Sentiment Analysis for Twitter Data

# Getting Ready for Analysis[¶](#Getting-Ready-for-Analysis)

Run the below cell to install all the required packages for this notebook.

In [1]:

!pip3 install -r requirements.txt

Defaulting to user installation because normal site-packages is not writeable  
Requirement already satisfied: emoji>=0.5.4 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 1)) (0.5.4)  
Requirement already satisfied: numpy>=1.17.1 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 2)) (1.17.1)  
Requirement already satisfied: pandas>=0.25.3 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 3)) (0.25.3)  
Requirement already satisfied: textblob>=0.15.3 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 4)) (0.15.3)  
Requirement already satisfied: scikit-learn>=0.22.1 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 5)) (0.22.1)  
Requirement already satisfied: vaderSentiment>=3.3.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 6)) (3.3.2)  
Requirement already satisfied: matplotlib>=3.1.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 7)) (3.1.2)  
Requirement already satisfied: nltk>=3.4.5 in /home/anurag/.local/lib/python3.6/site-packages (from -r requirements.txt (line 8)) (3.4.5)  
Requirement already satisfied: wordcloud>=1.7.0 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 9)) (1.7.0)  
Requirement already satisfied: pytz>=2017.2 in /home/anurag/.local/lib/python3.6/site-packages (from pandas>=0.25.3->-r requirements.txt (line 3)) (2019.3)  
Requirement already satisfied: python-dateutil>=2.6.1 in /home/anurag/.local/lib/python3.6/site-packages (from pandas>=0.25.3->-r requirements.txt (line 3)) (2.8.1)  
Requirement already satisfied: scipy>=0.17.0 in /home/anurag/.local/lib/python3.6/site-packages (from scikit-learn>=0.22.1->-r requirements.txt (line 5)) (1.4.1)  
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn>=0.22.1->-r requirements.txt (line 5)) (0.14.1)  
Requirement already satisfied: requests in /home/anurag/.local/lib/python3.6/site-packages (from vaderSentiment>=3.3.2->-r requirements.txt (line 6)) (2.22.0)  
Requirement already satisfied: cycler>=0.10 in /home/anurag/.local/lib/python3.6/site-packages (from matplotlib>=3.1.2->-r requirements.txt (line 7)) (0.10.0)  
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /home/anurag/.local/lib/python3.6/site-packages (from matplotlib>=3.1.2->-r requirements.txt (line 7)) (2.4.5)  
Requirement already satisfied: kiwisolver>=1.0.1 in /home/anurag/.local/lib/python3.6/site-packages (from matplotlib>=3.1.2->-r requirements.txt (line 7)) (1.1.0)  
Requirement already satisfied: six in /home/anurag/.local/lib/python3.6/site-packages (from nltk>=3.4.5->-r requirements.txt (line 8)) (1.13.0)  
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from wordcloud>=1.7.0->-r requirements.txt (line 9)) (6.2.0)  
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /home/anurag/.local/lib/python3.6/site-packages (from requests->vaderSentiment>=3.3.2->-r requirements.txt (line 6)) (3.0.4)  
Requirement already satisfied: idna<2.9,>=2.5 in /home/anurag/.local/lib/python3.6/site-packages (from requests->vaderSentiment>=3.3.2->-r requirements.txt (line 6)) (2.8)  
Requirement already satisfied: certifi>=2017.4.17 in /home/anurag/.local/lib/python3.6/site-packages (from requests->vaderSentiment>=3.3.2->-r requirements.txt (line 6)) (2019.11.28)  
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /home/anurag/.local/lib/python3.6/site-packages (from requests->vaderSentiment>=3.3.2->-r requirements.txt (line 6)) (1.24.3)  
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotlib>=3.1.2->-r requirements.txt (line 7)) (41.2.0)  
WARNING: You are using pip version 20.1; however, version 20.1.1 is available.  
You should consider upgrading via the '/usr/bin/python3 -m pip install --upgrade pip' command.

