“Sarcasm Detection”

(A data science project report)

Submitted By

Arun Kumar

arunk.recs.cse@gmail.com

To

edWisor.com

Contents

Chapter 1

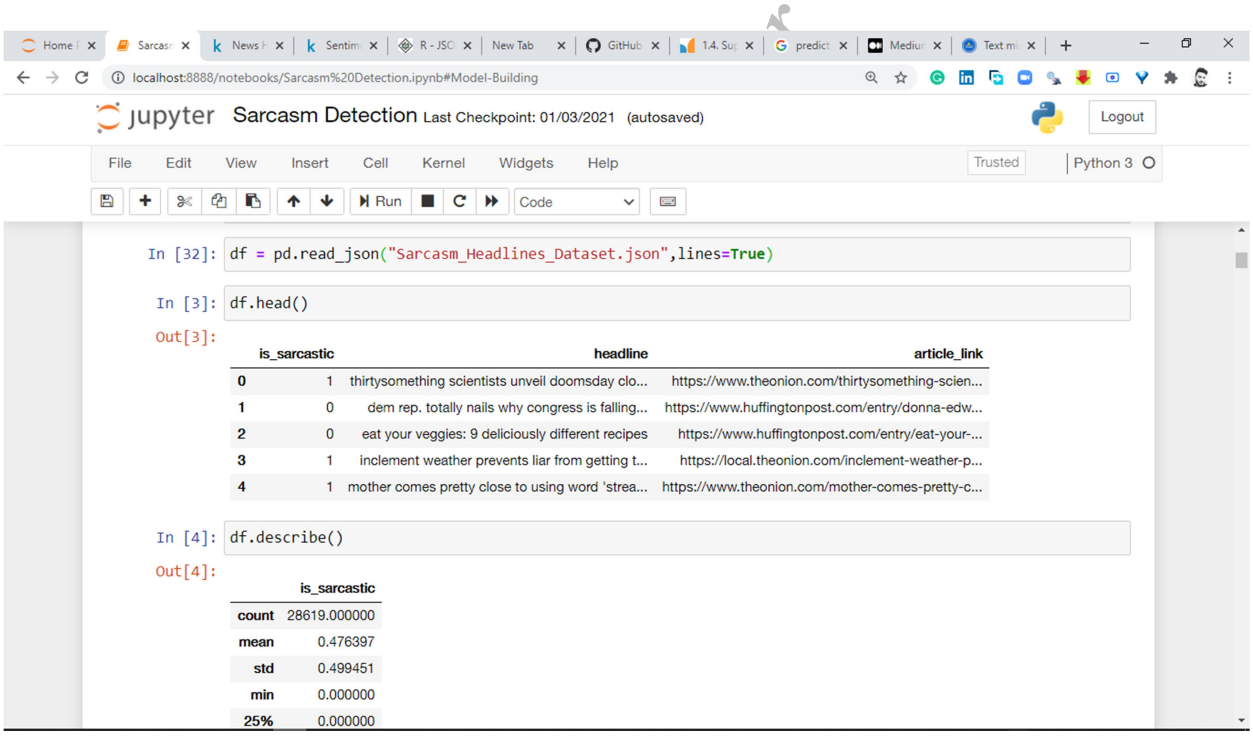
Introduction

1.1 Problem Statement

This case requires trainees to develop a text classification model to label a news headline as sarcastic or not. This News Headlines dataset for Sarcasm Detection is collected from two news website. We collect real (and non-sarcastic) news headlines from different Post.

1.2 Data

We have total 3 columns in given data. For more details look below pictures.



	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clo...	https://www.theonion.com/thirtysomething-scien...
1	0	dem rep. totally nails why congress is falling...	https://www.huffingtonpost.com/entry/donna-edw...
2	0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-...
3	1	inclement weather prevents liar from getting t...	https://local.theonion.com/inclement-weather-p...
4	1	mother comes pretty close to using word 'strea...	https://www.theonion.com/mother-comes-pretty-c...

	is_sarcastic
count	28619.000000
mean	0.476397
std	0.499451
min	0.000000
25%	0.000000

Attributes:-

- is_sarcastic – This column represent the news headline is sarcastic or not
- headline – this column represent the headline
- article_link- this contains the news source link

1.3 Project Explanation

This report is an analysis of Text data from News headlines. The data contains news headlines from two different websites. We have to classify whether a headline is **sarcastic or non-sarcastic based on words used** in it. The aim of this project is to build a **Text classification model** to classify statements to sarcastic or non-sarcastic **to understand the true context** in which it appears.

Chapter 2

Methodology

2.1 Exploratory Data Analysis

The Given dataset contains 28619 Observations and 3 is_sarcastic, headline and article_link. The Independent variables needed for modelling should be structured, but our only predictor variable is unstructured. So, our first goal is to convert the unstructured headline to structured variables (Terms). We use tm package for this text mining operations involved.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28619 entries, 0 to 28618
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   is_sarcastic  28619 non-null  int64
 1   headline      28619 non-null  object
 2   article_link  28619 non-null  object
dtypes: int64(1), object(2)
memory usage: 670.9+ KB
```


2.1.0. Understanding Data

Exploratory Data Analysis (EDA) is an approach to analysing data sets and Summarize their main characteristics.Given json data file consists 28619 observation and 3 variables. Data type of all variables is either int64 or object. As per the data analysis we have to find which variables are the categorical variables, continuous variables and target variable. Data types need to be change accordingly. We have distributed the variables on the basis of continuous and categorical variables. Target variable is binary. We have total 28619 observation, but as per above summary tables total observation is equal to 28619. Its means there is no missing values present in our dataset. Missing value analysis is required to further understand the data.

Data set description given below.

