNetLemmatizer()

text = [lemmen.lemmatize(word) for word in text if not word in

Stopwords]

text = " ".join(text)

corpus.append(text)

#Generating Word cloud

#!pip install wordcloud

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

import matplotlib.pyplot as plt

wordcloud = WordCloud(

background\_color='white',

stopwords=Stopwords,

max\_words=100,

max\_font\_size=50,

random\_state=42

).generate(str(corpus))

print(wordcloud)

fig = plt.figure(1)

plt.imshow(wordcloud)

plt.axis('off')

plt.show()

fig.savefig("word1.png", dpi=900)

#adding changes in text to original data

df['New Text'] = corpus

#feature extraction

#count vectorization and transformation

from sklearn.feature\_extraction.text import CountVectorizer

import re

cv=CountVectorizer(max\_df=0.8,stop\_words=Stopwords, max\_features=10000, ngram\_range=(1,3))

X=cv.fit\_transform(corpus)

X.shape

list(cv.vocabulary\_.keys())[:10]

#visualiztion of words with top unigram, bigram and trigram

#frequently occuring words that is common in the text column

def Unigram\_n\_words(corpus, n=None):

vec1 = CountVectorizer().fit(corpus)

bag\_of\_words = vec1.transform(corpus)

sum\_words1 = bag\_of\_words.sum(axis=0)

words\_freqs = [(word, sum\_words1[0, idx]) for word, idx in

vec1.vocabulary\_.items()]

words\_freqs =sorted(words\_freqs, key = lambda x: x[1],

reverse=True)

return words\_freqs[:n]

#put common words in pandas data frame

words = Unigram\_n\_words(corpus, n=20)

df\_uni = pd.DataFrame(words)

df\_uni.columns=["Word", "Freq"]

#visulalize the bar plot using seaborn library - unigram grams

sns.set(rc={'figure.figsize':(13,8)})

g = sns.barplot(x="Word", y="Freq", data=df\_uni)

g.set\_xticklabels(g.get\_xticklabels(), rotation=30)

#visulalize the bar plot using seaborn library - bi grams

def bigram\_words(corpus, n=None):

vec2 = CountVectorizer(ngram\_range=(2,2),

max\_features=2000).fit(corpus)

bag\_of\_words = vec2.transform(corpus)

sum\_words1 = bag\_of\_words.sum(axis=0)

words\_freqs = [(word, sum\_words1[0, idx]) for word, idx in

vec2.vocabulary\_.items()]

words\_freqs =sorted(words\_freqs, key = lambda x: x[1],

reverse=True)

return words\_freqs[:n]

top\_bigram\_words = bigram\_words(corpus, n=20)

df\_bigram = pd.DataFrame(top\_bigram\_words)

df\_bigram.columns=["Bi-gram", "Freq"]

print(df\_bigram)

#Barplot of most freq Bi-grams

sns.set(rc={'figure.figsize':(13,8)})

h=sns.barplot(x="Bi-gram", y="Freq", data=df\_bigram)

h.set\_xticklabels(h.get\_xticklabels(), rotation=45)

#visulalize the bar plot using seaborn library - tri grams

def trigram\_words(corpus, n=None):

vec1 = CountVectorizer(ngram\_range=(3,3),

max\_features=2000).fit(corpus)

bag\_of\_words = vec1.transform(corpus)

sum\_words = bag\_of\_words.sum(axis=0)

words\_freqs = [(word, sum\_words[0, idx]) for word, idx in

vec1.vocabulary\_.items()]

words\_freqs =sorted(words\_freqs, key = lambda x: x[1],

reverse=True)

return words\_freqs[:n]

top3\_words = trigram\_words(corpus, n=20)

df\_trigram = pd.DataFrame(top3\_words)

df\_trigram.columns=["Tri-gram", "Freq"]

print(df\_trigram)

#Barplot of most freq Tri-grams

sns.set(rc={'figure.figsize':(13,8)})

j=sns.barplot(x="Tri-gram", y="Freq", data=df\_trigram)

j.set\_xticklabels(j.get\_xticklabels(), rotation=45)

#convert word matrix to integer

from sklearn.feature\_extraction.text import TfidfTransformer

cov\_tfidf\_transformer=TfidfTransformer(smooth\_idf=True,use\_idf=True)

cov\_tfidf\_transformer.fit(X)

