Sentiment Analysis with Covid-19 Tweets

Mert Burak Burabak, Bahçeşehir University, Istanbul

Abstract

In this study, a dataset containing covid-19 themed tweets was used to contribute to sentiment analysis tasks. It is hoped that it will contribute to the real-time pandemic management processes by predicting the sentiment analysis correctly. Three different models were examined in the study. These are: LSTM, GRU, Bidirectional LSTM. As a result of the examinations, Bidirectional LSTM was found to be more successful than other models.

Keywords: Sentiment Analysis, LSTM, GRU, Bidirectional LSTM, Covid-19, Twitter

1. Introduction

Twitter, the world's largest social platform with 300 million monthly users, is increasingly used to obtain real-time health information through public health information and crowdsourcing[1]. Researchers used Twitter to investigate the spread of influenza and real-time outbreaks[2]. Researchers estimated the real-time disease activity of influenza A using keywords in 2009[3]. There are lots of study like above. The educational and informative effect of Twitter has also been noticed by international organizations over time. In 2015, the World Health Organization adopted the use of twitter and other social media. In the first 12 weeks of the Zika outbreak in late 2015, the WHO Twitter account was retweeted over 20,000 times, demonstrating its widespread impact on disseminating health information [4]. Millions of tweets were made about SARS-CoV-2, which started in 2019 and spread all over the world, and people expressed their thoughts. The scope of the study is classified as sequence classification only. In this study, sentiment analysis of coronavirus-themed tweets on Twitter has been made and it is hoped that it will contribute to real-time pandemic management processes.

2. Dataset Description

The dataset used in the study was found on kaggle[5]. The dataset contains encrypted username, screenname, location, tweetat, originaltweet, sentiment columns. Below you can see the definitions for the relevant columns:

- Username: encrypted user name
- Screenname: ecrypted screen name
- Location: geographical information about user
- Tweetat: created date
- Originaltweet: text of the tweet
- Sentiment: sentiment for the tweet

3. Methods

3.1. Data Collection

The dataset created about covid-19 from kaggle was used as the data source[5]. The dataset contains 5 different labels. We have renewed these 5 labels as “positive”, “neutral”, “negative”. Then, these texts were made ready for use by applying the necessary preprocessing.

3.2. Data Preprocessing

The preprocessing operations required to use the data sources in the models are:

- Removing urls
- Removing htmls
- Removing numbers
- Removing punctuation
- Removing stop words
- Removing mentions
- Removing hashtags
- Removing extra spaces

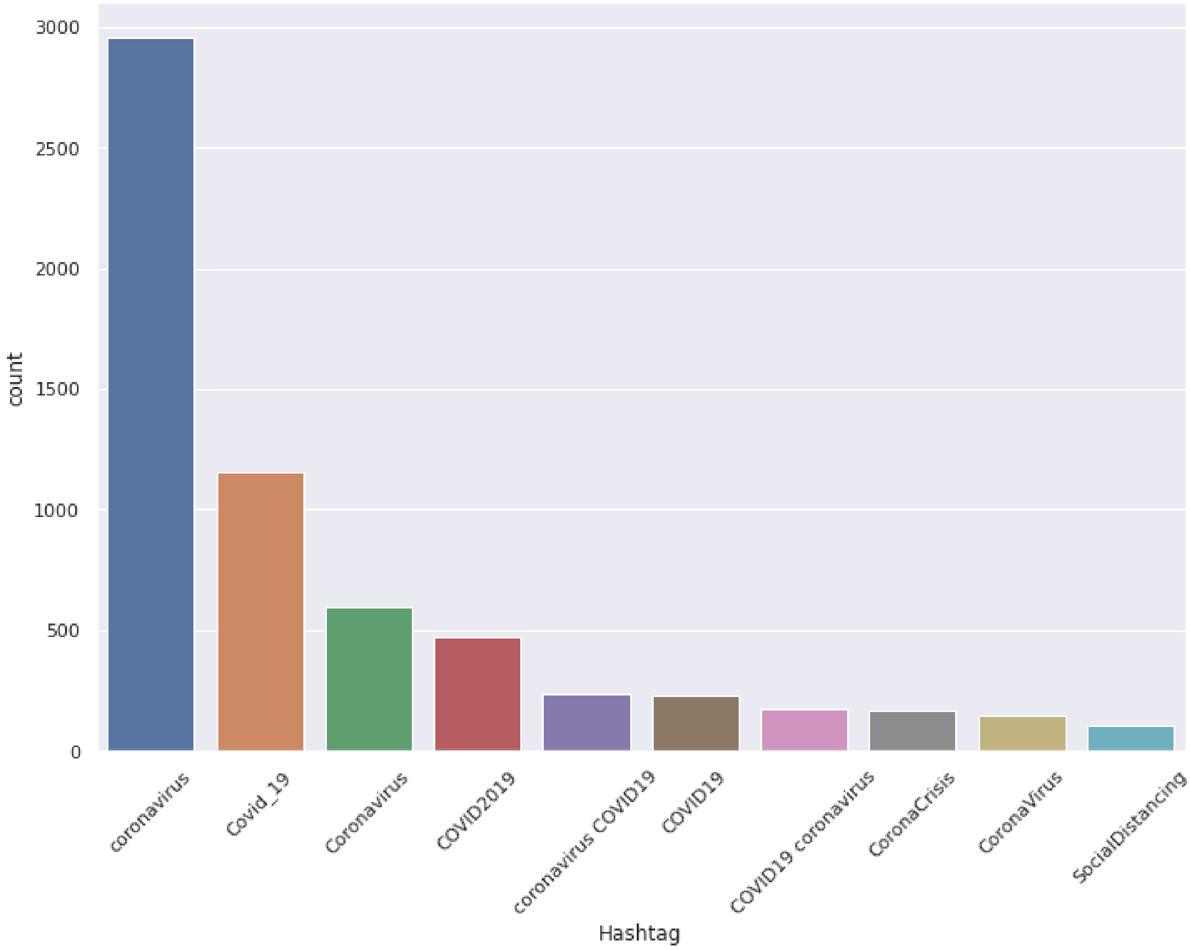
Then all text is tokenized with keras tokenizer function. To be able to use tokenized words in models, it is necessary to sequence these sentences. In the function we use for this, the padding type is post, the truncate type is post, and the maximum sequence length is determined as 60.