```
<bound method NDFrame.describe of       is_sarcastic                                  headline  \
0                1  thirtysomething scientists unveil doomsday clo...
1                0  dem rep. totally nails why congress is falling...
2                0  eat your veggies: 9 deliciously different recipes
3                1  inclement weather prevents liar from getting t...
4                1  mother comes pretty close to using word 'strea...
...            ...                                                ...
28614            1      jews to celebrate rosh hashasha or something
```

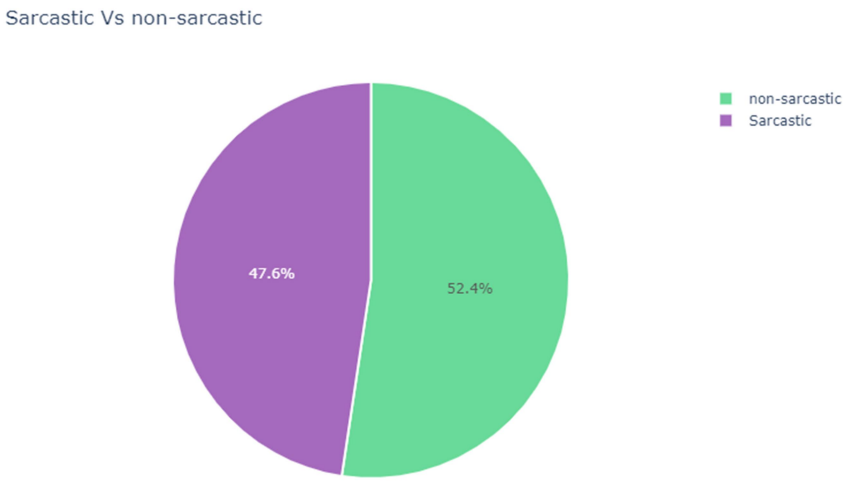
28615	1	internal affairs investigator disappointed con...
28616	0	the most beautiful acceptance speech this week...
28617	1	mars probe destroyed by orbiting spielberg-gat...
28618	1	dad clarifies this not a food stop

	article_link
0	https://www.theonion.com/thirtysomething-scien...
1	https://www.huffingtonpost.com/entry/donna-edw...
2	https://www.huffingtonpost.com/entry/eat-your-...
3	https://local.theonion.com/inclement-weather-p...
4	https://www.theonion.com/mother-comes-pretty-c...
...	...
28614	https://www.theonion.com/jews-to-celebrate-ros...
28615	https://local.theonion.com/internal-affairs-in...
28616	https://www.huffingtonpost.com/entry/andrew-ah...
28617	https://www.theonion.com/mars-probe-destroyed-...
28618	https://www.theonion.com/dad-clarifies-this-no...

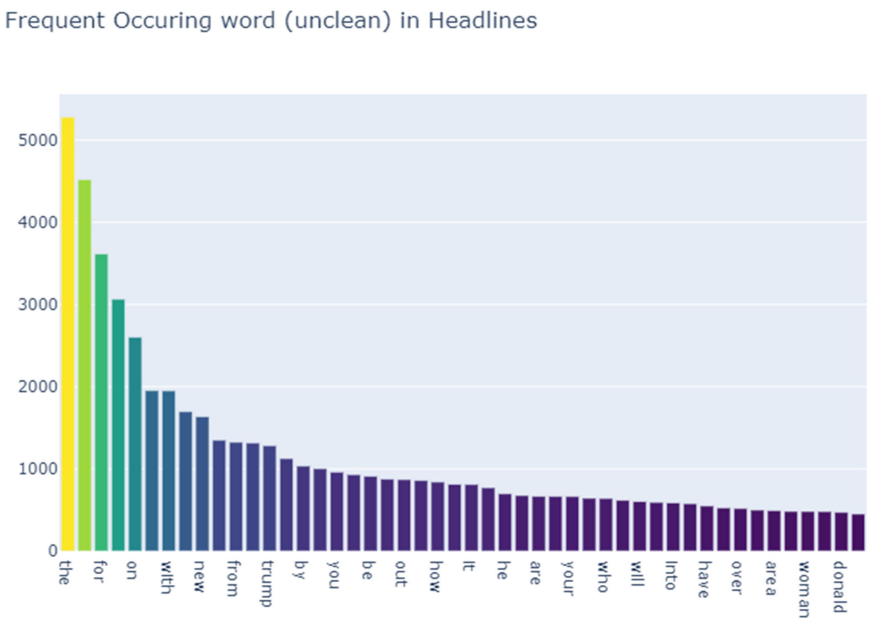
[28619 rows x 3 columns]>

2.1.1. Graphical EDA

Create Pie chart for see the % distribution of the sarcastic vs Non-sarcastic news headlines.



Check the most frequent words in un-clean data



2.2. Data Pre-processing and text cleaning

Data pre-processing is the main step of any data science or machine learning project. And this is the steps where most of time a data scientist or machine learning engineer spend. We perform many sub-steps in data pre-processing. In NLP project we perform mainly below performed steps-

2.2.0. Remove unnecessary characters

The is a primary step in the process of text cleaning. If we scrap some text from HTML/XML sources, we'll need to get rid of all the tags, HTML entities, punctuation, non-alphabets, and any other kind of characters which might not be a part of the language. The general methods of such cleaning involve regular expressions, which can be used to filter out most of the unwanted texts.
There are some systems where important English characters like the full-stops, question-marks, exclamation symbols, etc. are retained. Consider an example where you want to perform some sentiment analysis on human generated tweets, and you want to classify the tweets is very angry, angry, neutral, happy, and very happy. Simple sentiment analysis might find it hard to differentiate between a happy, and a very happy sentiment, because there can be some moments only words are not able to explain.
Consider the two sentences with the same semantic meaning:
"This food is good." and "This. Food. Is. Good!!!!!!!!".
See what I'm trying to say? Same words, but totally different sentiments, and the only information which can help us to see the different is the overused punctuation, which shows some sort of an "extra" feeling.
Emoticons, which are made up of non-alphabets also play a role in sentiment analysis. ":), :(, -_-, :D, xD", all these, when processed correctly, can help with a better sentiment analysis. Even if you want to develop a system that may classify whether some phrase is sarcasm, or not sarcasm, such little details can be helpful.

2.2.1. Tokenization

Tokenization is the process of breaking up the given text into units called tokens. The tokens may be words or number or punctuation mark. Tokenization does this task by locating word boundaries. Ending point of a word and beginning of the next word is called word boundaries. Tokenization is also known as word segmentation. Before tokenization we will remove the unnecessary characters from data

2.2.2. Removing Stopwords