# Sentiment Analysis[¶](#Sentiment-Analysis)

I used two approaches in predicting the sentiment of the given data.

* Lexicon based Sentiment Method
* Machine Learning based Sentiment Method

In the lexicon-based method, I have used two third-party libraries to predict the sentiments.

* TextBlob
* vaderSentiment

Both of these methods used a pre-defined valance score of the words to predict tweets' sentiment.

Coming to Machine Learning based prediction, we cannot use any traditional method of training the model and predicting on test data.

But we are provided with the unlabelled data. So, we cannot carry out the supervised learning strategy

So, I have used the IMDB Movie review dataset to train a neural network model and then use that model to predict the given tweets data.

This gave a very similar result compared to the lexicon-based methods.

**Why these modules?**

* *re:* It is used in preprocessing the data.
* *emoji:* To remove all the emoji's (part of preprocessing step)
* *numpy:* For doing some numerical calculations
* *pandas:* For loading the data into dataframe's
* *sklearn:* Splitting the dataset into train and test sets
* *tensorflow:* To build the machine learning
* *matplotlib:* For visualization of graphs and plots
* *textblob:* To compute lexicon based sentiment(method-1)
* *vaderSentiment:* To compute lexicon based sentiment(method-2)

In [2]:

# Required modules  
  
import re  
import emoji  
import numpy as np  
import pandas as pd  
import seaborn as sns  
import tensorflow as tf  
from matplotlib import pyplot as plt  
  
from textblob import TextBlob  
from sklearn.metrics import confusion\_matrix  
from sklearn.model\_selection import train\_test\_split  
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer  
  
%matplotlib inline  
plt.rcParams['figure.figsize'] = (12, 7)

/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_qint8 = np.dtype([("qint8", np.int8, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:524: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_quint8 = np.dtype([("quint8", np.uint8, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_qint16 = np.dtype([("qint16", np.int16, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_quint16 = np.dtype([("quint16", np.uint16, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_qint32 = np.dtype([("qint32", np.int32, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:532: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 np\_resource = np.dtype([("resource", np.ubyte, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow\_stub/dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_qint8 = np.dtype([("qint8", np.int8, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow\_stub/dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_quint8 = np.dtype([("quint8", np.uint8, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow\_stub/dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_qint16 = np.dtype([("qint16", np.int16, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow\_stub/dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_quint16 = np.dtype([("quint16", np.uint16, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow\_stub/dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 \_np\_qint32 = np.dtype([("qint32", np.int32, 1)])  
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow\_stub/dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.  
 np\_resource = np.dtype([("resource", np.ubyte, 1)])

In [3]:

# Loading the data  
  
data = pd.read\_csv('./data/tweets.zip', compression='zip')  
data.head()

Out[3]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | id | conversation\_id | created\_at | date | time | timezone | user\_id | username | name | place | ... | geo | source | user\_rt\_id | user\_rt | retweet\_id | reply\_to | retweet\_date | translate | trans\_src | trans\_dest |
| 0 | 1262787913311387649 | 1262787913311387649 | 1589907074000 | 2020-05-19 | 16:51:14 | UTC | 1250079805980045318 | dramaflick | The Drama Flick | NaN | ... | NaN | NaN | NaN | NaN | NaN | [{'user\_id': '1250079805980045318', 'username'... | NaN | NaN | NaN | NaN |
| 1 | 1262787786152620040 | 1262787786152620040 | 1589907044000 | 2020-05-19 | 16:50:44 | UTC | 807843238648299520 | knowpuneet | TravelTrainee | NaN | ... | NaN | NaN | NaN | NaN | NaN | [{'user\_id': '807843238648299520', 'username':... | NaN | NaN | NaN | NaN |
| 2 | 1262787219498000384 | 1262787219498000384 | 1589906909000 | 2020-05-19 | 16:48:29 | UTC | 1085426639570235392 | narasinhpurohit | Narasinh Purohit | NaN | ... | NaN | NaN | NaN | NaN | NaN | [{'user\_id': '1085426639570235392', 'username'... | NaN | NaN | NaN | NaN |
| 3 | 1262786998592434176 | 1262786998592434176 | 1589906856000 | 2020-05-19 | 16:47:36 | UTC | 1104213868467806208 | ka\_trolls | Humans Of Hindutva | NaN | ... | NaN | NaN | NaN | NaN | NaN | [{'user\_id': '1104213868467806208', 'username'... | NaN | NaN | NaN | NaN |
| 4 | 1262786970163441669 | 1262786970163441669 | 1589906849000 | 2020-05-19 | 16:47:29 | UTC | 392180204 | rajendrabohora | rajendrabohora | NaN | ... | NaN | NaN | NaN | NaN | NaN | [{'user\_id': '392180204', 'username': 'rajendr... | NaN | NaN | NaN | NaN |