The last activity in this section was to determine the train and test data. All data is randomly divided into train and test sets as 0.75-0.25

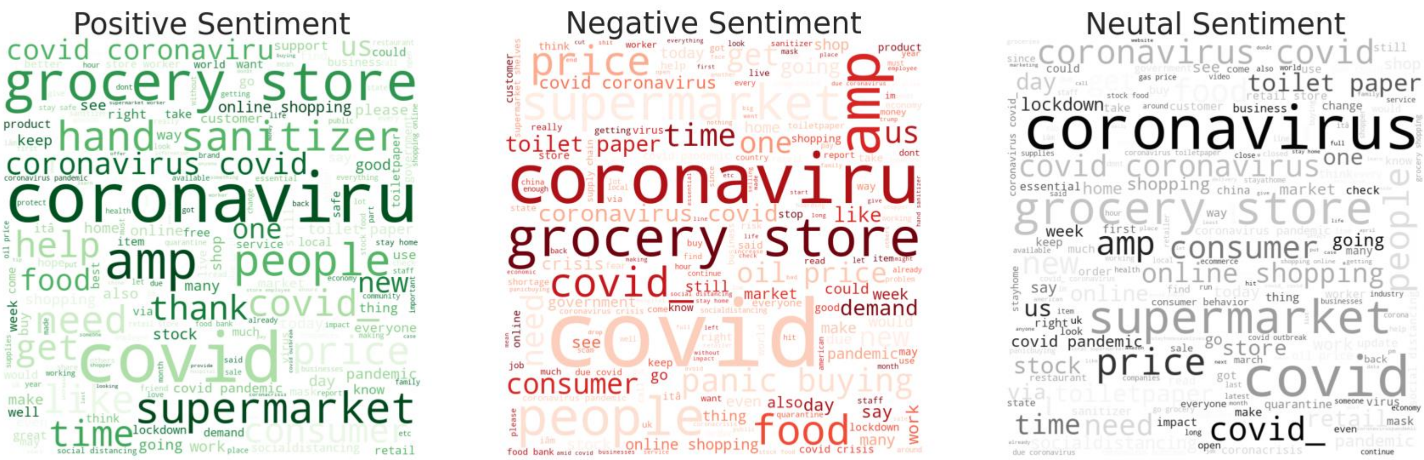
3.3. Exploratory Data Analysis

We have equipped our texts with the necessary tags. Below you can see the distribution of data by tags.

When we look at the hashtags of the tweets in the data source, we can see that they are related to the corona-virus as follows.



The most used words in the texts are as follows:



4. Experimental Results

We worked with 3 different algorithms for sentiment analysis. These are LSTM, GRU, Bidirectional LSTM.

All models were run with 64 batch sizes for 20 epochs. The difference between all models is only the different algorithm in a single layer. We used adam as the optimizer. We used sparse_categorical_crossentropy as the Loss function. We used accuracy as a metric. All models run with 20% validation split, and earlystop and learning rate reduction functions are used in necessary conditions to prevent overfitting.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Adam Optimer Algorithm

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

Cross Entropy Loss:

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

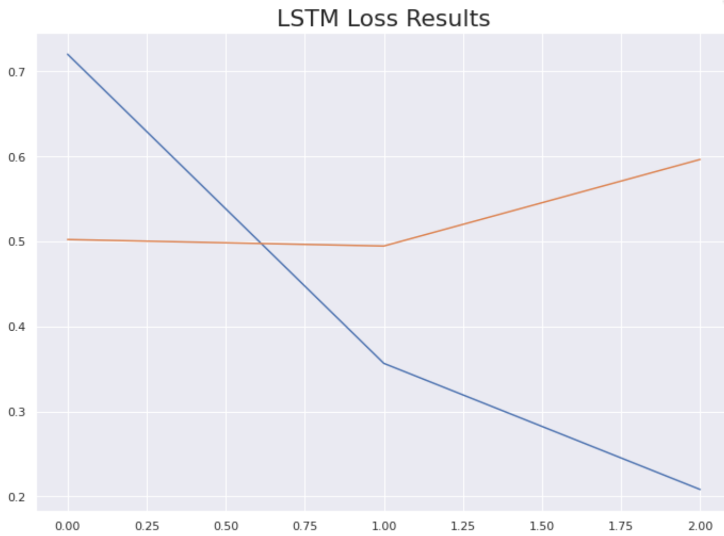
The architectures of the required models are as follows:

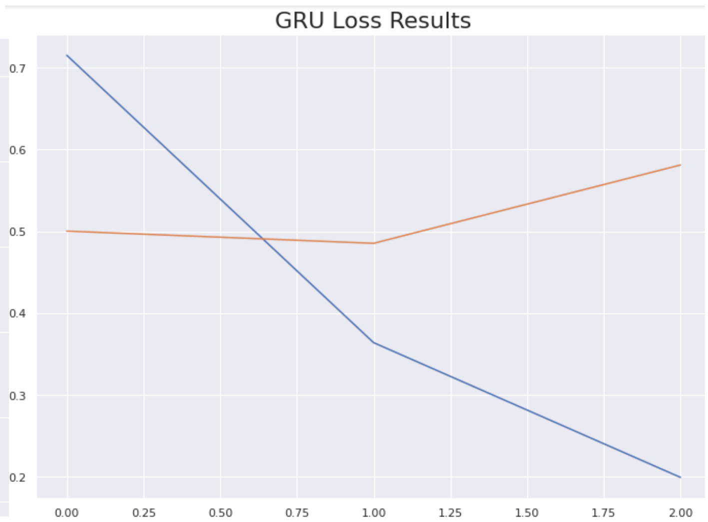
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 60, 32)            1731360
_________________________________________________________________
lstm_4 (LSTM)                (None, 60, 64)            24832
_________________________________________________________________
global_max_pooling1d_5 (Glob (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 195
=================================================================
Total params: 1,760,547
Trainable params: 1,760,547
Non-trainable params: 0
```

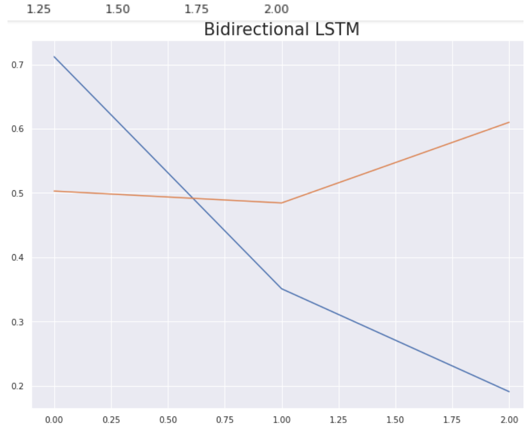
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 60, 32)            1731360
_________________________________________________________________
gru (GRU)                    (None, 60, 64)            18816
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 195
=================================================================
Total params: 1,754,531
Trainable params: 1,754,531
Non-trainable params: 0
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 60, 32)            1731360
_________________________________________________________________
bidirectional (Bidirectional (None, 60, 128)           49664
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 195
=================================================================
Total params: 1,789,475
Trainable params: 1,789,475
Non-trainable params: 0
```

All models were successful in capturing the required patterns, at the same time all 3 models were stopped early due to overfitting. You can see the losses of all models below. Blue colored ones are used for training and red ones are used for validation.