This cleaning step also depends on what you'll eventually be doing with your data after pre-processing. Stopwords are the words which are used very frequently, and they're so frequent, that they somewhat lose their semantic meaning. Words like "of, are, the, it, is" are some examples of stopwords. In applications like document search engines and document classification, where keywords are more important than general terms, removing

stopwords can be a good idea, but if there's some application about, for instance, songs lyrics search, or search specific quotes, stopwords can be important. Consider some examples like "To be, or not not be", "Look what you made me do" etc. Stopwords in such phrases actually play an important role, and hence, should not be dropped. We would not want these words taking up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to be stop words.
We will remove the stopwords provided by NLTK package from our headlines. Let's find some words and clean them because stopwords have no meaning for a sentence. We will use python library (nltk) to detect them.

2.2.3. Stemming/ Lemmatisation

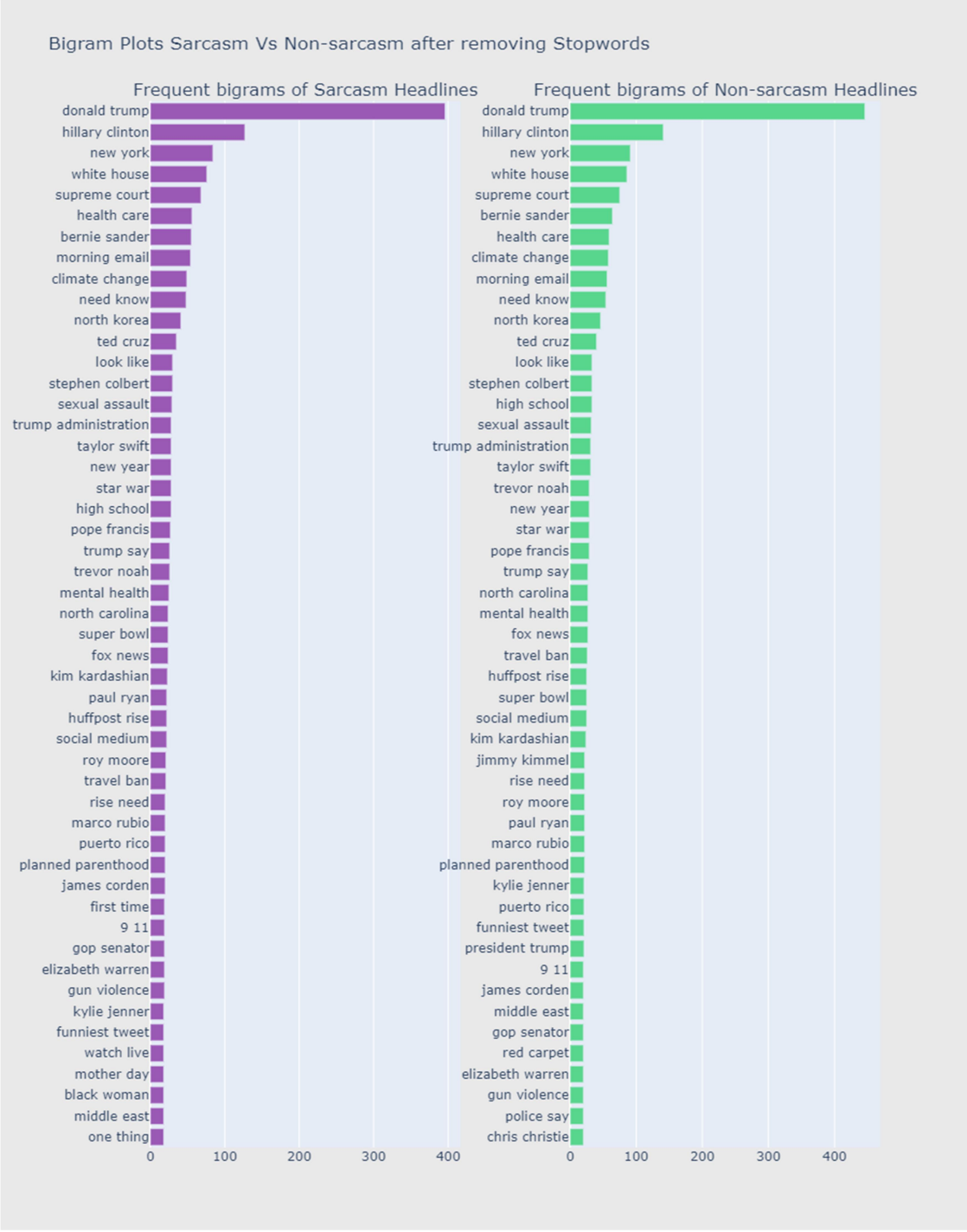
"The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form." With that being said, stemming/lemmatizing helps us reduce the number of overall terms to certain "root" terms.
Organizer, organizes, organization, organized all these get reduced to a root term, maybe "organize".
Stemming is a crude way of reducing terms to their root, by just defining rules of chopping off some characters at the end of the word, and hopefully, gets good results most of the time.

2.3. Data Visualization

After complete data pre-processing. I have visualized data and plot some graphs to solve the problem graphically.

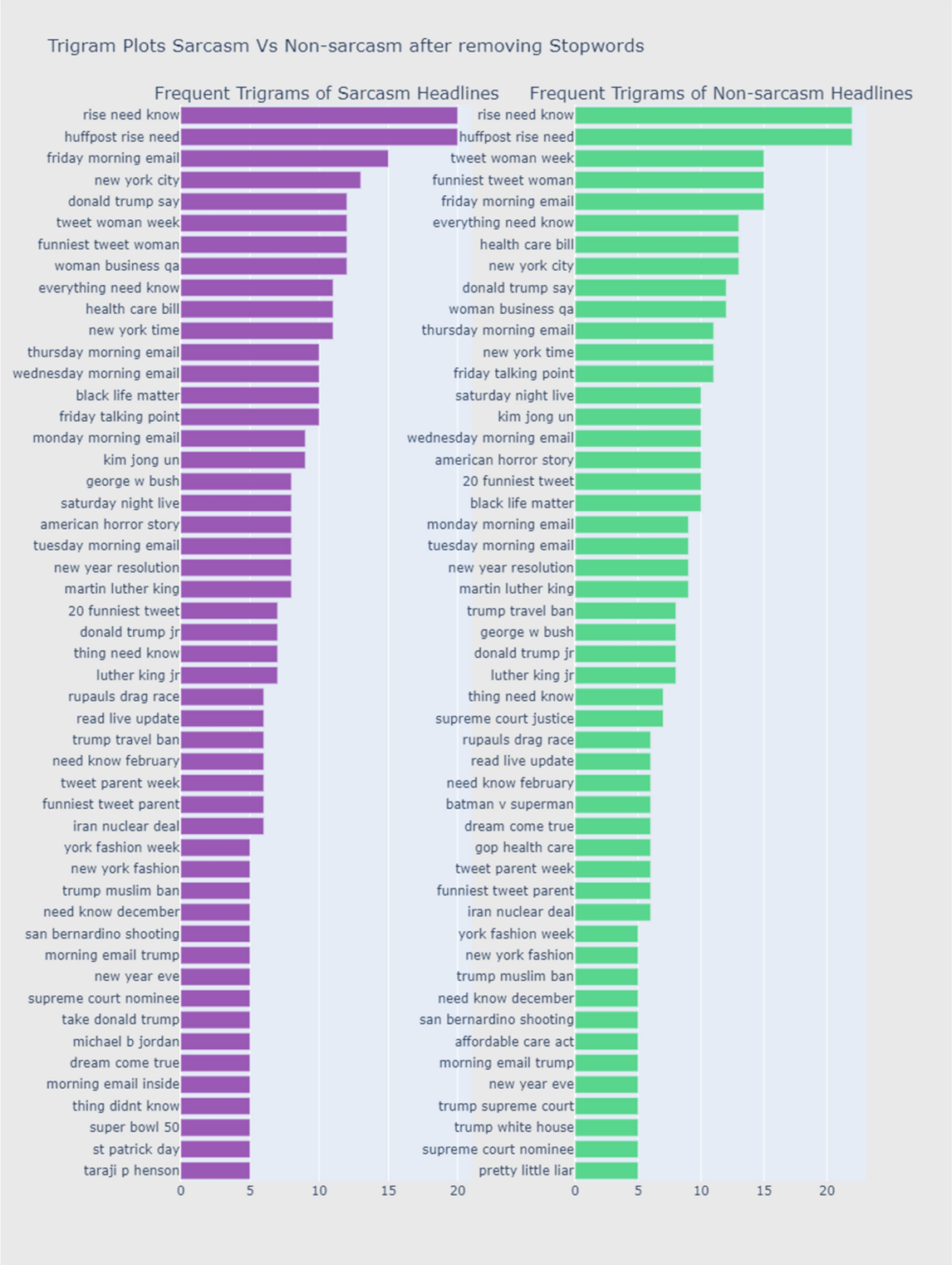
2.3.0. Bi-grams

Bi-gram is a two-word sequence of words. like “please turn”, “turn your”, or ”your homework”,



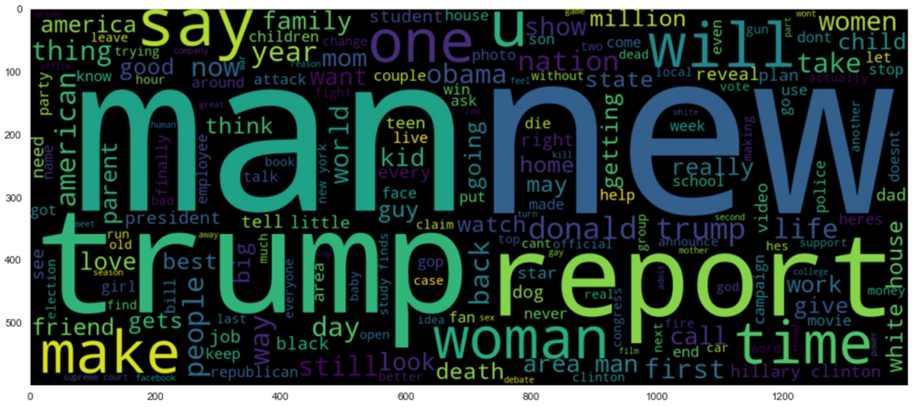
2.3.1. Tri-grams

A trigram is a three-word sequence of words like “please turn your”, or “turn your homework”.

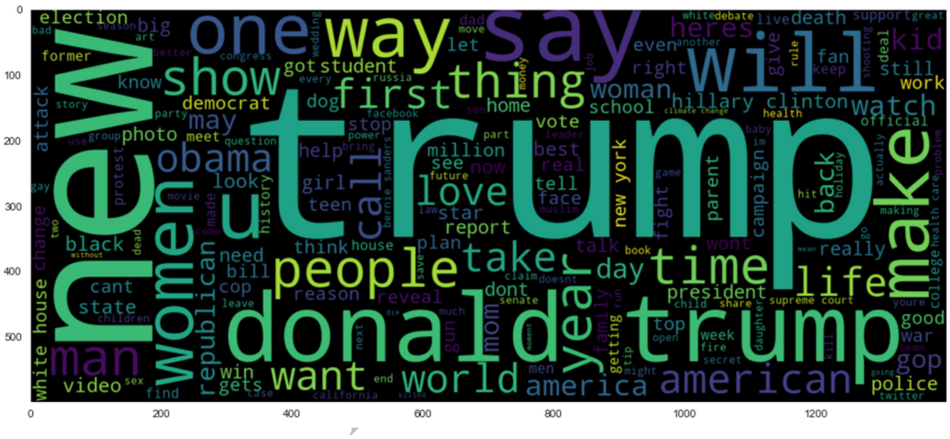


2.3.2. Word cloud

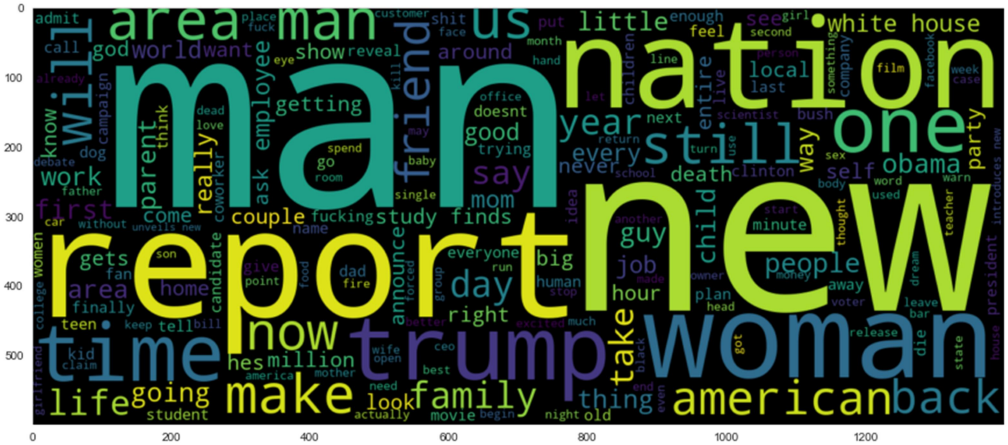
Word cloud for 200 frequant words in complete headlines data.



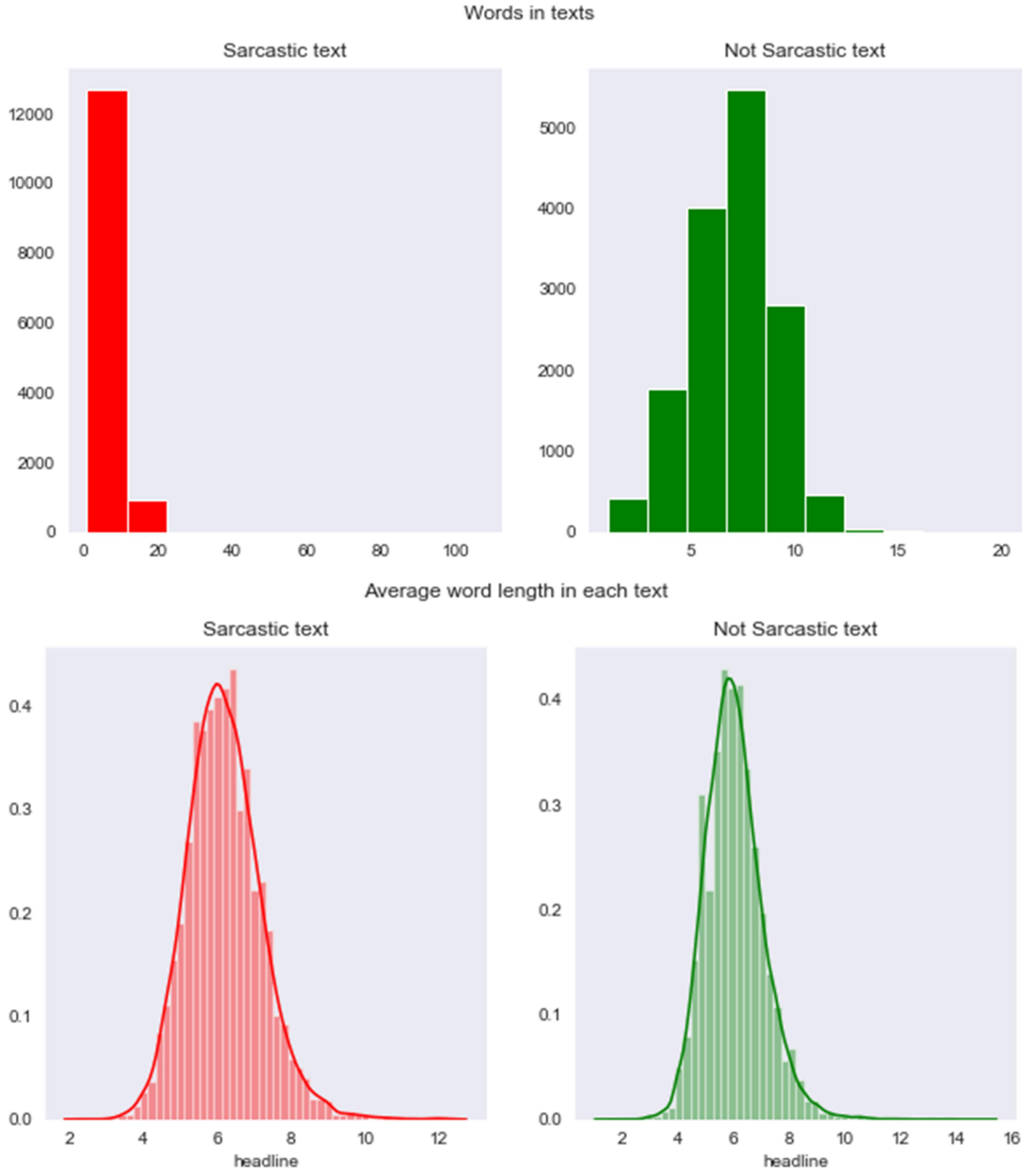
Word cloud for 200 frequant words in non-sarcastic headlines.

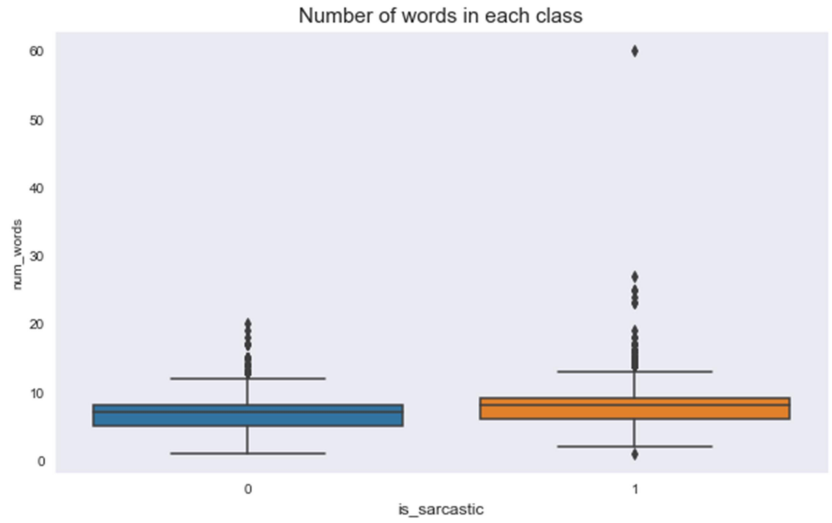


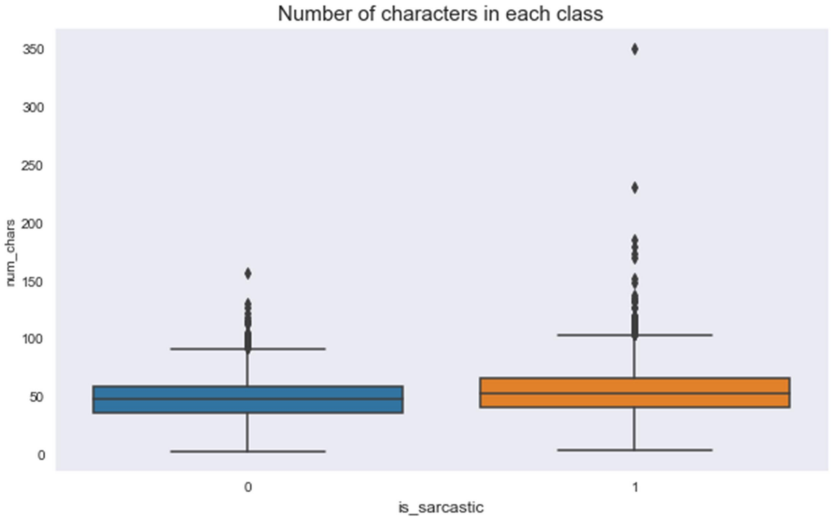
Word cloud for 200 frequant words in non-sarcastic headlines.

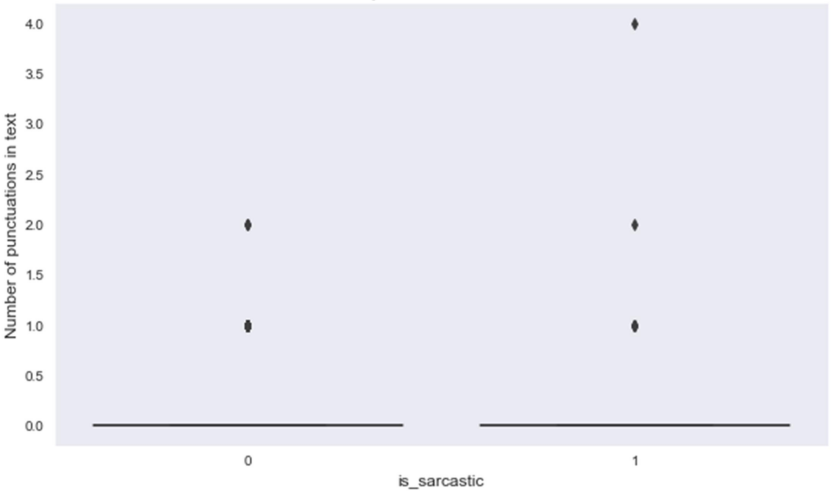


2.3.3. Distribution of Sarcasm Vs Non-sarcasm Headlines

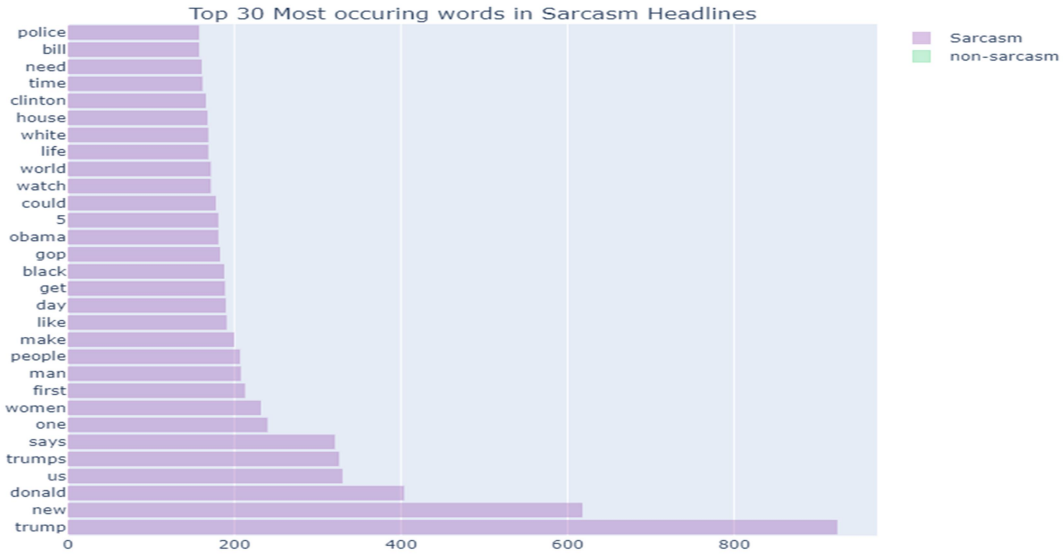


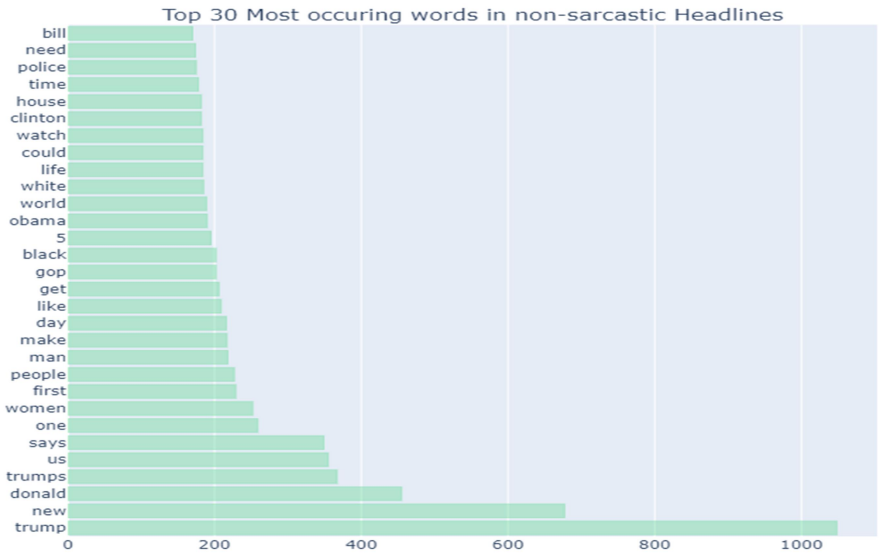

Number of words in each class
num_words
0
10
20
30
40
50
60
0
1
is_sarcastic


Number of characters in each class
num_chars
0
50
100
150
200
250
300
350
0
1
is_sarcastic


Number of punctuations in each class
Number of punctuations in text
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
1
is_sarcastic

Most 30 frequant words visualization in sarcastic and mon-sarcastic headline after pre-processing.





2.2. Model Development

Our problem statement wants us to predict the sarcasm in news headline. This is a classification problem. So, we are going to build classification models on training data and predict it on test data. In this project I have built models using 5 classification Algorithms:

2.2.1. Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).

2.2.2. SVM

SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs.

2.2.3. Multinomial NB

The term Multinomial Naive Bayes simply lets us know that each p(fi|c) is a multinomial distribution, rather than some other distribution. This works well for data which can easily be turned into counts, such as word counts in text.

2.2.4. Deep Learning Model

I create a deep learning based model using LSTM layers. We can call it RNN algorithm. Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data

2.5. Model Evolution

We will evaluate performance on validation dataset which was generated using Sampling. We will deal with specific metrics like – Classification report and confusion matrix for our Models.

This are the performs reports of all models attached below:-

Logistic Regression

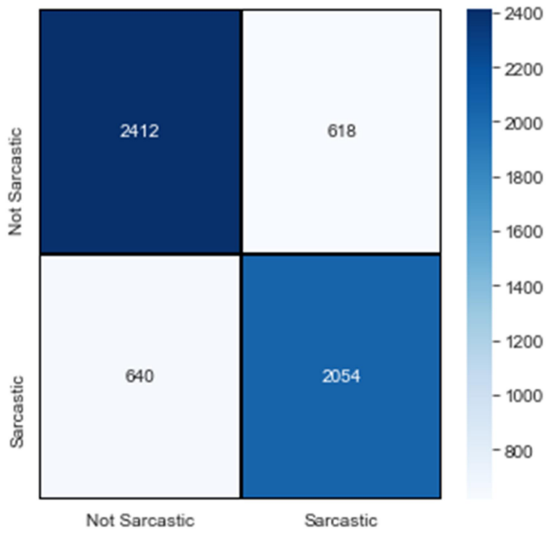
Classification report for LR