5 rows × 34 columns

In [4]:

# Inspecting the data  
  
data.info()  
data.describe()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 124384 entries, 0 to 124383  
Data columns (total 34 columns):  
id 124384 non-null int64  
conversation\_id 124384 non-null int64  
created\_at 124384 non-null int64  
date 124384 non-null object  
time 124384 non-null object  
timezone 124384 non-null object  
user\_id 124384 non-null int64  
username 124384 non-null object  
name 124384 non-null object  
place 9143 non-null object  
tweet 124384 non-null object  
mentions 124384 non-null object  
urls 124384 non-null object  
photos 124384 non-null object  
replies\_count 124384 non-null int64  
retweets\_count 124384 non-null int64  
likes\_count 124384 non-null int64  
hashtags 124384 non-null object  
cashtags 124384 non-null object  
link 124384 non-null object  
retweet 124384 non-null bool  
quote\_url 13033 non-null object  
video 124384 non-null int64  
near 0 non-null float64  
geo 0 non-null float64  
source 0 non-null float64  
user\_rt\_id 0 non-null float64  
user\_rt 0 non-null float64  
retweet\_id 0 non-null float64  
reply\_to 124384 non-null object  
retweet\_date 0 non-null float64  
translate 0 non-null float64  
trans\_src 0 non-null float64  
trans\_dest 0 non-null float64  
dtypes: bool(1), float64(10), int64(8), object(15)  
memory usage: 31.4+ MB

We can see that most of the columns in the dataset are empty, so dropping these columns is better.

## Dealing with Missing values[¶](#Dealing-with-Missing-values)

In [5]:

nan\_cols = data.columns[data.isna().any()]  
print("Columns which contains missing values: ")  
nan\_cols

Columns which contains missing values:

In [6]:

print("Percentage of Missing values in the columns: ")  
(data[nan\_cols].isna().sum()) / len(data)

Percentage of Missing values in the columns:

In [7]:

# Dropping all the columns which consists of missing values  
  
data.drop(nan\_cols, axis=1, inplace=True)

In [8]:

print(data.columns)

Index(['id', 'conversation\_id', 'created\_at', 'date', 'time', 'timezone',  
 'user\_id', 'username', 'name', 'tweet', 'mentions', 'urls', 'photos',  
 'replies\_count', 'retweets\_count', 'likes\_count', 'hashtags',  
 'cashtags', 'link', 'retweet', 'video', 'reply\_to'],  
 dtype='object')

By careful inspection, we can observe that all the columns except 'tweet' and 'hashtags' are redundant and can be dropped.