The results of the classification models on the test data are as follows:

	precision	recall	f1-score	support
Negative	0.85	0.70	0.76	3406
Neutral	0.61	0.76	0.68	1667
Positive	0.81	0.85	0.83	3918
accuracy			0.78	8991
macro avg	0.76	0.77	0.76	8991
weighted avg	0.79	0.78	0.78	8991

1-LSTM

	precision	recall	f1-score	support
Negative	0.83	0.75	0.79	3406
Neutral	0.79	0.65	0.71	1667
Positive	0.78	0.90	0.83	3918
accuracy			0.80	8991
macro avg	0.80	0.77	0.78	8991
weighted avg	0.80	0.80	0.79	8991

2-GRU

	precision	recall	f1-score	support
Negative	0.80	0.83	0.81	3406
Neutral	0.81	0.72	0.76	1667
Positive	0.84	0.86	0.85	3918
accuracy			0.82	8991
macro avg	0.82	0.80	0.81	8991
weighted avg	0.82	0.82	0.82	8991

3-Bidirectional LSTM

As can be seen from the images above, all models yielded close results, but when examined in detail, the Bi-directional LSTM is slightly ahead in estimating the emotions correctly.

5. Conclusions

In this study, it is hoped that it will contribute to real-time pandemic management by examining covid-19 themed tweets. Text data has been cleaned by making necessary preprocessing. Then, three different models were examined. These worked with LSTM, GRU, Bidirectional LSTM algorithms. The only difference in all architectures is the algorithm used in the middleware. As a result of the examinations, it was understood that Bidirectional LSTM was the best working model with 82% accuracy. To develop this project, much larger and pre-trained models such as BERT can be examined and more successful models can be established.

6. References

[1] – Scanfeld D, Scanfeld V, Larson EL. Dissemination of health information through social networks: Twitter and antibiotics. American Journal of Infection Control 2010; 38:182–188

[2] – Chorianopoulos K, Talvis K. Flutrack.org: Open-source and linked data for epidemiology. Health Informatics J 2016; 22:962–974

[3] – Signorini A, Segre AM, Polgreen PM. The Use of Twitter to Track Levels of Disease Activity and Public Concern in the U.S. during the Influenza A H1N1 Pandemic. PLoS ONE 2011; 6:e19467.

[4] – Stefanidis A, Vraga E, Lamprianidis G, et al. Zika in Twitter: Temporal Variations of Locations, Actors, and Concepts. JMIR Public Health Surveill 2017; 3:e22.

[5] - https://www.kaggle.com/datatattle/covid-19-nlp-text-classification

7. Appendix

https://colab.research.google.com/drive/1oUN35NZ-SaEN32AvtSlb2_IG_jSZsAFt?usp=sharing https://www.kaggle.com/datatattle/covid-19-nlp-text-classification

Sentiment Analysis with Covid-19 Tweets

Mert Burak Burabak, Bahçeşehir University, Istanbul

In [ ]:

```
##########Load the kaggle api token###########
!pip install --upgrade --force-reinstall --no-deps kaggle
!pip install transformers
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json
!kaggle datasets download -d datatattle/covid-19-nlp-text-classification
!unzip covid-19-nlp-text-classification
```

In [2]:

```
### Libraries ###
import re
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense,LSTM,Embedding,Input,GlobalMaxPool1D,Bidirectional,GRU
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import EarlyStopping,ReduceLROnPlateau
from transformers import TFBertModel
from transformers import AdamW,get_linear_schedule_with_warmup
from transformers import BertTokenizer
from torch.utils.data import TensorDataset
import tensorflow as tf
from torch.utils.data import RandomSampler,DataLoader
from tqdm.notebook import tqdm



import seaborn as sns
import matplotlib.pylab as plt
from plotly.subplots import make_subplots
import plotly.graph_objects as go

nltk.download('stopwords')
```

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

Out[2]:

```
True
```

0.Loading Data

In [3]:

```
train=pd.read_csv("Corona_NLP_train.csv",encoding='latin1')
```

```
test=pd.read_csv("Corona_NLP_test.csv",encoding='latin1')


train['OriginalTweet']=train['OriginalTweet'].astype(str)
train['Sentiment']=train['Sentiment'].astype(str)
test['OriginalTweet']=test['OriginalTweet'].astype(str)
test['Sentiment']=test['Sentiment'].astype(str)

df=pd.concat([train,test])
df['OriginalTweet']=df['OriginalTweet'].astype(str)
df['Sentiment']=df['Sentiment'].astype(str)
df.head()
```

Out[3]:

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
3	3802	48754	NaN	16-03-2020	My food stock is not the only one which is emp...	Positive
4	3803	48755	NaN	16-03-2020	Me, ready to go at supermarket during the #COV...	Extremely Negative

In [4]:

```
train.shape, test.shape, df.shape
```

Out[4]:

```
((41157, 6), (3798, 6), (44955, 6))
```

In [4]:

```
#Missing Values
df.isnull().sum()
```

Out[4]:

```
UserName            0
ScreenName          0
Location         9424
TweetAt             0
OriginalTweet       0
Sentiment           0
dtype: int64
```

In [5]:

```
#Sentiment Distribution
print(df.Sentiment.unique())
print(df.Sentiment.value_counts())
```

```
['Neutral' 'Positive' 'Extremely Negative' 'Negative' 'Extremely Positive']
Positive              12369
Negative              10958
Neutral                8332
Extremely Positive     7223
Extremely Negative     6073
Name: Sentiment, dtype: int64
```

In [6]:

```
# We will copy the text in another column so that the original text is also there for com
parison
df['text'] = df.OriginalTweet
df["text"] = df["text"].astype(str)

train['text'] = train.OriginalTweet
train["text"] = train["text"].astype(str)
```

```
test['text'] = test.OriginalTweet
test["text"] = test["text"].astype(str)

# Data has 5 classes, let's convert them to 3
def classes_def(x):
    if x ==  "Extremely Positive":
        return "positive"
    elif x == "Extremely Negative":
        return "negative"
    elif x == "Negative":
        return "negative"
    elif x ==  "Positive":
        return "positive"
    else:
        return "neutral"

df['sentiment']=df['Sentiment'].apply(lambda x:classes_def(x))
train['sentiment']=train['Sentiment'].apply(lambda x:classes_def(x))
test['sentiment']=test['Sentiment'].apply(lambda x:classes_def(x))

round(df.sentiment.value_counts(normalize= True),2)
```

Out[6]:

```
positive    0.44
negative    0.38
neutral     0.19
Name: sentiment, dtype: float64
```

1. EDA Analysis

In [7]:

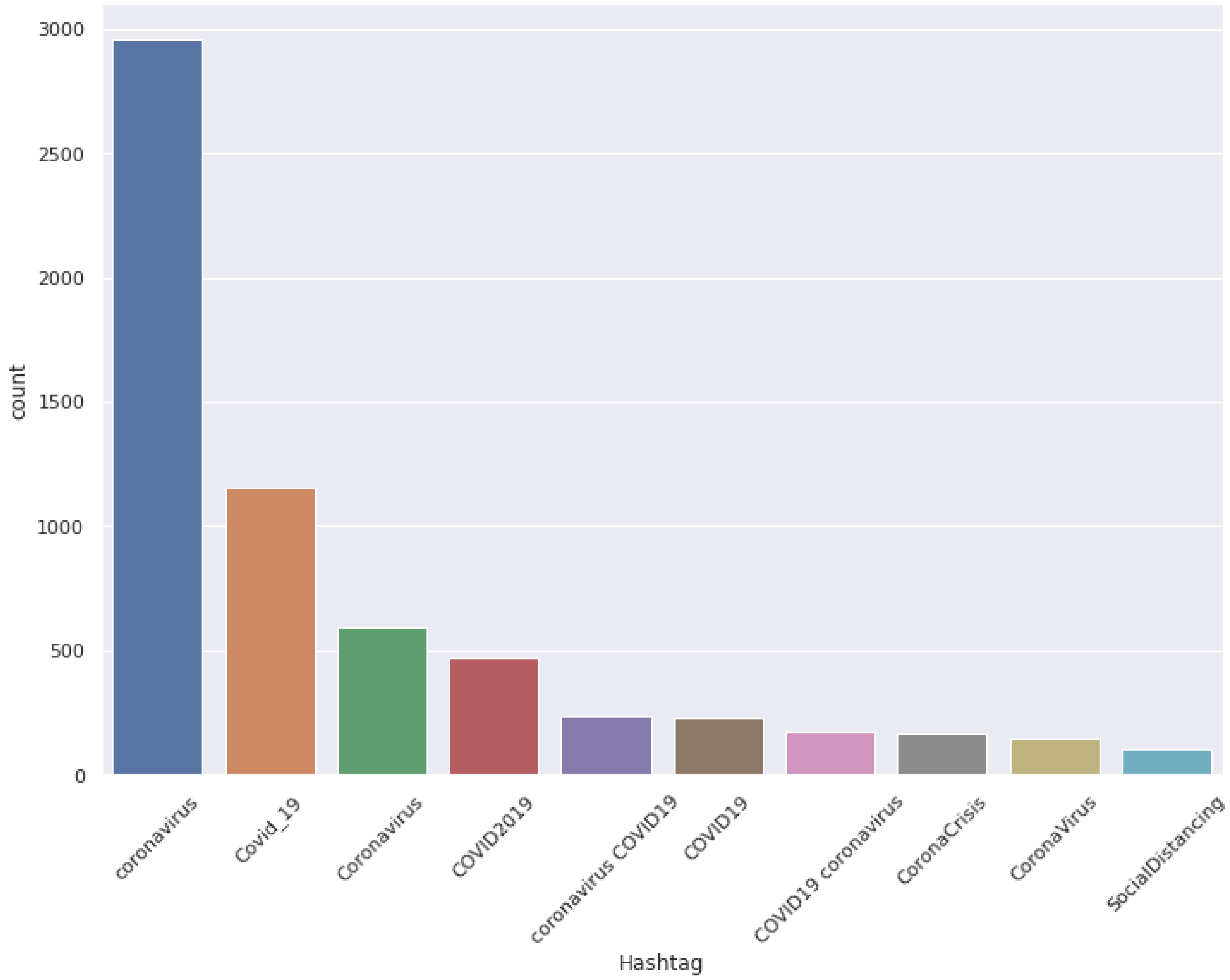
```
fig=make_subplots(1,2,subplot_titles=('Train set','Test set'))
x=train.sentiment.value_counts()
fig.add_trace(go.Bar(x=x.index,y=x.values,
                     marker_color=['#17C37B','#F92969','#FACA0C'],name='train'),row=1,co
l=1)
x=test.sentiment.value_counts()
fig.add_trace(go.Bar(x=x.index,y=x.values,
                     marker_color=['#17C37B','#F92969','#FACA0C'],name='test'),row=1,col
=2)
```

In [8]:

```
def find_hash(text):
    line=re.findall(r'(?<=#)\w+',text)
    return " ".join(line)

sns.set(rc={'figure.figsize':(11.7,8.27)})
df['hash']=df['text'].apply(lambda x:find_hash(x))
temp=df['hash'].value_counts()[:][1:11]
temp= temp.to_frame().reset_index().rename(columns={'index':'Hashtag','hash':'count'})
sns.barplot(x="Hashtag",y="count", data = temp)
plt.xticks(rotation=45)
```