```
accuracy 0.7802236198462613
              precision    recall  f1-score   support

     sarcasm       0.79      0.80      0.79      3030
 non-sarcasm       0.77      0.76      0.77      2694

    accuracy                           0.78      5724
   macro avg       0.78      0.78      0.78      5724
weighted avg       0.78      0.78      0.78      5724
```

Confusion matrix for LR



SVM Model

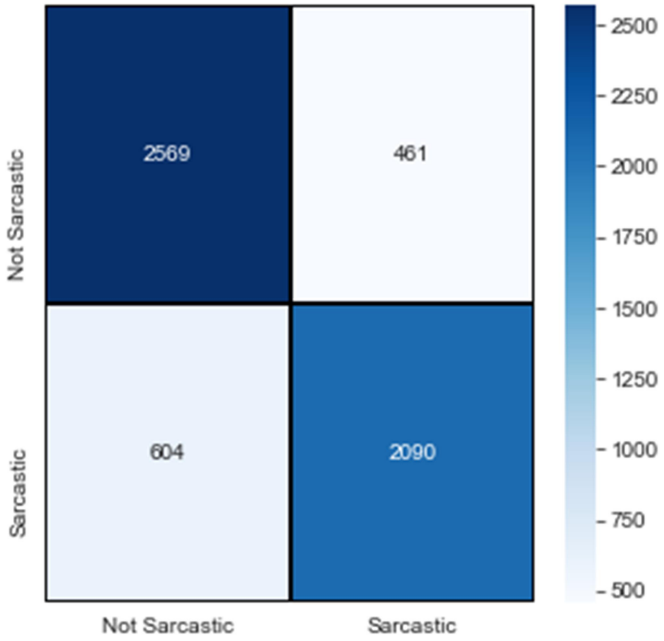
Classification report for SVM

```
accuracy 0.8139412997903563
              precision    recall  f1-score   support

     sarcasm       0.81      0.85      0.83      3030
 non-sarcasm       0.82      0.78      0.80      2694

    accuracy                           0.81      5724
   macro avg       0.81      0.81      0.81      5724
weighted avg       0.81      0.81      0.81      5724
```

Confusion matrix for SVM



MNB Model

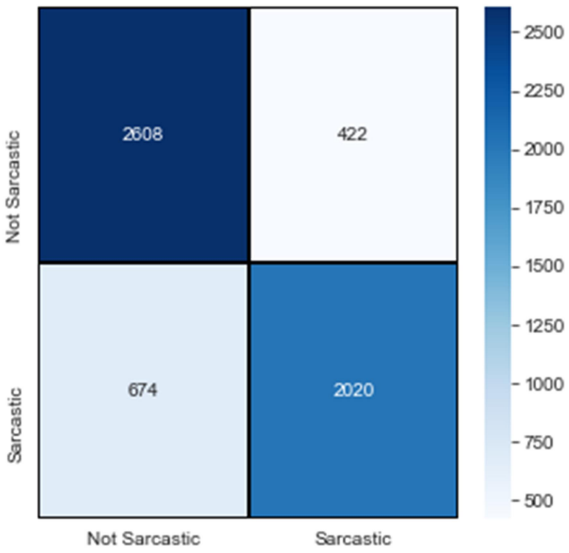
Classification report for MNB

```
accuracy 0.8085255066387141
              precision    recall  f1-score   support

     sarcasm       0.79      0.86      0.83      3030
 non-sarcasm       0.83      0.75      0.79      2694

    accuracy                           0.81      5724
   macro avg       0.81      0.81      0.81      5724
weighted avg       0.81      0.81      0.81      5724
```

Confusion matrix for MNB



Deep Learning Model

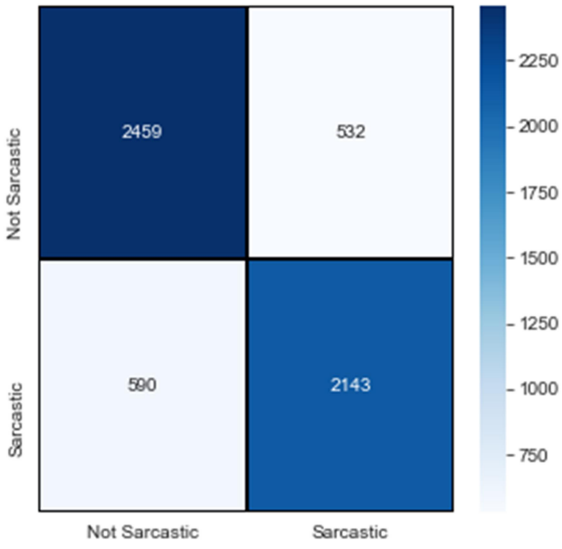
Classification report for DL Model

```
accuracy 0.8039832285115304
              precision    recall  f1-score   support

     sarcasm       0.81      0.82      0.81      2991
 non-sarcasm       0.80      0.78      0.79      2733