In [9]:

actual\_data = data[['tweet', 'hashtags']]  
actual\_data

Out[9]:

|  |  |  |
| --- | --- | --- |
|  | tweet | hashtags |
| 0 | https://www.youtube.com/watch?v=-CRbO7ExO1k …... | ['#lockdownindia', '#lockdown', '#indiafightsc... |
| 1 | Lockdown 4.0 ka.naam hi lockdown hai\nHai sab ... | ['#lockdownindia', '#locldown4'] |
| 2 | CORONA VIRUS THREAT-\nHOW TO OVERCOME STRESS A... | ['#covid\_19', '#covid\_19sa', '#covid\_19india',... |
| 3 | Could you please\n\n#lockdownindia\n@Bhuvan\_Ba... | ['#lockdownindia', '#roastchallenge', '#journa... |
| 4 | In fight with #COVID19, You are the best Docto... | ['#covid19', '#coronavirus', '#patiencechallen... |
| ... | ... | ... |
| 124379 | I pledge to follow the appeal given by Hon'ble... | ['#staysafestayhome', '#janta\_curfew', '#janta... |
| 124380 | Four new cases of Coronavirus detected in Luck... | ['#coronaindia', '#coronavirusoutbreakindia', ... |
| 124381 | Do you sometimes feel a tingling #sensation or... | ['#sensation', '#hands', '#thevoiceofwoman', '... |
| 124382 | Some Time we have to Stay Back ... Just to Sav... | ['#stayback', '#gobackcorona', '#coronafighter... |
| 124383 | Sir, National Medical Emergency should declare... | ['#coronaindia', '#coronavirusupdate', '#wewil... |

124384 rows × 2 columns

## Preprocessing data[¶](#Preprocessing-data)

In this, preprocessing step I have removed all the unneccessary information using the regular expression(re) modules like:

re.sub(r'(#\w+)', ' ', x) # to remove all the hashtags in the text.

re.sub(r'(\n)', ' ', x) # to remove all the newline characters in the text.

re.sub(r'(@\w+)', ' ', x) # to remove all the handles in the text.

re.sub(r'(…)', ' ', x) # to remove '…' from the text.

re.sub(r'(pic.twitter.com/\w+)', ' ', x) # to remove all pic URLS in the text.

re.sub(r'([^a-zA-Z0-9])\1+', r'\1', x) # to remove all repeating characters except alphabets and numbers.

re.sub(r'(http|https|ftp)://[a-zA-Z0-9\\.\?\-\=/]+\xa0', ' ', x) # to remove all URLS in the text.

In [10]:

# Removing all the unwanted information from of data  
  
actual\_data['tweet'] = actual\_data['tweet'].apply(lambda x: re.sub(r'#\w+', ' ', x))   
actual\_data['tweet'] = actual\_data['tweet'].apply(lambda x: re.sub(r'\n', ' ', x))   
actual\_data['tweet'] = actual\_data['tweet'].apply(lambda x: re.sub(r'pic.twitter.com/\w+', ' ', x))   
actual\_data['tweet'] = actual\_data['tweet'].apply(lambda x: re.sub(r'@\w+', ' ', x))  
actual\_data['tweet'] = actual\_data['tweet'].apply(lambda x: re.sub(r'(http|https|ftp)://[a-zA-Z0-9\\.\?\-\=/]+\xa0', ' ', x))  
actual\_data['tweet'] = actual\_data['tweet'].apply(lambda x: re.sub(r'…', ' ', x))   
actual\_data['tweet'] = actual\_data['tweet'].apply(lambda x: re.sub(r'([^a-zA-Z0-9])\1+', r'\1', x))

/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:3: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 This is separate from the ipykernel package so we can avoid doing imports until  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:4: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 after removing the cwd from sys.path.  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 """  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:6: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
   
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:7: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 import sys  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:8: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
   
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:9: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 if \_\_name\_\_ == '\_\_main\_\_':

data1, data2, data3 all are the copies of the actual\_data dataframe's

* data1 is used in sentiment prediction using textblob.
* data2 is used in sentiment prediction using vaderSentiment.
* data3 is used in sentiment prediction using machine learning method.

In [11]:

# Making a copy of original dataset to be used for different strategy  
  
data1 = actual\_data.copy()  
data2 = actual\_data.copy()  
data3 = actual\_data.copy()

## Lexicon based sentiment analysis[¶](#Lexicon-based-sentiment-analysis)

### TextBlob[¶](#TextBlob)

This sentiment analysis is done using TextBlob function of the module textblob.