# get feature names

feature\_names=cv.get\_feature\_names()

# fetch document for which keywords needs to be extracted

Text\_words=corpus[532]

#generate tf-idf for the given document

tf\_idf\_vector=cov\_tfidf\_transformer.transform(cv.transform([Text\_words]))

tf\_idf\_vector.shape

print(tf\_idf\_vector)

#based on higest td-idf score we can extract higest scores

#Function for sorting tf\_idf in descending order

from scipy.sparse import coo\_matrix

def sort\_coo\_matrix(coo\_matrix):

values = zip(coo\_matrix.col, coo\_matrix.data)

return sorted(values, key=lambda x: (x[1], x[0]), reverse=True)

def feature\_from\_vector(feature\_names, sorted\_items, topn=10):

#use only topn items from vector

sorted\_items = sorted\_items[:topn]

score\_vals = []

feature\_vals = []

# word index and corresponding tf-idf score

for idx, score in sorted\_items:

#keep track of feature name and its corresponding score

score\_vals.append(round(score, 3))

feature\_vals.append(feature\_names[idx])

#create a tuples of feature,score

#results = zip(feature\_vals,score\_vals)

results= {}

for idx in range(len(feature\_vals)):

results[feature\_vals[idx]]=score\_vals[idx]

return results

#sort the tf-idf vectors by descending order of scores

sorted\_items=sort\_coo\_matrix(tf\_idf\_vector.tocoo())

#extract only the top n; n here is 10

Feautured\_Words=feature\_from\_vector(feature\_names,sorted\_items,25)

# now print the features with their probalility

print("\nText:")

print(doc)

print("\n Featured Word Score:")

for f in Feautured\_Words:

print(f,Feautured\_Words[f])

#find higest weight of particular word

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vector = TfidfVectorizer()

tfidf\_vector.fit(df['New Text'])

X\_transformation = tfidf\_vector.transform(df['New Text'])

#text at loc[1]

df['New Text'][1]

#find importance of words using vectorization method

print([X\_transformation[1,tfidf\_vector.vocabulary\_['find']]])

#check subjectivity and polarity of news text

from textblob import TextBlob

from textblob.sentiments import NaiveBayesAnalyzer

polarity=[]

subjectivity=[]

df['Polarity']=df.TEXT.apply(lambda x: TextBlob(x).sentiment.polarity)

df['Subj']=df.TEXT.apply(lambda x: TextBlob(x).sentiment.subjectivity)

df.head(10)

#Create 2 arrays

polarity=[]

subj=[]

#Get polarity and sentiment for each row and put it in either polarity or sentiment

for t in df.TEXT:

tx=TextBlob(t)

polarity.append(tx.sentiment.polarity)

subj.append(tx.sentiment.subjectivity)

blob = TextBlob(t, analyzer=NaiveBayesAnalyzer())

#sentiment classification for the text

blob.sentiment #Sentiment(classification='pos', p\_pos=1.0, p\_neg=3.633586025330017e-19)

#Put in dataframe polsubj which has a column of polarity values and a column of subjectivity values

polsubj = pd.DataFrame({'polarity': polarity,'subjectivity': subj})

#Plot the line graph

polsubj.plot(title='Polarity and Subjectivity')

#sentiment analysis

#!pip install vaderSentiment

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

analyser = SentimentIntensityAnalyzer()

#sentiment analyser

df['Sentiment Analyser'] = df.TEXT.apply(lambda x:analyser.polarity\_scores(x))

#create a different column of all scores which are present as json array

k=df['Sentiment Analyser'].apply(pd.DataFrame,index=[0]).tolist()

final\_df = pd.concat(k)

final\_df.index = pd.Series(final\_df.index).shift(-1).fillna(0).cumsum()

#add only compound score to our original data set

df = pd.DataFrame(df)

df1=pd.DataFrame(final\_df)

#add score to original data frame

df['Compound Score'] = df1['compound'].tolist()

#function to determine category of sentiment

def Category\_function(x):

if x> 0.05 :

return "Positive"

elif x > -0.05 and x <0.05:

return "Neutral"

elif x <= -0.05:

return "Negative"

df['Sentiment\_Category']= df['Compound Score'].apply(lambda x: Category\_function(x) )