    accuracy                           0.80      5724
   macro avg       0.80      0.80      0.80      5724
weighted avg       0.80      0.80      0.80      5724
```

Confusion matrix for DL Model



Chapter 3

Conclusion

Asked questions in problem statement:

Question 1. Find the 6 topics related to news article headlines.

Answer 1. After Analysis of news headlines I got below 6 topic that are more related to news headlines-

1. Politics
2. Entertainments
3. Sport
4. Study
5. Medical
6. Travel

Question 2. Do EDA to find the top 50 sarcastic words. Make a word cloud for top 200 frequent words.

Answer 2. These are the top 50 sarcastic words-

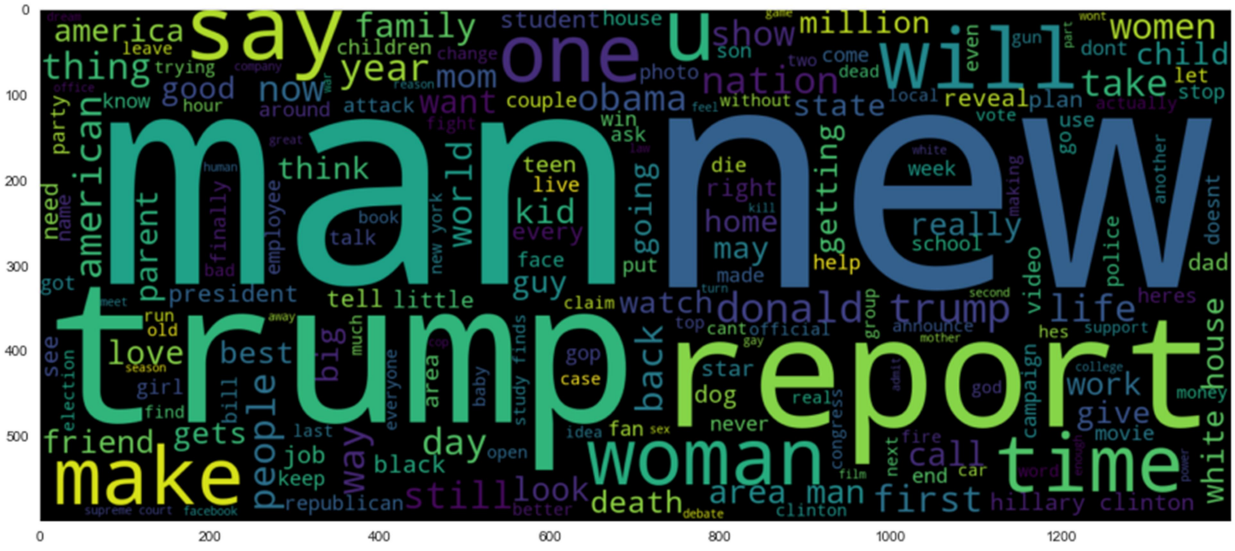
index	Words	Freq
0	trump	923
1	new	617
2	donald	403
3	us	329
4	trumps	325
5	says	320
6	one	239
7	women	231
8	first	212
9	man	207

index	Words	Freq
10	people	206
11	make	199
12	like	190
13	day	189
14	get	188
15	black	187
16	gop	182
17	obama	180
18	5	180
19	could	177
20	watch	171
21	world	171
22	life	168
23	white	168
24	house	167
25	clinton	165
26	time	161
27	need	160
28	bill	157

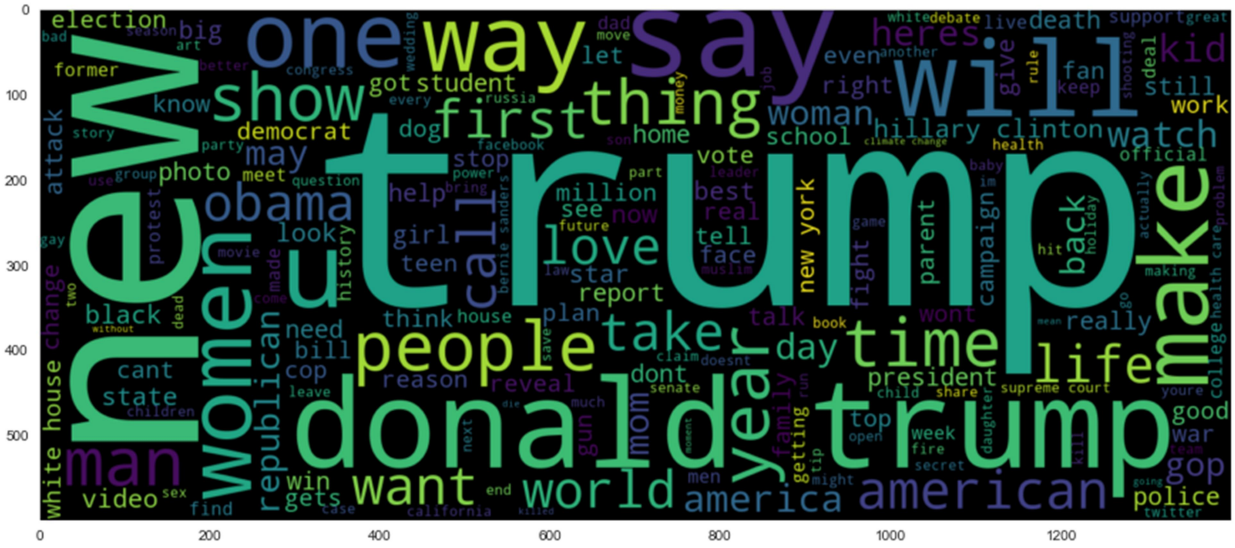
index	Words	Freq
29	police	157
30	heres	155
31	best	152
32	years	151
33	health	150
34	video	149
35	hillary	147
36	things	146
37	know	145
38	love	144
39	president	143
40	say	141
41	10	135
42	woman	135
43	way	134
44	show	131
45	kids	129
46	may	123
47	dont	120

index	Words	Freq
48	change	119
49	want	118

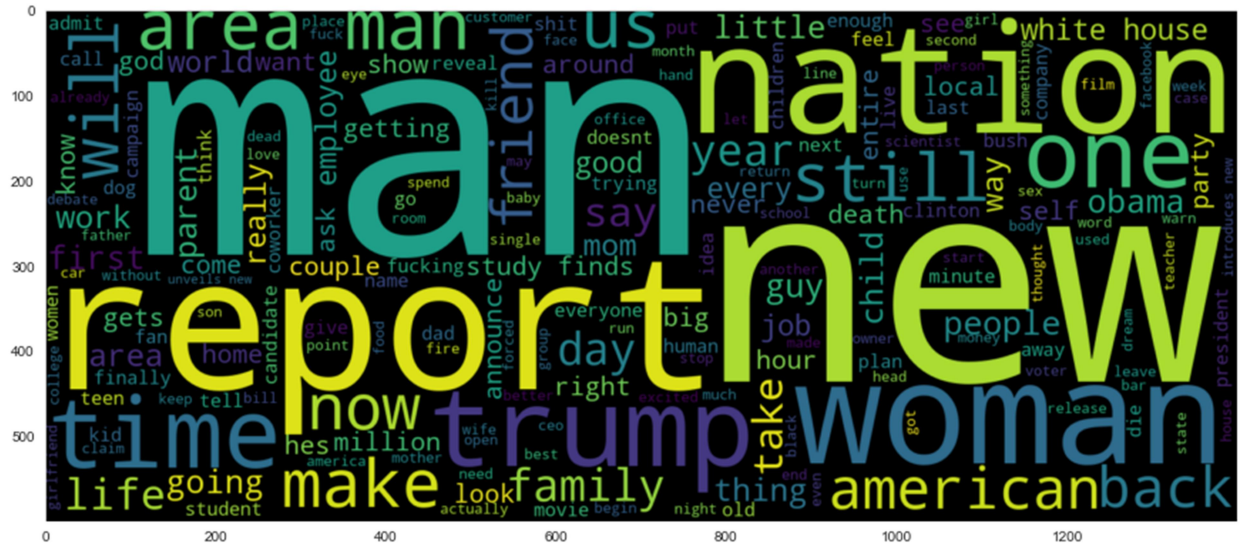
And the word cloud for 200 frequant words in complete headlines data.



Word cloud for 200 frequant words in non-sarcastic headlines.

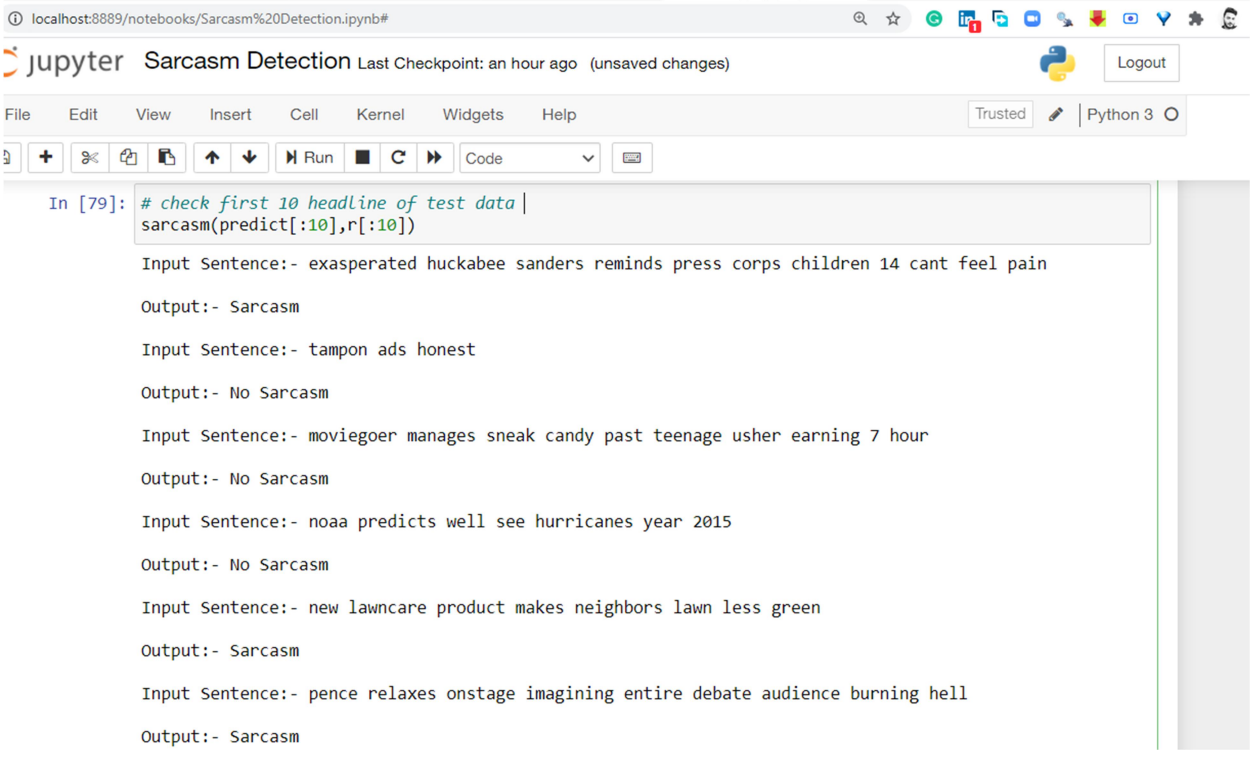


Word cloud for 200 frequant words in non-sarcastic headlines.



Question 3. Can you identify sarcastic sentences? Can you distinguish between fake news and legitimate news?

Answer 3. I have identify first 10 headlines of test data using created model. And got below results



On the base of model classification we can identify if model predict the 0 then this news headline will be non-sarcastic and if model predict the 1 then this news headline will be sarcastic. In general if a news is not real then it will we sarcastic news. If the news has really correct then it will have proper details at its source point.

Chapter 4

Codes

4.1. Python code

```
# In[1]:


#import all necessary libs
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from collections import Counter
import string
import nltk
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import re,string,unicodedata
from keras.preprocessing import text, sequence
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import train_test_split
from string import punctuation
import keras
from keras.models import Sequential
from keras.layers import Dense,Embedding,LSTM,Dropout,Bidirectional,GRU
import tensorflow as tf


# In[2]:


# read the data file
data = pd.read_json("Sarcasm_Headlines_Dataset.json",lines=True)
df=data.copy()
```

In[3]:

```
# CHeck the top 5 rows of the data set
df.head()
```

In[4]:

```
# describtion of data
df.describe()
```

In[5]:

```
# check the information of the data
df.info()
```

In[6]:

```
df.describe
```

Dataset Exploratory Data Analysis

In the dataset, if headline is sarcastic, "is_sarcastic" column value 1. If not sarcastic, value=0. Also article value is provided. But our main focus is to build a model that can detect sarcastic news based on "headline" and "is_sarcastic" column.

In[7]:

```
#lets find if there is any NaN valus, because NaN values give wrong visualization
df.isna().sum()
```

In[8]:

```
#Lets drop the "article link" column from dataframe as it is not needed
#del df['article_link']
```

In[9]:

```
#Lets create a barplot/countplot to compare between sarcastic or non-sarcastic news
sns. set_style("dark")
sns.countplot(df.is_sarcastic);
```

so there are almost same number of sarcastic and non-sarcastic news.

In[10]:

```
#  Check the different sourse of news
df.article_link.apply(lambda x: x.split('.')[1]).value_counts()


# In[11]:


# create website column for sourse link of the news
df['website'] = df.article_link.apply(lambda x: x.split('.')[1])
df.info()


# In[12]:


#view different news sourses
sns.countplot(x= df.website ,data=df, order = df['website'].value_counts().index);


# In[13]:


sar_acc_tar = df['is_sarcastic'].value_counts()
labels = ['non-sarcastic', 'Sarcastic']
sizes = (np.array((sar_acc_tar / sar_acc_tar.sum())*100))
colors = ['#58D68D', '#9B59B6']

trace = go.Pie(labels=labels, values=sizes, opacity = 0.9, hoverinfo='label+percent',
            marker=dict(colors=colors, line=dict(color='#FFFFFF', width=2)))
layout = go.Layout(
    title='Sarcastic Vs non-sarcastic'
)
data = [trace]
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename="Sa_Ac")


# #### Frequent Occuring word (unclean) in Headlines

# In[14]:


all_words = df['headline'].str.split(expand=True).unstack().value_counts()
data = [go.Bar(
            x = all_words.index.values[2:50],
            y = all_words.values[2:50],
            marker= dict(colorscale='Viridis',
                         color = all_words.values[2:100]
                        ),
            text='Word counts'
    )]

layout = go.Layout(
    title='Frequent Occuring word (unclean) in Headlines'
)

fig = go.Figure(data=data, layout=layout)
```

```
iplot(fig, filename='basic-bar')


# From the above plot its clearly evident that the headlines need to be cleaned as the top 50 most occuring words are
        joing words and indirect words which does not provide any meaning.