The output of the function is a 2-element tuple which has a structure (polarity\_value, subjectivity).

In [12]:

data1['sentiment'] = data1['tweet'].apply(lambda x: TextBlob(x).sentiment)

In [13]:

data1['polarity\_value'] = data1['sentiment'].apply(lambda x: x[0])  
data1['subjectivity'] = data1['sentiment'].apply(lambda x: x[1])

In [14]:

data1['polarity\_type'] = np.where(data1['polarity\_value'].values > 0, 'Positive', np.where(data1['polarity\_value'].values < 0, 'Negative', 'Neutral'))

In [15]:

# Count of each type of tweets  
  
data1['polarity\_type'].value\_counts()

Out[15]:

Positive 54165  
Neutral 47780  
Negative 22439  
Name: polarity\_type, dtype: int64

#### Visualization of data[¶](#Visualization-of-data)

First plot, contains the Number of tweets for each type of tweets.

Second plot, contains the distribution of sentiment of all the tweets on the scale of [-1, 1](-1 being most negative, 1 being most positive).

In [16]:

# Polarity count  
  
\_, ax = plt.subplots()  
ax.set\_title("Value Counts", fontsize=20)  
ax.set\_xlabel("Polarity Type", fontsize=15)  
ax.set\_ylabel("Count", fontsize=15)  
ax.bar(data1['polarity\_type'].unique(), height=data1['polarity\_type'].value\_counts())

Out[16]:

<BarContainer object of 3 artists>

In [17]:

# Distribution of the polarity value  
  
\_, ax = plt.subplots()  
ax.set\_title("Distribution of polarity", fontsize=20)  
ax.set\_xlabel("Range [-1, 1]", fontsize=15)  
ax.set\_ylabel("No. of tweets", fontsize=15)  
  
ax.hist(data1['polarity\_value'], bins=20)

Out[17]:

(array([3.2400e+02, 5.0000e+01, 2.7300e+02, 3.4700e+02, 4.8100e+02,  
 2.1510e+03, 1.8610e+03, 3.3110e+03, 6.7980e+03, 6.8430e+03,  
 5.9218e+04, 1.3030e+04, 8.6690e+03, 6.9420e+03, 2.3010e+03,  
 7.0940e+03, 1.8880e+03, 4.6600e+02, 1.2830e+03, 1.0540e+03]),  
 array([-1. , -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0. ,  
 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),  
 <a list of 20 Patch objects>)

### VaderSentiment[¶](#VaderSentiment)

This sentiment analysis is done using VaderSentiment module.

The output of the function is a dictionary which contains four keys 'positive', 'negative', 'neutral', 'compound'

compound gives the overall sentiment of the tweet.

positive says how much positive is the given tweet.

negative says how much negative is the given tweet.

neutral says how much neutral is the given tweet.

In [18]:

analyzer = SentimentIntensityAnalyzer()  
  
data2['sentiment'] = data2['tweet'].apply(lambda x: analyzer.polarity\_scores(x))

In [19]:

data2['positive'] = data2['sentiment'].apply(lambda x: x['pos'])  
data2['negative'] = data2['sentiment'].apply(lambda x: x['neg'])  
data2['neutral'] = data2['sentiment'].apply(lambda x: x['neu'])  
  
data2['polarity\_value'] = data2['sentiment'].apply(lambda x: x['compound'])

In [20]:

data2['polarity\_type'] = np.where(data2['polarity\_value'].values > 0.05, 'Positive', np.where(data2['polarity\_value'].values < -0.05, 'Negative', 'Neutral'))

In [21]:

# Count of each type of tweets  
  
data2['polarity\_type'].value\_counts()

Out[21]:

Positive 59641  
Neutral 34188  
Negative 30555  
Name: polarity\_type, dtype: int64

#### Visulization of data[¶](#Visulization-of-data)

First plot, contains the Number of tweets for each type of tweets.

Second plot, contains the degree of degree of polarity of each polarity type.