#Sentiment\_Category is converted to category type

df['Sentiment\_Category'] = df['Sentiment\_Category'].astype('category')

df.dtypes

#add a column to have numerical columns to category

df['Sentiment\_Category\_Code'] = df['Sentiment\_Category'].cat.codes

Sentiment\_Category\_Code= df['Sentiment\_Category\_Code']

df['Sentiment\_Category\_code\_1']= df['Sentiment\_Category\_Code'].apply(lambda x: 1 if x>=1 else 0)

Sentiment\_Category\_code\_1=df['Sentiment\_Category\_code\_1']

#naive bayes

from sklearn.model\_selection import train\_test\_split

X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X, Sentiment\_Category\_code\_1, test\_size=0.3)

from sklearn.naive\_bayes import MultinomialNB

clf=MultinomialNB()

clf.fit(X\_train1, y\_train1)

print (clf.score(X\_train1, y\_train1))

print (clf.score(X\_test1, y\_test1))

predicted\_result=clf.predict(X\_test1)

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

print(classification\_report(y\_test1,predicted\_result))

Naive\_bayes\_score = accuracy\_score(y\_test1, predicted\_result)

Naive\_bayes\_score

#accuracy score 0.82

#Using random forest

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

rf.fit(X\_train1, y\_train1)

# calculating the random forest accuracy

y\_pred\_rf = rf.predict(X\_test1)

random\_forest\_score = accuracy\_score(y\_test1, y\_pred\_rf)

#accuracy 0.749

random\_forest\_score

print(classification\_report(y\_test1,y\_pred\_rf))

#SVM

from sklearn import svm

svc = svm.SVC()

svc.fit(X\_train1, y\_train1)

#calculating the SVM accyracy

y\_pred\_svm = svc.predict(X\_test1)

svc\_score = accuracy\_score(y\_test1, y\_pred\_svm)

svc\_score

#0.6632

#lstm

from sklearn.feature\_extraction.text import CountVectorizer

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM

from sklearn.model\_selection import train\_test\_split

from keras.utils.np\_utils import to\_categorical

import re

max\_fatures = 30000

tokenizer = Tokenizer(nb\_words=max\_fatures, split=' ')

tokenizer.fit\_on\_texts(df['New Text'].values)

X2 = tokenizer.texts\_to\_sequences(df['New Text'].values)

X2 = pad\_sequences(X2)

Y2 = pd.get\_dummies(df['Sentiment\_Category\_code\_1']).values

X2\_train, X2\_test, Y2\_train, Y2\_test = train\_test\_split(X2,Y2, random\_state = 42)

print(X2\_train.shape,Y2\_train.shape)

print(X2\_test.shape,Y2\_test.shape)

embed\_dim = 150

lstm\_out = 200

model = Sequential()

model.add(Embedding(max\_fatures, embed\_dim,input\_length = X2.shape[1], dropout=0.2))

model.add(LSTM(lstm\_out, dropout\_U=0.2,dropout\_W=0.2))

model.add(Dense(2,activation='softmax'))

model.compile(loss = 'categorical\_crossentropy', optimizer='adam',metrics = ['accuracy'])

prssint(model.summary())

history=model.fit(X2\_train, Y2\_train, epochs=10, batch\_size=250, validation\_split=0.2)

#plot loss function

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(loss))

plt.plot(epochs, loss, 'orange', label='train loss')

plt.plot(epochs, val\_loss, 'blue', label='test loss')

plt.title("Training and validation loss")

plt.legend()

#plot accuracy

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

plt.plot(epochs, acc, 'orange', label='train accuracy')

plt.plot(epochs, val\_acc, 'blue', label='test accuracy')

plt.title("Training and validaton accuracy")

plt.legend()

# print the model summary

print (model.summary())

# test the model with pretrained weights

scores = model.evaluate(X2\_test, Y2\_test, verbose=1)

print("Accuracy: %.2f%%" % (scores[1]\*100))

score,acc = model.evaluate(X2\_test, Y2\_test, verbose = 2, batch\_size = 10)

print("score: %.2f" % (score))

print("acc: %.2f" % (acc))