# ## Tokenization
# Tokenization is the process of breaking up the given text into units called tokens. The tokens may be words or
        number or punctuation mark. Tokenization does this task by locating word boundaries. Ending point of a
        word and beginning of the next word is called word boundaries. Tokenization is also known as word
        segmentation.
#
# Before tokenization we will remove the unnecessary characters from data

# In[16]:


REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;:]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')

def clean_text(text):
    """
        text: a string

        return: modified initial string
    """
    text = BeautifulSoup(text, "lxml").text # HTML decoding
    text = text.lower() # lowercase text
    text = REPLACE_BY_SPACE_RE.sub(' ', text) # replace REPLACE_BY_SPACE_RE symbols by space in text
    text = BAD_SYMBOLS_RE.sub('', text) # delete symbols which are in BAD_SYMBOLS_RE from text
    return text

df['headline'] = df['headline'].apply(clean_text)


# In[17]:


#non-sarcatic headlines tokenization
n_sar_list=[]
for i in df[df['is_sarcastic']==0]['headline'].values:
    n_sar_list.append(i.split())


# In[18]:


n_sar=[]
for i in range(len(n_sar_list)):
    for j in n_sar_list[i]:
        n_sar.append(j)


# In[19]:


#sarcatic headlines tokenization
sar_list=[]
for i in df[df['is_sarcastic']==1]['headline'].values:
    l=i.split()
```

```
    sar_list.append(l)



# In[20]:


sar = []
for i in range(len(sar_list)):
    for j in n_sar_list[i]:
        sar.append(j)


# ## Removing Stopwords
#
# A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed
        to ignore, both when indexing entries for searching and when retrieving them as the result of a search
        query.
#
# We would not want these words taking up space in our database, or taking up valuable processing time. For this,
        we can remove them easily, by storing a list of words that you consider to be stop words.
#
# We will remove the stopwords provided by NLTK package from our headlines.

# Lets find some somewords and clean them because stopwords have no meaning for a sentence. we will use python
        library(nltk) to detect them.

# In[21]:


#set and define stop word
#nltk.download('stopwords')
stopwords = set(nltk.corpus.stopwords.words('english'))
punctuation = list(string.punctuation)
stopwords.update(punctuation)


# In[22]:


sar_list_restp = [word for word in sar if word.lower() not in stopwords]
n_sar_list_restp = [word for word in n_sar if word.lower() not in stopwords]

print("Length of original Sarcasm list: {0} words\n"
      "Length of Sarcasm list after stopwords removal: {1} words"
      .format(len(sar), len(sar_list_restp)))

print("=="*46)

print("Length of original non_sarcastic list: {0} words\n"
      "Length of non_sarcastic list after stopwords removal: {1} words"
      .format(len(n_sar), len(n_sar_list_restp)))


# In[23]:


def remove_stopwd(text):
    text = ' '.join(word for word in text.split() if word not in STOPWORDS) # delete stopwors from text
    return text
```

```
df['headline']=df['headline'].apply(remove_stopwd)


# In[24]:


df.head()


# #### Top 30 Occuring words after removing Stopwords from Headlines - Sarcasm Vs Non-sarcasm

# In[25]:


#Data cleaning for getting top 30
sar_cnt = Counter(sar_list_restp)
acc_cnt = Counter(n_sar_list_restp)

#Dictonary to Dataframe
sar_cnt_df = pd.DataFrame(list(sar_cnt.items()), columns = ['Words', 'Freq'])
sar_cnt_df = sar_cnt_df.sort_values(by=['Freq'], ascending=False)
acc_cnt_df = pd.DataFrame(list(acc_cnt.items()), columns = ['Words', 'Freq'])
acc_cnt_df = acc_cnt_df.sort_values(by=['Freq'], ascending=False)

#Top 30
sar_cnt_df_30 = sar_cnt_df.head(30)
acc_cnt_df_30 = acc_cnt_df.head(30)


# In[26]:


#Plotting the top 30 Sarcasm Vs Acclaim
from plotly import tools
sar_tr  = go.Bar(
   x=sar_cnt_df_30['Freq'],
   y=sar_cnt_df_30['Words'],
   name='Sarcasm',
   marker=dict(
      color='rgba(155, 89, 182, 0.6)',
      line=dict(
         color='rgba(155, 89, 182, 1.0)',
         width=.3,
      )
   ),
   orientation='h',
   opacity=0.6
)

acc_tr  = go.Bar(
   x=acc_cnt_df_30['Freq'],
   y=acc_cnt_df_30['Words'],
   name='non-sarcasm',
   marker=dict(
      color='rgba(88, 214, 141, 0.6)',
      line=dict(
         color='rgba(88, 214, 141, 1.0)',
         width=.5,
      )
```

```
    ),
    orientation='h',
    opacity=0.6
)

fig = tools.make_subplots(rows=2, cols=1, subplot_titles=('Top 30 Most occuring words in Sarcasm Headlines',
                                                          'Top 30 Most occuring words in non-sarcastic Headlines'))

fig.append_trace(sar_tr, 1, 1)
fig.append_trace(acc_tr, 2, 1)


fig['layout'].update(height=1600, width=800)

iplot(fig, filename='sar_vs_n_sar')


# ### Stemming
# In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes
        derived) words to their word stem, base or root form - generally a written word form. The stem need not be
        identical to the morphological root of the word; it is usually sufficient that related words map to the same
        stem, even if this stem is not in itself a valid root.
#
# Source:
# https://en.wikipedia.org/wiki/Stemming
#
# NLTK provides three different forms of steamming namely Porter stemming algorithm, the lancaster stemmer and
        the Snowball stemmer. Here, for our analysis we will be using Snowball stemmer.

# In[27]:


# Example of snowballstemmer algorithm
stemmer = nltk.stem.SnowballStemmer("english", ignore_stopwords=True)
print("The stemmed form of learning is: {}".format(stemmer.stem("learning")))
print("The stemmed form of learns is: {}".format(stemmer.stem("learns")))
print("The stemmed form of learn is: {}".format(stemmer.stem("learn")))
print("=="*46)
print("The stemmed form of leaves is: {}".format(stemmer.stem("leaves")))
print("=="*46)


# Here is the caveat in using stemming. In the above example for the word 'leaves', it just stemms the word. As the
        name 'stemming' suggest, it at times simply stems the word which will become meaningless. So, to
        overcome this issue we have lemmatization.

# ### Lemmatisation
#
#
# Lemmatisation (or lemmatization) in linguistics is the process of grouping together the inflected forms of a word
        so they can be analysed as a single item, identified by the word's lemma, or dictionary form. Unlike a
        stemmer, lemmatizing the dataset aims to reduce words based on an actual dictionary or vocabulary (the
        Lemma) and therefore will not chop off words into stemmed forms that do not carry any lexical meaning.
#
# Source:
# https://en.wikipedia.org/wiki/Lemmatisation

# In[28]:
```

```
#download wordnet
# nltk.download('wordnet')


# In[29]:


lemm = WordNetLemmatizer()
print("The lemmatized form of leaves is: {}".format(lemm.lemmatize("leaves")))


# In[30]:


sar_wost_lem=[]
sar_list_lemm = [lemm.lemmatize(word) for word in sar_list_restp]
sar_wost_lem.append(sar_list_lemm)


# In[31]:


n_sar_wost_lem = []
n_sar_list_lemm = [lemm.lemmatize(word) for word in n_sar_list_restp]
n_sar_wost_lem.append(n_sar_list_lemm)


# In[32]:


# top 50 frequant sarcastic word
sar_cnt_df.head(50).reset_index(drop=True)


# ### Bi-grams
#
# A bigram or digram is a sequence of two adjacent elements from a string of tokens, which are typically letters,
        syllables, or words. A bigram is an n-gram for n=2. The frequency distribution of every bigram in a string
        is commonly used for simple statistical analysis of text in many applications, including in computational
        linguistics, cryptography, speech recognition, and so on.
#
# Source:
# https://en.wikipedia.org/wiki/Bigram

# In[33]:


sar_wost_lem_df = pd.DataFrame({'sarcasm':sar_wost_lem})
n_sar_wost_lem_df = pd.DataFrame({'Non-sarcasm':n_sar_wost_lem})

## custom function for ngram generation ##
def generate_ngrams(text, n_gram=1):
    ngrams = zip(*[text[i:] for i in range(n_gram)])
    return [" ".join(ngram) for ngram in ngrams]

## custom function for horizontal bar chart ##
def horizontal_bar_chart(df, color):
    trace = go.Bar(
        y=df["word"].values[::-1],
        x=df["wordcount"].values[::-1],
```

```
        showlegend=False,
        orientation = 'h',
        marker=dict(
            color=color,
        ),
    )
    return trace

#Plotting the Bigram plot
from collections import defaultdict
freq_dict = defaultdict(int)
for sent in sar_wost_lem_df["sarcasm"]:
    for word in generate_ngrams(sent,2):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
sar_2 = horizontal_bar_chart(fd_sorted.head(50), '#9B59B6')


freq_dict = defaultdict(int)
for sent in n_sar_wost_lem_df["Non-sarcasm"]:
    for word in generate_ngrams(sent,2):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
acc_2 = horizontal_bar_chart(fd_sorted.head(50), '#58D68D')

# Creating two subplots
fig = tools.make_subplots(rows=1, cols=2, vertical_spacing=0.04,horizontal_spacing=0.15,
                          subplot_titles=["Frequent bigrams of Sarcasm Headlines",
                                          "Frequent bigrams of Non-sarcasm Headlines"])
fig.append_trace(sar_2, 1, 1)
fig.append_trace(acc_2, 1, 2)
fig['layout'].update(height=1200, width=900, paper_bgcolor='rgb(233,233,233)', title="Bigram Plots Sarcasm Vs
            Non-sarcasm after removing Stopwords")
iplot(fig, filename='word-plots')


# ### Trigram

# In[34]:


#Plotting the Trigram plot
from collections import defaultdict
freq_dict = defaultdict(int)
for sent in sar_wost_lem_df["sarcasm"]:
    for word in generate_ngrams(sent,3):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
sar_2 = horizontal_bar_chart(fd_sorted.head(50), '#9B59B6')


freq_dict = defaultdict(int)
for sent in n_sar_wost_lem_df["Non-sarcasm"]:
    for word in generate_ngrams(sent,3):
        freq_dict[word] += 1
fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x[1])[::-1])
fd_sorted.columns = ["word", "wordcount"]
```

```
acc_2 = horizontal_bar_chart(fd_sorted.head(50), '#58D68D')

# Creating two subplots
fig = tools.make_subplots(rows=1, cols=2, vertical_spacing=0.04,horizontal_spacing=0.15,
                          subplot_titles=["Frequent Trigrams of Sarcasm Headlines",
                                          "Frequent Trigrams of Non-sarcasm Headlines"])
fig.append_trace(sar_2, 1, 1)
fig.append_trace(acc_2, 1, 2)
fig['layout'].update(height=1200, width=900, paper_bgcolor='rgb(233,233,233)', title="Trigram Plots Sarcasm Vs
        Non-sarcasm after removing Stopwords")
iplot(fig, filename='word-plots')


# ### Wordcloud
#
# Create wordcloud for top 200 frequant word in all headlines

# In[35]:


plt.figure(figsize = (15,15)) # non-sarcastic words wordcloud
wordcld = WordCloud(max_words = 200 , width = 1400 , height = 600).generate(" ".join(df.headline))
plt.imshow(wordcld , interpolation = 'bilinear');


# Lets create a wordcloud from non-sarcastic news which will give us a view of sights which words are used here
        most

# In[36]:


plt.figure(figsize = (15,15)) # non-sarcastic words wordcloud
wordcld = WordCloud(max_words = 200 , width = 1400 , height = 600).generate(" ".join(df[df.is_sarcastic ==
        0].headline))
plt.imshow(wordcld , interpolation = 'bilinear');


# Lets create a wordcloud from sarcastic news which will give us a view of sights which words are used here most

# In[37]:


plt.figure(figsize = (15,15)) # non-sarcastic words wordcloud
wordcld = WordCloud(max_words = 200 , width = 1400 , height = 600).generate(" ".join(df[df.is_sarcastic ==
        1].headline))
plt.imshow(wordcld , interpolation = 'bilinear');