In [22]:

# Polarity count  
  
\_, ax = plt.subplots()  
ax.set\_title("Value Counts", fontsize=20)  
ax.set\_xlabel("Polarity Type", fontsize=15)  
ax.set\_ylabel("Count", fontsize=15)  
ax.bar(data2['polarity\_type'].unique(), height=data2['polarity\_type'].value\_counts())

Out[22]:

<BarContainer object of 3 artists>

In [23]:

# Degree of polarity  
  
fig, ax = plt.subplots(nrows=1, ncols=3)  
fig.suptitle("Degree of polarity", fontsize=20)  
ax[0].set\_xlabel("Positive", fontsize=15)  
ax[1].set\_xlabel("Negative", fontsize=15)  
ax[2].set\_xlabel("Neutral", fontsize=15)  
ax[0].set\_ylabel("Count", fontsize=15)  
  
ax[0].hist(data2['positive'])  
ax[1].hist(data2['negative'])  
ax[2].hist(data2['neutral'])

Out[23]:

(array([ 228., 184., 712., 1327., 3573., 8682., 16542., 25040.,  
 27808., 40288.]),  
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),  
 <a list of 10 Patch objects>)

## Machine Learning based Sentiment Analysis[¶](#X0774a2cd3de7085487b7df449e3513e4a9c5d2d)

Using the imdb dataset to train and build the neural network, after that apply the neural network on the tweets data given.

In [24]:

# Loading the data  
  
imdb\_data = pd.read\_csv('./data/imdb\_data.zip', compression='zip')  
imdb\_data.head()

Out[24]:

|  |  |  |
| --- | --- | --- |
|  | review | sentiment |
| 0 | One of the other reviewers has mentioned that ... | positive |
| 1 | A wonderful little production. <br /><br />The... | positive |
| 2 | I thought this was a wonderful way to spend ti... | positive |
| 3 | Basically there's a family where a little boy ... | negative |
| 4 | Petter Mattei's "Love in the Time of Money" is... | positive |

In [25]:

# Inspecting data  
  
imdb\_data.info()  
imdb\_data.describe()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 50000 entries, 0 to 49999  
Data columns (total 2 columns):  
review 50000 non-null object  
sentiment 50000 non-null object  
dtypes: object(2)  
memory usage: 781.4+ KB

### Preprocessing[¶](#Preprocessing)

Removing all the html tags in the text.

In [26]:

# Some preprocssing  
# There are some html break tags(replacing them with ' ')  
  
imdb\_data['review'] = [re.sub(r'<.\*?>', ' ', review) for review in imdb\_data['review']]

In [27]:

# Adding few columns  
  
imdb\_data['review\_len'] = imdb\_data['review'].apply(lambda x: len(x.split()))

In [28]:

# Converting 'sentiment' into numerical value  
  
imdb\_data['sentiment\_numerical'] = np.where(imdb\_data['sentiment'] == 'positive', 1, 0)

In [29]:

# Separating out features and labels  
  
X = imdb\_data['review']  
y = imdb\_data['sentiment\_numerical']

In [30]:

# Train-Test Split  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=88)

In [31]:

max\_words = 10000  
oov\_token = '<OOV>'  
  
tokenizer = tf.keras.preprocessing.text.Tokenizer(num\_words=max\_words, oov\_token=oov\_token)

In [32]:

tokenizer.fit\_on\_texts(X)

In [33]:

# Converting tokens into Sequences  
  
X\_train\_tokens = tokenizer.texts\_to\_sequences(X\_train)  
X\_test\_tokens = tokenizer.texts\_to\_sequences(X\_test)

In [34]:

max\_train\_word = int(np.mean(imdb\_data['review\_len']) + 2 \* np.std(imdb\_data['review\_len']))

In [35]:

# Padding the input to make it uniform length  
  
pad\_type = 'pre'  
  
X\_train\_padded = tf.keras.preprocessing.sequence.pad\_sequences(X\_train\_tokens, maxlen=max\_train\_word, padding=pad\_type, truncating=pad\_type)  
X\_test\_padded = tf.keras.preprocessing.sequence.pad\_sequences(X\_test\_tokens, maxlen=max\_train\_word, padding=pad\_type, truncating=pad\_type)

***Model Architecture***

My model consists of 5 layers,

1. Embedding layer (with a dimension of 8)

* which finds the word embedding's of all the unique words in the text which are tokenized by the 'tokenizer' function.

1. LSTM (with an input dimension of 16)
2. LSTM (with an input dimension of 8)
3. LSTM (with an input dimension of 4)
4. Dense (this output, the sentiment)

For more information about the model you can see cell number 39.

***Hyperparameters***

The Hyperparameters for this model are

* Input dimension
* Ouput dimension
* Length of the input sequence
* Number of layers and there dimension's
* Number of epochs
* Batch Size

In [36]:

# Defining the model  
  
embedding\_dim = 8  
  
model = tf.keras.models.Sequential([  
 tf.keras.layers.Embedding(input\_dim=max\_words,  
 output\_dim=embedding\_dim,  
 input\_length=max\_train\_word,  
 name='embedding\_layer'),  
 tf.keras.layers.LSTM(units=16, return\_sequences=True),  
 tf.keras.layers.LSTM(units=8, return\_sequences=True),  
 tf.keras.layers.LSTM(units=4),  
 tf.keras.layers.Dense(1, activation='sigmoid')  
])

In [37]:

# Defining the optimizer  
  
optimizer = tf.keras.optimizers.Adam(lr=0.1)

In [38]:

model.compile(loss='binary\_crossentropy',  
 optimizer=optimizer,  
 metrics=['acc'])

In [39]:

model.summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
embedding\_layer (Embedding) (None, 568, 8) 80000   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
unified\_lstm (UnifiedLSTM) (None, 568, 16) 1600   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
unified\_lstm\_1 (UnifiedLSTM) (None, 568, 8) 800   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
unified\_lstm\_2 (UnifiedLSTM) (None, 4) 208   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 1) 5   
=================================================================  
Total params: 82,613  
Trainable params: 82,613  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [40]:

# Model fitting on the training data  
  
epochs=3  
batch\_size = 64  
  
history = model.fit(X\_train\_padded, y\_train, validation\_split=0.05, epochs=epochs, batch\_size=batch\_size)

Train on 35625 samples, validate on 1875 samples  
Epoch 1/3  
35625/35625 [==============================] - 509s 14ms/sample - loss: 0.5812 - acc: 0.6847 - val\_loss: 0.6976 - val\_acc: 0.5893  
Epoch 2/3  
35625/35625 [==============================] - 569s 16ms/sample - loss: 0.5110 - acc: 0.7490 - val\_loss: 0.4740 - val\_acc: 0.7685  
Epoch 3/3  
35625/35625 [==============================] - 640s 18ms/sample - loss: 0.4290 - acc: 0.8065 - val\_loss: 0.4418 - val\_acc: 0.8011

In [41]:

# Measuring Training Loss   
  
train\_loss = model.evaluate(X\_train\_padded, y\_train)  
print("Training accuracy: {:6f}".format(train\_loss[1]))

37500/37500 [==============================] - 130s 3ms/sample - loss: 0.3955 - acc: 0.8259s - loss: 0.3951 - acc:   
Training accuracy: 0.825867

In [42]:

# Measuring Testing Loss  
  
test\_loss = model.evaluate(X\_test\_padded, y\_test)  
print("Testing accuracy: {:6f}".format(test\_loss[1]))

12500/12500 [==============================] - 40s 3ms/sample - loss: 0.4173 - acc: 0.8118  
Testing accuracy: 0.811760

## Now introducing the original data[¶](#Now-introducing-the-original-data)

### Some preprocessing[¶](#Some-preprocessing)

Removing all the emoji's from the text and converting the text into lower case.