# #### Distribution of Sarcasm Vs Non-sarcasm Headlines

# In[38]:


## Number of words in the text ##
df["num_words"] = df["headline"].apply(lambda x: len(str(x).split()))

## Number of unique words in the text ##
df["num_unique_words"] = df["headline"].apply(lambda x: len(set(str(x).split())))

## Number of characters in the text ##
```

```
df["num_chars"] = df["headline"].apply(lambda x: len(str(x)))

## Number of stopwords in the text ##
df["num_stopwords"] = df["headline"].apply(lambda x: len([w for w in str(x).lower().split() if w in
        STOPWORDS]))

## Number of punctuations in the text ##
df["num_punctuations"] =df['headline'].apply(lambda x: len([c for c in str(x) if c in string.punctuation]) )

## Number of title case words in the text ##
df["num_words_upper"] = df["headline"].apply(lambda x: len([w for w in str(x).split() if w.isupper()]))

## Number of title case words in the text ##
df["num_words_title"] = df["headline"].apply(lambda x: len([w for w in str(x).split() if w.istitle()]))

## Average length of the words in the text ##
df["mean_word_len"] = df["headline"].apply(lambda x: np.mean([len(w) for w in str(x).split()]))


# In[39]:


df.head()


# In[40]:


## Truncate some extreme values for better visuals ##
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()

df['num_words'].loc[df['num_words']>60] = 60 #truncation for better visuals
df['num_punctuations'].loc[df['num_punctuations']>10] = 10 #truncation for better visuals
df['num_chars'].loc[df['num_chars']>350] = 350 #truncation for better visuals

df['num_words'].loc[df['num_words']>60] = 60 #truncation for better visuals
df['num_punctuations'].loc[df['num_punctuations']>10] = 10 #truncation for better visuals
df['num_chars'].loc[df['num_chars']>350] = 350 #truncation for better visuals

f, axes = plt.subplots(3, 1, figsize=(10,20))
sns.boxplot(x='is_sarcastic', y='num_words', data=df, ax=axes[0])
axes[0].set_xlabel('is_sarcastic', fontsize=12)
axes[0].set_title("Number of words in each class", fontsize=15)

sns.boxplot(x='is_sarcastic', y='num_chars', data=df, ax=axes[1])
axes[1].set_xlabel('is_sarcastic', fontsize=12)
axes[1].set_title("Number of characters in each class", fontsize=15)

sns.boxplot(x='is_sarcastic', y='num_punctuations', data=df, ax=axes[2])
axes[2].set_xlabel('is_sarcastic', fontsize=12)
plt.ylabel('Number of punctuations in text', fontsize=12)
axes[2].set_title("Number of punctuations in each class", fontsize=15)
plt.show()


# Lets visualize number of words and each word length in dataset

# In[41]:
```

```
#number of words
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,5))
text_len=df[df['is_sarcastic']==1]['headline'].str.split().map(lambda x: len(x))
ax1.hist(text_len,color='red')
ax1.set_title('Sarcastic text')
text_len=df[df['is_sarcastic']==0]['headline'].str.split().map(lambda x: len(x))
ax2.hist(text_len,color='green')
ax2.set_title('Not Sarcastic text')
fig.suptitle('Words in texts')
plt.show()

#average word length
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,5))
word=df[df['is_sarcastic']==1]['headline'].str.split().apply(lambda x : [len(i) for i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax1,color='red')
ax1.set_title('Sarcastic text')
word=df[df['is_sarcastic']==0]['headline'].str.split().apply(lambda x : [len(i) for i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax2,color='green')
ax2.set_title('Not Sarcastic text')
fig.suptitle('Average word length in each text')


# #### Lets split dataset into train and test

# In[43]:


X = df.headline
Y = df.is_sarcastic
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state = 0)


# # Model Building

# ### Logistic Regression

# In[44]:


from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

logreg = Pipeline([('vect', CountVectorizer()),
                   ('tfidf', TfidfTransformer()),
                   ('clf', LogisticRegression(n_jobs=1, C=1e5)),
                  ])
logreg.fit(X_train, Y_train)


logreg_pred = logreg.predict(X_test)

print('accuracy %s' % accuracy_score(logreg_pred, Y_test))
print(classification_report(Y_test, logreg_pred,target_names=['sarcasm','non-sarcasm']))


# In[45]:
```

```
conmat=confusion_matrix(Y_test,logreg_pred)
conmat = pd.DataFrame(conmat , index = ['Not Sarcastic','Sarcastic'] , columns = ['Not Sarcastic','Sarcastic'])
plt.figure(figsize = (5,5))
sns.heatmap(conmat,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot = True, fmt='' , xticklabels = ['Not
        Sarcastic','Sarcastic'] , yticklabels = ['Not Sarcastic','Sarcastic']);


# ## SVM

# In[46]:


svm = Pipeline([('vect', CountVectorizer()),
          ('tfidf', TfidfTransformer()),
          ('clf', SVC()),
         ])
svm.fit(X_train, Y_train)


svm_pred = svm.predict(X_test)

print('accuracy %s' % accuracy_score(svm_pred, Y_test))
print(classification_report(Y_test, svm_pred,target_names=['sarcasm','non-sarcasm']))


# In[47]:


conmat=confusion_matrix(Y_test,svm_pred)
conmat = pd.DataFrame(conmat , index = ['Not Sarcastic','Sarcastic'] , columns = ['Not Sarcastic','Sarcastic'])
plt.figure(figsize = (5,5))
sns.heatmap(conmat,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot = True, fmt='' , xticklabels = ['Not
        Sarcastic','Sarcastic'] , yticklabels = ['Not Sarcastic','Sarcastic']);


# ### MultinomialNB

# In[48]:


nb = Pipeline([('vect', CountVectorizer()),
          ('tfidf', TfidfTransformer()),
          ('clf', MultinomialNB()),
         ])
nb.fit(X_train, Y_train)

from sklearn.metrics import classification_report
nb_pred = nb.predict(X_test)

print('accuracy %s' % accuracy_score(nb_pred, Y_test))
print(classification_report(Y_test, nb_pred,target_names=['sarcasm','non-sarcasm']))


# In[49]:


conmat=confusion_matrix(Y_test,nb_pred)
conmat = pd.DataFrame(conmat , index = ['Not Sarcastic','Sarcastic'] , columns = ['Not Sarcastic','Sarcastic'])
plt.figure(figsize = (5,5))
```

```
sns.heatmap(conmat,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot = True, fmt='' , xticklabels = ['Not
        Sarcastic','Sarcastic'] , yticklabels = ['Not Sarcastic','Sarcastic']);


# ### LETS BUILD DEEP LEARNING MODEL AND TRAIN

# ### Prepare training and testing data for deeplearning model

# Now we will use word2vec, tokenization, padding using keras text and word preprocessing libraries.

# In[50]:


#lets convert our text data into more acceptable format
#split words from a sentence and keep is sentence in the list which will help us for tokenization
words = []
for i in df.headline.values:
    words.append(i.split())
print("splitted words:",words[:5])

# use genism lib for word2vec wordembedding
import gensim
#Dim for max embedding
EMBEDDING_DIM = 200
#lets create word2vec model
w2v_model = gensim.models.Word2Vec(sentences = words , size=EMBEDDING_DIM , window = 5 , min_count
        = 1)


# now we will use tokenizer which keep tracks of every word in dataset by assigning an unique token for each word.
        Also to match length of each sentence, padding can be used
#

# In[51]:


# import keras.preprocessing lib for token
tokenizer = text.Tokenizer(num_words=38071)
tokenizer.fit_on_texts(words)
tokenized_traindata = tokenizer.texts_to_sequences(words)
x = sequence.pad_sequences(tokenized_traindata, maxlen = 20)


# In[52]:


x_train, x_test, y_train, y_test = train_test_split(x, df.is_sarcastic , test_size = 0.20 , random_state = 0)


# In[53]:


print("before tokenization:",len(w2v_model.wv.vocab))
#  vocab size increases by 1
vocab_size = len(tokenizer.word_index) + 1
print("after tokenization, vocab_size:", vocab_size)


# here we see after tokenization, size increases by 1 because of extra index for unknown words
```

```
# #### Lets create word vectors by creating weight matrix for non-embedding keras layers

# In[54]:


#generate weightmatrix using numpy zeros
weight_matrix=np.zeros((vocab_size, EMBEDDING_DIM))
#lets fill each zeros with value model
for word, k in tokenizer.word_index.items():
        weight_matrix[k] = w2v_model[word]


# In[55]:


#define dnn model
model = Sequential()
#adding embeddidng layers using bidirectional LSTM
model.add(Embedding(vocab_size, output_dim=EMBEDDING_DIM, weights=[weight_matrix], input_length=20,
            trainable=True))

model.add(Bidirectional(LSTM(units=128 , recurrent_dropout = 0.2 , dropout = 0.2,return_sequences = True)))
model.add(Bidirectional(GRU(units=64 , recurrent_dropout = 0.1 , dropout = 0.1)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=keras.optimizers.Adam(lr = 0.0001), loss='binary_crossentropy', metrics=['acc'])

model.summary()


# In[58]:


#Lets train our model
from keras.callbacks import EarlyStopping
history = model.fit(x_train, y_train, batch_size = 100 , validation_split =0.1 , epochs =
            5,callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)])


# In[60]:


# Lets analyze training and validation accuracy/loss
epochs = [i for i in range(4)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['acc']
train_loss = history.history['loss']
val_acc = history.history['val_acc']
val_loss = history.history['val_loss']
fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Testing Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'ro-' , label = 'Testing Loss')
ax[1].set_title('Training & Testing Loss')
```

```
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()


# In[69]:


#predict values over x_test data
predict = model.predict_classes(x_test)


# In[68]:


predict[:10]


# In[62]:

#lets use confusion matrix to check on predictions
conmat=confusion_matrix(y_test,predict)
conmat = pd.DataFrame(conmat , index = ['Not Sarcastic','Sarcastic'] , columns = ['Not Sarcastic','Sarcastic'])
plt.figure(figsize = (5,5))
sns.heatmap(conmat,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot = True, fmt='' , xticklabels = ['Not
            Sarcastic','Sarcastic'] , yticklabels = ['Not Sarcastic','Sarcastic']);


# In[63]:


#check classification report
print(classification_report(y_test, predict, target_names = ['Not Sarcastic','Sarcastic']))


# In[64]:


# get index of values of getting headline
r=y_test.index


# In[65]:


# Create a funcation to check test data is sarcastic or non-sarcastic
def sarcasm(a,b):
    for i,j in zip(a,b):
        print(df.headline[j])
        result = i
        if (result[0]==1):
            pos=1
        else:
            pos=0
        sentiment_dict = {0:'No Sarcasm',1:'Sarcasm'}
        print(sentiment_dict[pos] +"\n" )
    return
```

In[67]:

```
# check first 50 value of test data
sarcasm(predict[:50],r[:50])
```

```
# ## Thank you for review
#                                                    submitted by: Arun Kumar
#                                                    arunk.recs.cse@gmail.com
```

Chapter 4

Codes

4.2 R Code

```
# clear the environment
rm(list= ls())
# check and set working directory
getwd()
setwd(getwd())
# loading required libraries

require('tm')          # for text mining
require('data.table')   # for faster data operations
library("keras")
library('jsonlite')
library('tidyverse')
require('wordcloud')
require('ggplot2')
require("caTools")
require("e1071")

# read in our json file (one object person)
headlines <- file("Sarcasm_Headlines_Dataset.json", open = "r") %>%
  stream_in() %>%as_data_frame()


# checking structure and summary
dim(headlines)
str(headlines)
summary(headlines)

head(headlines)
tail(headlines)

# vec2clean.corp function takes two arguments: x = vector to be cleaned, y = document number to show the process
by printing
vec2clean.corp <- function(x,y=NULL){