In [43]:

# Removing unnecessary spaces and emoji's   
  
data3['tweet'] = data3['tweet'].apply(lambda x: " ".join(map(str.lower, x.split())))  
data3['tweet'] = data3['tweet'].apply(lambda x: "".join([char for char in x if char not in emoji.UNICODE\_EMOJI]))

### Prediction using pretrained model[¶](#Prediction-using-pretrained-model)

In [44]:

# Applying trained model on tweets data  
  
X\_data\_tokens = tokenizer.texts\_to\_sequences(data3['tweet'])  
X\_data\_padded = tf.keras.preprocessing.sequence.pad\_sequences(X\_data\_tokens, maxlen=max\_train\_word, padding=pad\_type, truncating=pad\_type)

In [49]:

predicted = model.predict(X\_data\_padded)  
sentiment = np.where(predicted > 0.5, 'Positive', 'Negative')

In [50]:

final\_data = pd.DataFrame(data=np.append(predicted, sentiment, axis=1), columns=['Polarity\_value', 'Polarity\_type'])

### Visualization[¶](#Visualization)

In [51]:

# Some plots  
  
\_, ax = plt.subplots()  
ax.set\_xlabel("Polarity Type", fontsize=15)  
ax.set\_ylabel("Number of Tweets", fontsize=15)  
ax.set\_title("Value Count of each Polarity", fontsize=20)  
ax.bar(final\_data['Polarity\_type'].unique(), final\_data['Polarity\_type'].value\_counts())

Out[51]:

<BarContainer object of 2 artists>

# Comparision between Lexicon and Machine learning Prediction[¶](#Xf6d1797c8a0ec79ac481b5dc683d925fda0c0d9)

The prediction from the neural network is actually in between 0 and 1(0 most negative, and 1 most positive).

So, I have mapped the [0, 1] range to [-1, 1] using the below function.

In [52]:

# Required function  
  
def map\_ab\_to\_cd(x, a, b, c, d):  
 shifted = ((d - c) \* ((x - a) / (b - a))) + c  
   
 return shifted

In [53]:

# Map [0, 1] sentiment to [-1, 1]  
  
predicted\_map = map\_ab\_to\_cd(predicted, 0, 1, -1, 1)  
sentiment\_map = np.where(predicted\_map > 0.05, 'Positive', np.where(predicted\_map < -0.05, 'Negative', 'Neutral'))

In [71]:

# Confusion matrix between textblob and neural network model  
  
con\_mat = confusion\_matrix(sentiment\_map, data1['polarity\_type'].values)  
  
\_, ax = plt.subplots()  
sns.heatmap(con\_mat, annot=True, vmin=0, ax=ax)  
ax.set\_title("Confusion Matrix(textblob vs neural network)", fontsize=20)  
ax.set\_xlabel("Neural Network", fontsize=15)  
ax.set\_ylabel("TextBlob", fontsize=15)  
ax.xaxis.set\_ticklabels(['Negative', 'Neutral', 'Positive'])  
ax.yaxis.set\_ticklabels(['Negative', 'Neutral', 'Positive'])  
ax.set\_ylim(sorted(ax.get\_xlim(), reverse=True))

Out[71]:

(3.0, 0.0)

In [72]:

# Confusion matrix between textblob and neural network model  
  
con\_mat = confusion\_matrix(sentiment\_map, data2['polarity\_type'].values)  
  
\_, ax = plt.subplots()  
sns.heatmap(con\_mat, annot=True, vmin=0, ax=ax)  
ax.set\_title("Confusion Matrix(vaderSentiment vs neural network)", fontsize=20)  
ax.set\_xlabel("Neural Network", fontsize=15)  
ax.set\_ylabel("Vader Sentiment", fontsize=15)  
ax.xaxis.set\_ticklabels(['Negative', 'Neutral', 'Positive'])  
ax.yaxis.set\_ticklabels(['Negative', 'Neutral', 'Positive'])  
ax.set\_ylim(sorted(ax.get\_xlim(), reverse=True))

Out[72]:

(3.0, 0.0)