  # As there are many languages used in the data, we consider stopwords of all the languages
  a = c(stopwords("english"))

  # Function to replace ' and " to spaces before removing punctuation to avoid different words from binding
  AposToSpace = function(x){
```

```
   x= gsub("""", ' ', x)
   x= gsub("'", ' ', x)
   x =gsub('break','broke',x) # break may interrupt control flow in few functions
   return(x)
  }

  x = Corpus(VectorSource(x))
  print(x$content[[y]])
  x = tm_map(x, tolower)
  print(x$content[[y]])
  x = tm_map(x, removeNumbers)
  print(x$content[[y]])
  x = tm_map(x, removeWords, a)
  print(x$content[[y]])
  x = tm_map(x, AposToSpace)
  print(x$content[[y]])
  x = tm_map(x, removePunctuation)
  print(x$content[[y]])
  x = tm_map(x, stemDocument)
  print(x$content[[y]])
  x = tm_map(x, stripWhitespace)
  print(x$content[[y]])

  return(x)

}


# Calling the vec2clean.corp with headlines(x) and we desire to check the progress with document 3(y)

corp <- vec2clean.corp(headlines$headline,37)
corp

# Creating Document Term Matrix from the corp with TF weighting
dtm <- DocumentTermMatrix(corp, control =
                          list(weighting = weightTf))
dtm

# dtm has (documents: 28619, terms: 19087) with 100% sparsity


# Removing Sparse term and take out those words which are more relevant
sparse.dtm <- removeSparseTerms(dtm, 0.999 )
sparse.dtm
# sparse.dtm has (documents: 28619, terms: 1484)

# converting Document term matrix to a Data frame
df <- data.frame(as.matrix(sparse.dtm))


## creating wordcloud for entire data set

sparse.dtm <- removeSparseTerms(dtm, 0.999 )
sparse.dtm
```

```
# creating word frequency data frame
word.freq <- sort(colSums(data.table(as.matrix(sparse.dtm))), decreasing = T)
word.freq <- data.table(Terms = names(word.freq), frequency = word.freq)

# creating word cloud for top 200 words in frequency
wordcloud(word.freq$Terms, word.freq$frequency,max.words = 150, scale = c(4,0.75),
          random.order = F, colors=brewer.pal(8, "Dark2"))
a <- word.freq[frequency>200]

# Bar graph for words that are more frequent appearing more than 2000 times
ggplot(a, aes(Terms, frequency))+
  geom_bar(stat = 'identity', colour = '#041838', fill = '#0b439e')+
  labs(title= 'Alphabetical ordered High frequent Terms')
#clearing graphical memory
dev.off()

## Seperate Word clouds for sarcastic and non-sarcastic news
# subsetting sarcasm and non-sarcasm news
library(dplyr)
sarcasm     <- filter(headlines,is_sarcastic == 1)
sarcasm.not <- filter(headlines,is_sarcastic == 0)

# corpus for both sarcastic and non-sarcastic

corp.sarcasm = vec2clean.corp(sarcasm$headline,5)
# Wordcloud for Sarcasm subset
wordcloud(corp.sarcasm, min.freq = 300, max.words = 300,
          random.order = F, scale = c(5 ,0.75),  colors=brewer.pal(8, "Dark2"))

corp.sarcasm.not = vec2clean.corp(sarcasm.not$headline, 5)
# Wordcloud for Non Sarcasm subset
wordcloud(corp.sarcasm.not, min.freq = 200, max.words = 300,
          random.order = F, scale = c(5 ,0.75),  colors=brewer.pal(8, "Dark2"))

# DTM for sarcasm and non-sarcasm corpora
dtm.sar <- DocumentTermMatrix(corp.sarcasm)
dtm.non <- DocumentTermMatrix(corp.sarcasm.not)

# Most frequent 30 words in Sarcasm and non-sarcasm DTMs
findFreqTerms(dtm.sar)[seq(30)]
findFreqTerms(dtm.non)[seq(30)]

# Finding out the most common words and frequent words
# These words should be eliminated from the master DTM
common <- NULL
for(i in findFreqTerms(dtm.non)[seq(100)]){
  for(j in findFreqTerms(dtm.sar)[seq(100)]){
    if(identical(i,j)){
      common = c(common,i)
      print(i)
    }
  }
}
common
```

```
headlines$is_sarcastic <- as.factor(headlines$is_sarcastic)

#remove sparsity and prepare data frame
sparse <- removeSparseTerms(dtm, 0.9992)
sparse


df <- data.table(as.matrix(sparse))
# Find associations
unlist(findAssocs(sparse, findFreqTerms(sparse,100),corlimit = 0.5))
# this may not provide all correlated pairs
rm(dtm,sparse) # as it is not necessary

# we go for pearson correlation matrix to get more detailed info
corr <- data.table(cor(df, use = "complete.obs", method= "pearson"))
corr.terms <- NULL
for(i in 1:(nrow(corr)-1)){
  for(j in (i+1):ncol(corr)){
    if((abs(corr[[i,j]])>0.49) ==T){
      corr.terms = c(corr.terms, names(corr)[i])
      print(paste(colnames(corr)[i],',',colnames(corr)[j])) # print rows and column numbers which are correlated
    }
  }
}
# corr.terms consist of correlated terms which are more than 50% with any other variable
# only one term out correlated pair is added while 'for' loop
rm(corr,i,j)
corr.terms

# combining both common and del.words
del.words <- c(corr.terms, common)
del.words <- unique(del.words)

# removing del.words features from master
df[, (del.words) := NULL]
dim(df)

# creating master data set
master <- data.table(is_sarcastic = headlines$is_sarcastic, df)

# saving numeric data master for sampling
save(master, file='master.numeric.dat')

# We are going to prepare master.factor from master
rm(list = ls())
# loading numeric master
load('master.numeric.dat')
master.factor <- as.data.frame(master)

#Binning
master.factor <- data.frame(lapply(master[,2:ncol(master)], function(x){ifelse(x==0,0,1)}))

# Converting numericals to factors
master.factor <- data.frame(lapply(master.factor, as.factor))
# master.factor has all categorical variables 0 and 1 factors
```

```
master.factor <- cbind(is_sarcastic = master$is_sarcastic, master.factor)

sample2train.test <- function(master.x, seed.x, samp.ratio= 0.055, train.ratio= 0.8){
  set.seed(seed = seed.x)
  samp.split = sample.split(master.x$is_sarcastic, samp.ratio)
  sample = subset(master.x, samp.split == T)

  # training and testing
  spl = sample.split(sample$is_sarcastic, train.ratio)
  train.x = subset(sample, spl == T )
  test.x  = subset(sample, spl == F )
  return(list(train.x, test.x))
}


# choose the data set you like to use and load the data set



# pass the desired master data set, seed(to produce random sample), sample ratio and train ratio
# sample ratio and train ratio are defaulted with 0.055(5022 observations) and 0.8 respectively

# For producing Training and Testing sets of master.numeric
Train.Test.list <- sample2train.test(master,123)

train.num <- Train.Test.list[[1]]
test.num  <- Train.Test.list[[2]]

# saving numeric train and test sets
save(train.num, test.num, file = 'TrainTest_num.dat')

# For producing Training and Testing sets of master.factor
Train.Test.list <- sample2train.test(master.factor,123)

train <- Train.Test.list[[1]]
test  <- Train.Test.list[[2]]
# Naive Bayes model classic
n.model <- naiveBayes(label~ ., data = train)


# Naive Bayes model with laplace estimator 1
# laplace = 1 ensures a non-zero probability for every feature
n.model.lap <- naiveBayes(label~ ., data = train, laplace = 1)
# Predicting the test target class
# for classic Naive Bayes model
n.pred <- predict(n.model, test[,-1], type = 'class')

xtab.n <- table('Actual class' = test[,1], 'Predicted class' = n.pred )
caret::confusionMatrix(xtab.n)

# for robust Naive Bayes model with laplace estimator
n.pred.lap <- predict(n.model.lap, test[,-1], type = 'class')

# Logistics Regression
```

```
# Logistics Regression
glm.fit <- glm(as.factor(label~.), data = train, family = "binomial")

xtab.lap <- table('Actual class' = test[,1], 'Predicted class' = n.pred.lap )
caret::confusionMatrix(xtab.lap)

# loading required package
require(h2o) # to implement random forest quick

# Initializing h2o cluster
h2o.init(nthreads = -1)

#check h2o cluster status
h2o.init()

set.seed(123)
# loading data to h2o clusters
h.train.num <- as.h2o(train.num)
h.train     <- as.h2o(train)


# creating predictor and target indices
x <- 2:ncol(train)
y <- 1
# Building random forest model on numeric data
rf.model.num <- h2o.randomForest(x=x, y=y, training_frame = h.train.num, ntrees = 1000)

# Building random forest model on factor data
rf.model     <- h2o.randomForest(x=x, y=y, training_frame = h.train, ntrees = 1000)

# Evaluating random forest models

# Random forest evaluation for Numeric data
pred.num <- as.data.frame(h2o.predict(rf.model.num, h.test.num))
caret::confusionMatrix(table('Actual class' = test$label,'Predicted class' =  pred.num$predict))

# Random forest evaluation for Factor data
pred <- as.data.frame(h2o.predict(rf.model, h.test))
caret::confusionMatrix(table('Actual class' = test$label, 'Predicted class' = pred$predict))

#shuting down h2o cluster
h2o.shutdown(prompt = F)

# set seed
set.seed(42)

# hyperparmeters
input_maxlen = 20
batch_size = 64
epochs = 10

# preprocessing!

# tokenization
tokenizer <- text_tokenizer(num_words = 50000)
```

```
tokenizer %>%
  fit_text_tokenizer(headlines$headline)
headline <- texts_to_sequences(tokenizer, headlines$headline)

# pad/chop all input sequecnes to 20
headline_padded <- pad_sequences(headline,
                                 maxlen = input_maxlen,
                                 value = 50000 + 1)


# testing/training split

# take a 10th of the data as validation data
val_sample <- sample.int(nrow(headline_padded),
                         size = 0.1*nrow(headline_padded))

# training & validation data
train_headline_padded <- headline_padded[-val_sample,]
train_is_sarcastic <- headlines$is_sarcastic[-val_sample]

val_headline_padded <- headline_padded[val_sample,]
val_is_sarcastic <- headlines$is_sarcastic[val_sample]

# input layer
input <- layer_input(shape = c(input_maxlen), name = "input_headline")

# embedding layer
word_embedder <- layer_embedding(
  input_dim = 50000 + 2, # vocab size + UNK token + padding value
  output_dim = 128,      # hyperparameter - embedding size
  input_length = input_maxlen,     # padding size,
  embeddings_regularizer = regularizer_l2(0.0001) # hyperparameter - regularization
)

# lstm sequence embedder
seq_embedder <- layer_lstm(
  units = 128, # hyperparameter -- sequence embedding size
  kernel_regularizer = regularizer_l2(0.0001) # hyperparameter - regularization
)

# final output layer (specifying complete structure)
output <- input %>%
  word_embedder() %>%
  seq_embedder() %>%
  layer_dense(units = 1, activation = "sigmoid")

# model specificiation
model <- keras_model(input,output)
model %>% compile(
  optimizer = "adam",
  metrics = c('accuracy'), #binary accuracy don't work
  loss = "binary_crossentropy"
)

# check out our compiled model! :)
```

```
summary(model)


# Fit model to data
history <- model %>% fit(
  train_headline_padded,
  train_is_sarcastic,
  batch_size = batch_size,
  epochs = epochs,
  verbose = 1,
  validation_split = 0.2
)


model.
plot(history)
```

Chapter 5

Reference

These are the refrances that I have used during my project work.

https://stackoverflow.com/
https://www.kaggle.com/
https://www.edwisor.com/
https://machinelearningmastery.com/
https://towardsdatascience.com/