Out[8]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```

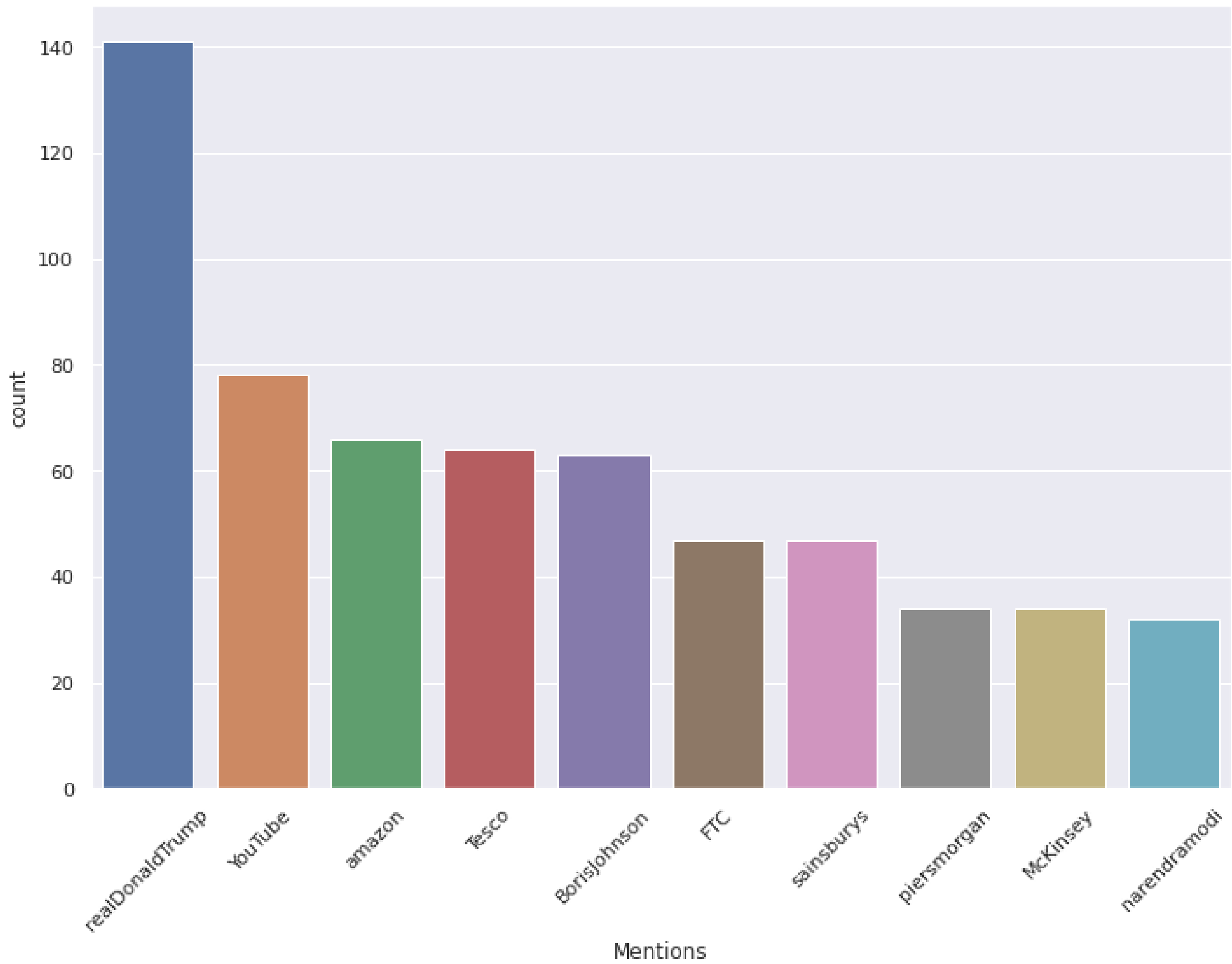


In [9]:

```
def mentions(text):
    line=re.findall(r'(?<=@)\w+',text)
    return " ".join(line)
df['mentions']=df['text'].apply(lambda x:mentions(x))

temp=df['mentions'].value_counts()[:][1:11]
temp =temp.to_frame().reset_index().rename(columns={'index':'Mentions','mentions':'count
'})

sns.barplot(x="Mentions",y="count", data = temp)
plt.xticks(rotation=45)
```

Out[9]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```



2.Preprocessing

In [10]:

```
STOPWORDS = set(stopwords.words('english'))

#Remove Urls and HTML links
def remove_urls(text):
    url_remove = re.compile(r'https?://\S+|www\.\S+')
    return url_remove.sub(r'', text)

def remove_html(text):
    html=re.compile(r'<.*?>')
    return html.sub(r'',text)

# Lower casing
def lower(text):
    low_text= text.lower()
    return low_text

# Number removal
def remove_num(text):
    remove= re.sub(r'\d+', '', text)
    return remove

#Remove stopwords & Punctuations
def punct_remove(text):
    punct = re.sub(r"[^\w\s\d]","", text)
    return punct

def remove_stopwords(text):
```

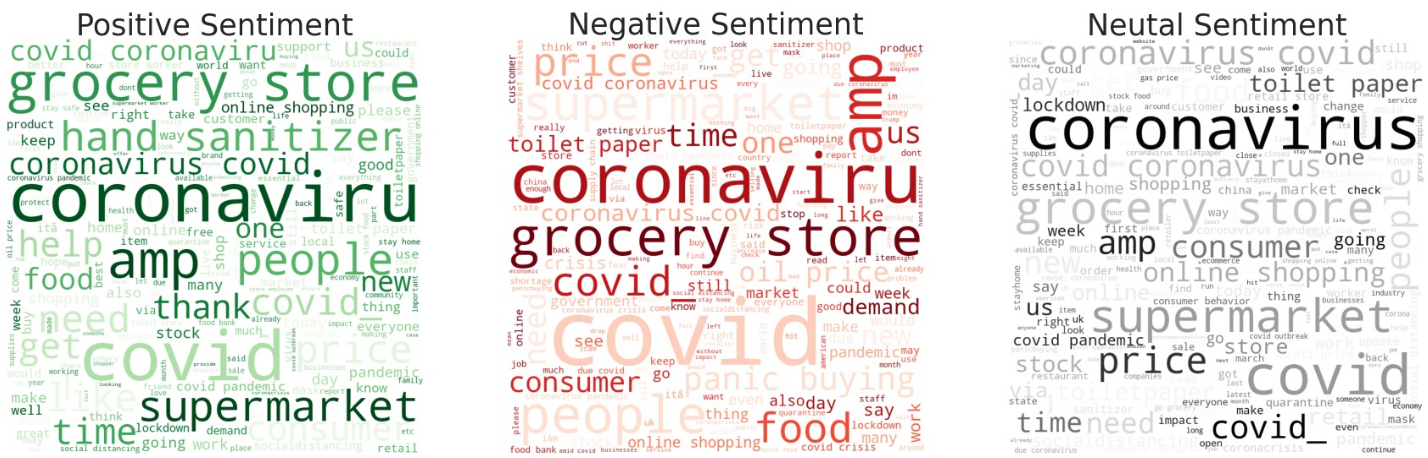
```
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

#Remove mentions and hashtags
def remove_mention(x):
    text=re.sub(r'@\w+','',x)
    return text

def remove_hash(x):
    text=re.sub(r'#\w+','',x)
    return text

#Remove extra white space left while removing stuff
def remove_space(text):
    space_remove = re.sub(r"\s+"," ",text).strip()
    return space_remove

df['text_new']=df['text'].apply(lambda x:remove_urls(x))
df['text']=df['text_new'].apply(lambda x:remove_html(x))
df['text_new']=df['text'].apply(lambda x:lower(x))
df['text']=df['text_new'].apply(lambda x:remove_num(x))
df['text_new']=df['text'].apply(lambda x:punct_remove(x))
df['text']=df['text_new'].apply(lambda x:remove_stopwords(x))
df['text_new']=df['text'].apply(lambda x:remove_mention(x))
df['text']=df['text_new'].apply(lambda x:remove_hash(x))
df['text_new']=df['text'].apply(lambda x:remove_space(x))
df = df.drop(columns=['text_new'])
```

3.Wordcloud

In [11]:

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=[30, 15])

df_pos = df[df["sentiment"]=="positive"]
df_neg = df[df["sentiment"]=="negative"]
df_neu = df[df["sentiment"]=="neutral"]

comment_words = ''
stopwords = set(STOPWORDS)

for val in df_pos.text:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "


wordcloud1 = WordCloud(width = 800, height = 800,
                background_color ='white',
                colormap="Greens",
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

ax1.imshow(wordcloud1)
ax1.axis('off')
ax1.set_title('Positive Sentiment',fontsize=35);

comment_words = ''

for val in df_neg.text:

    # typecaste each val to string
```

```
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "



wordcloud2 = WordCloud(width = 800, height = 800,
                background_color ='white',
                colormap="Reds",
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
ax2.imshow(wordcloud2)
ax2.axis('off')
ax2.set_title('Negative Sentiment',fontsize=35);



comment_words = ''
for val in df_neu.text:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud3 = WordCloud(width = 800, height = 800,
                background_color ='white',
                colormap="Greys",
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
ax3.imshow(wordcloud3)
ax3.axis('off')
ax3.set_title('Neutal Sentiment',fontsize=35);
```



4.Models

In [12]:

```
df['labelled_sentiment'] = LabelEncoder().fit_transform(df.sentiment)
```

```
xtrain, xvalid, ytrain, yvalid = train_test_split(df.text, df.labelled_sentiment,
                                                  stratify=df.labelled_sentiment,
                                                  random_state=42,
                                                  test_size=0.2, shuffle=True)
```

In [13]:

```
class Lemmatizer(object):
    def __init__(self):
        self.lemmatizer = WordNetLemmatizer()
    def __call__(self, sentence):
        sentence=re.sub('(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)',' ',s
entence)
        sentence=re.sub('[^0-9a-z]',' ',sentence)
        return [self.lemmatizer.lemmatize(word) for word in sentence.split() if len(word
)>1]

# using keras tokenizer here
token = Tokenizer(num_words=None)
max_len = 60

token=Tokenizer(num_words=None,oov_token=Lemmatizer())
token.fit_on_texts(xtrain)
train_x = token.texts_to_sequences(xtrain)
train_x_padded = pad_sequences(train_x ,maxlen=max_len, padding='post', truncating='post
')
test_x = token.texts_to_sequences(xvalid)
test_x_padded = pad_sequences(test_x, maxlen=max_len, padding='post', truncating='post')
```

1.LSTM

In [34]:

```
embedding_dimension=32
v=len(token.word_index)
num_epoch=20
size_batch=64

early_stop=EarlyStopping(monitor='val_accuracy',patience=1)
reduceLR=ReduceLROnPlateau(monitor='val_accuracy',patience=1)

model=Sequential()
model.add(Input(shape=(60,)))
model.add(Embedding(v+1,embedding_dimension))
model.add(LSTM(64,return_sequences=True))
model.add(GlobalMaxPool1D())
model.add(Dense(64))
model.add(Dense(3,activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accurac
y'])
r=model.fit(train_x_padded, ytrain, validation_split=0.2,
            epochs=num_epoch, batch_size=size_batch,
            callbacks=[reduceLR,early_stop])
```

```
WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `keras.Input` to Seque
ntial model. `keras.Input` is intended to be used by Functional model.
Epoch 1/20
450/450 [==============================] - 11s 21ms/step - loss: 0.8830 - accuracy: 0.564
2 - val_loss: 0.5023 - val_accuracy: 0.8216
Epoch 2/20
450/450 [==============================] - 9s 20ms/step - loss: 0.3485 - accuracy: 0.8870
- val_loss: 0.4946 - val_accuracy: 0.8220
Epoch 3/20
450/450 [==============================] - 9s 20ms/step - loss: 0.1934 - accuracy: 0.9424
- val_loss: 0.5965 - val_accuracy: 0.7860
```

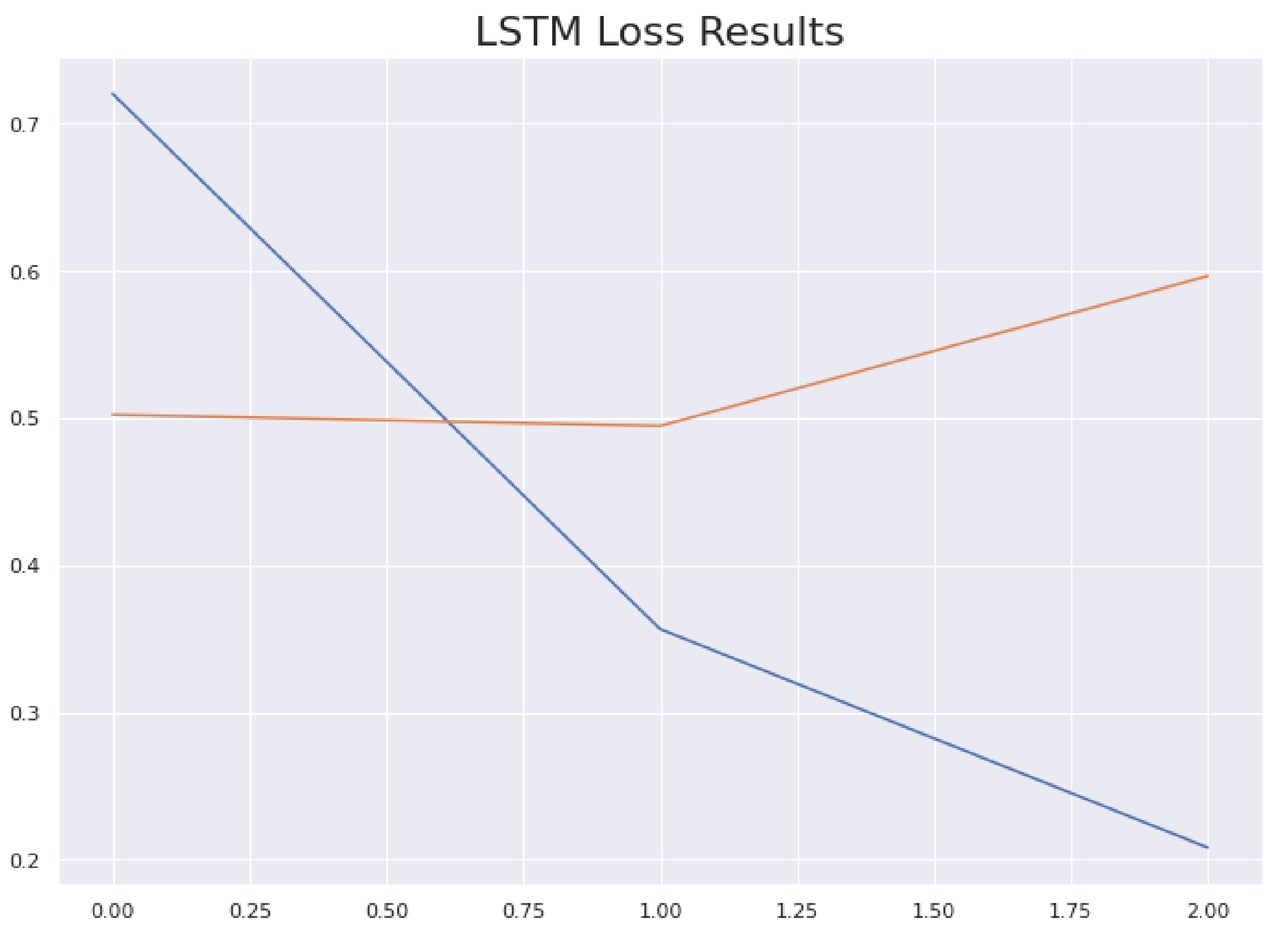
In [36]:

```
plt.plot(r.history['loss'])
```

```
plt.plot(r.history['val_loss'])
plt.title('LSTM Loss Results',fontdict={'size':'22'})
plt.plot()
```

Out[36]:

```
[]
```



In [37]:

```
ypred = model.predict(test_x_padded)
ypred = [np.argmax(i) for i in ypred]
```

In [38]:

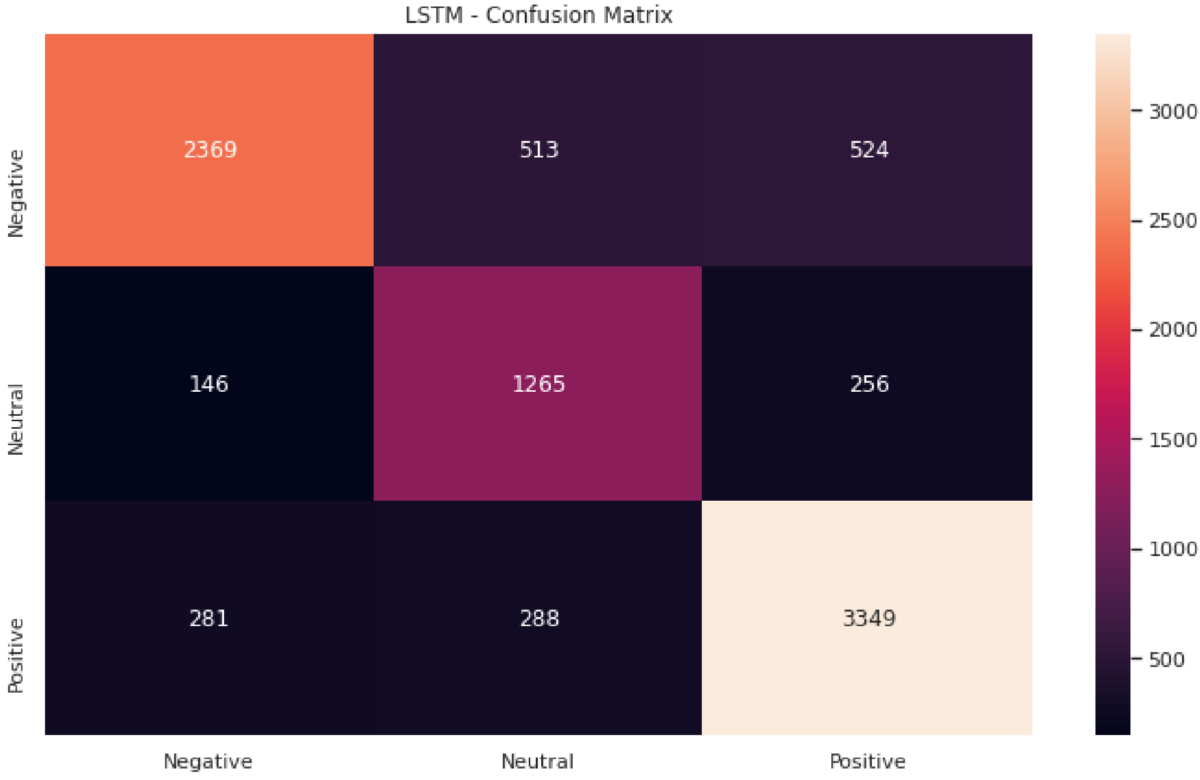
```
print(classification_report(yvalid,ypred,target_names=['Negative', 'Neutral', 'Positive'
]))
```

```
              precision    recall  f1-score   support

    Negative       0.85      0.70      0.76      3406
     Neutral       0.61      0.76      0.68      1667
    Positive       0.81      0.85      0.83      3918

    accuracy                           0.78      8991
   macro avg       0.76      0.77      0.76      8991
weighted avg       0.79      0.78      0.78      8991
```

In [39]:

```
labels = ['Negative', 'Neutral', 'Positive']
conf = confusion_matrix(yvalid, ypred)
cm = pd.DataFrame(
    conf, index = [i for i in labels],
    columns = [i for i in labels]
)

plt.figure(figsize = (12,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.title("LSTM - Confusion Matrix")
```

```
plt.show()
```



2. GRU

In [40]:

```
num_epoch=20
size_batch=64

model = Sequential()
model.add(Input(shape=(60,)))
model.add(Embedding(v+1,embedding_dimension))
model.add(GRU(64,return_sequences=True))
model.add(GlobalMaxPool1D())
model.add(Dense(64))
model.add(Dense(3,activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accurac
y'])
model.summary()
r=model.fit(train_x_padded, ytrain, validation_split=0.2,
            epochs=num_epoch, batch_size=size_batch,
            callbacks=[reduceLR,early_stop])
```

```
WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `keras.Input` to Seque
ntial model. `keras.Input` is intended to be used by Functional model.
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 60, 32)            1731360
_________________________________________________________________
gru_1 (GRU)                  (None, 60, 64)            18816
_________________________________________________________________
global_max_pooling1d_7 (Glob (None, 64)                0
_________________________________________________________________
dense_14 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_15 (Dense)             (None, 3)                 195
=================================================================
Total params: 1,754,531
Trainable params: 1,754,531
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
```

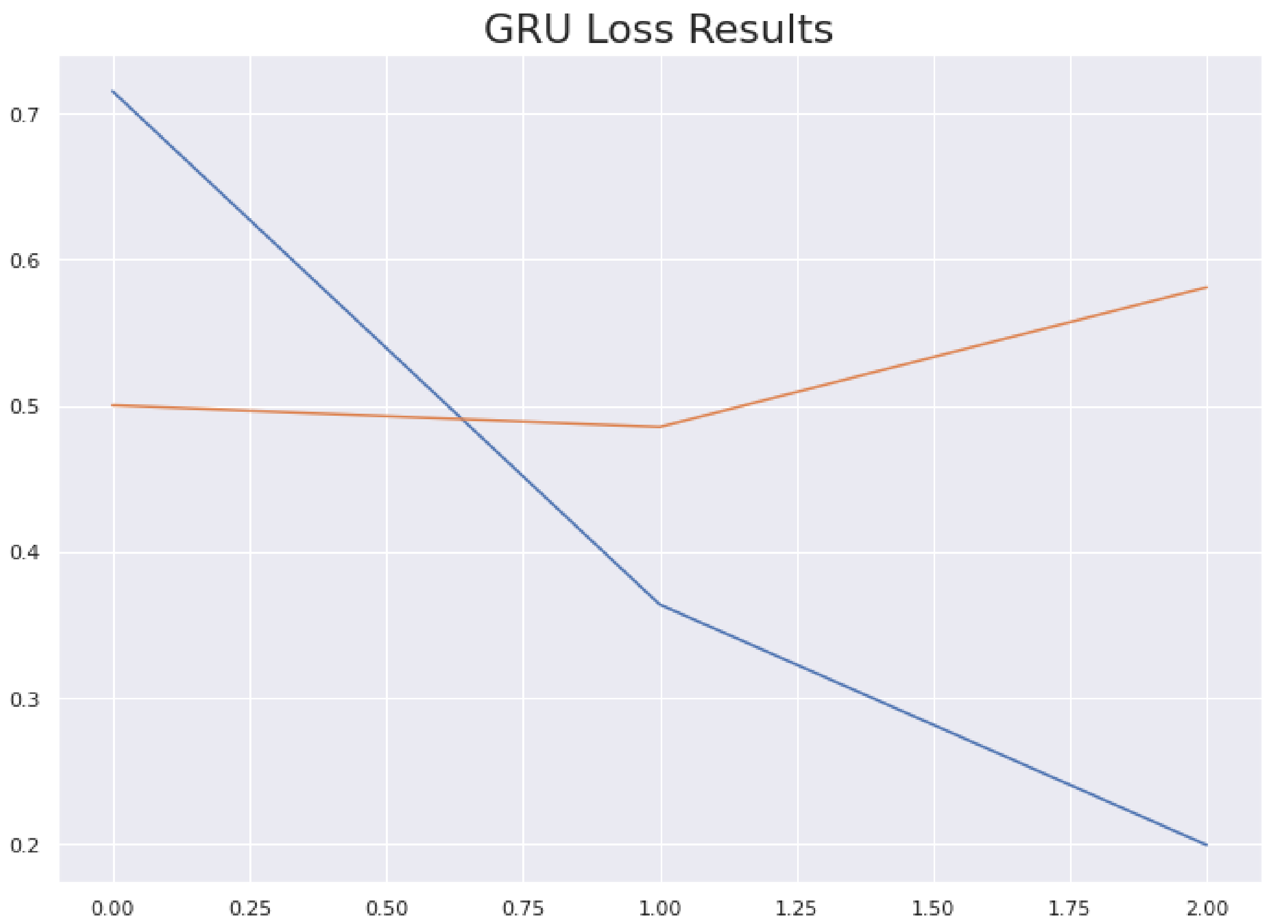
```
450/450 [==============================] - 11s 22ms/step - loss: 0.8792 - accuracy: 0.565
3 - val_loss: 0.5004 - val_accuracy: 0.8184
Epoch 2/20
450/450 [==============================] - 9s 20ms/step - loss: 0.3623 - accuracy: 0.8825
- val_loss: 0.4855 - val_accuracy: 0.8257
Epoch 3/20
450/450 [==============================] - 9s 21ms/step - loss: 0.1859 - accuracy: 0.9415
- val_loss: 0.5812 - val_accuracy: 0.8077
```

In [42]:

```
plt.plot(r.history['loss'])
plt.plot(r.history['val_loss'])
plt.title('GRU Loss Results',fontdict={'size':'22'})
plt.plot()
```

Out[42]:

```
[]
```

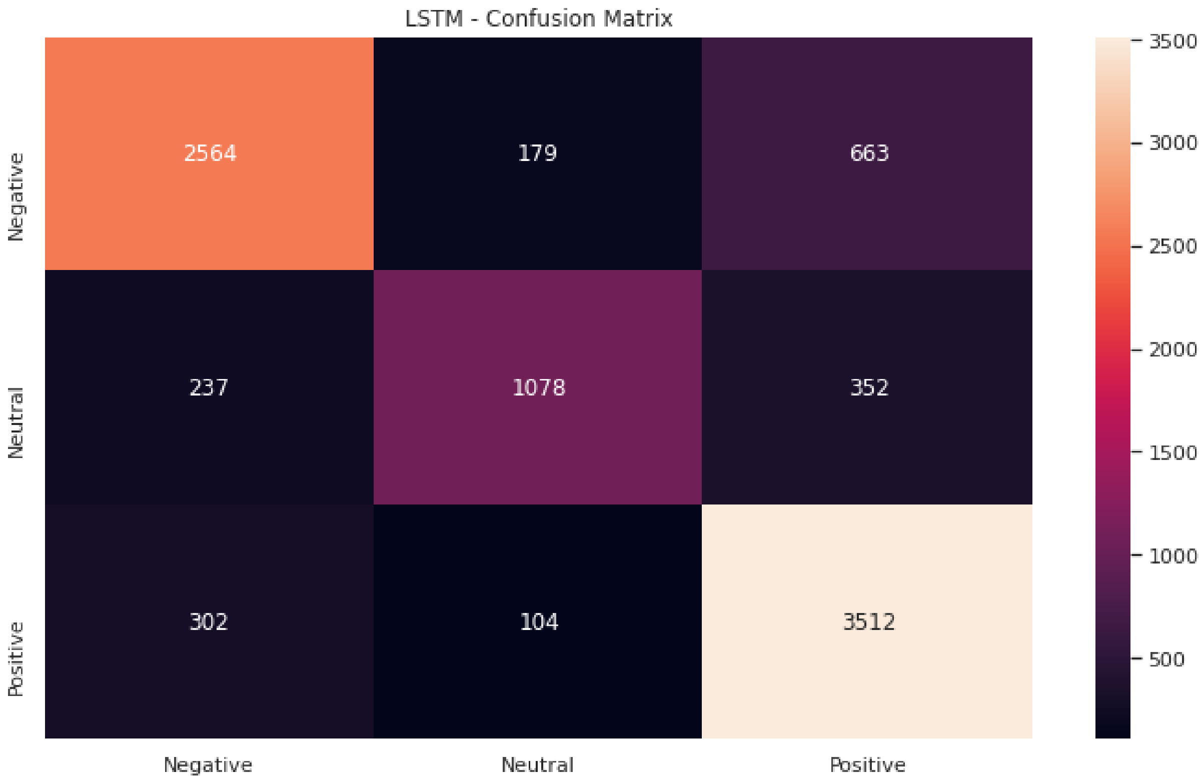


In [43]:

```
ypred = model.predict(test_x_padded)
ypred = [np.argmax(i) for i in ypred]
print(classification_report(yvalid,ypred,target_names=['Negative', 'Neutral', 'Positive'
]))
labels = ['Negative', 'Neutral', 'Positive']
conf = confusion_matrix(yvalid, ypred)
cm = pd.DataFrame(
    conf, index = [i for i in labels],
    columns = [i for i in labels]
)

plt.figure(figsize = (12,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.title("LSTM - Confusion Matrix")
plt.show()
```

```
              precision    recall  f1-score   support

    Negative       0.83      0.75      0.79      3406
```

```
     Neutral       0.79      0.65      0.71      1667
    Positive       0.78      0.90      0.83      3918

    accuracy                           0.80      8991
   macro avg       0.80      0.77      0.78      8991
weighted avg       0.80      0.80      0.79      8991
```



3.Bi-Directional LSTM

In [44]:

```
num_epoch=20
size_batch=64

model = Sequential()
model.add(Input(shape=(60,)))
model.add(Embedding(v+1,embedding_dimension))
model.add(Bidirectional(LSTM(64,return_sequences=True)))
model.add(GlobalMaxPool1D())
model.add(Dense(64))
model.add(Dense(3,activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accurac
y'])
model.summary()
r=model.fit(train_x_padded, ytrain, validation_split=0.2,
            epochs=num_epoch, batch_size=size_batch,
            callbacks=[reduceLR,early_stop])
```

```
WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `keras.Input` to Seque
ntial model. `keras.Input` is intended to be used by Functional model.
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 60, 32)            1731360
_________________________________________________________________
bidirectional_1 (Bidirection (None, 60, 128)           49664
_________________________________________________________________
global_max_pooling1d_8 (Glob (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_17 (Dense)             (None, 3)                 195
```

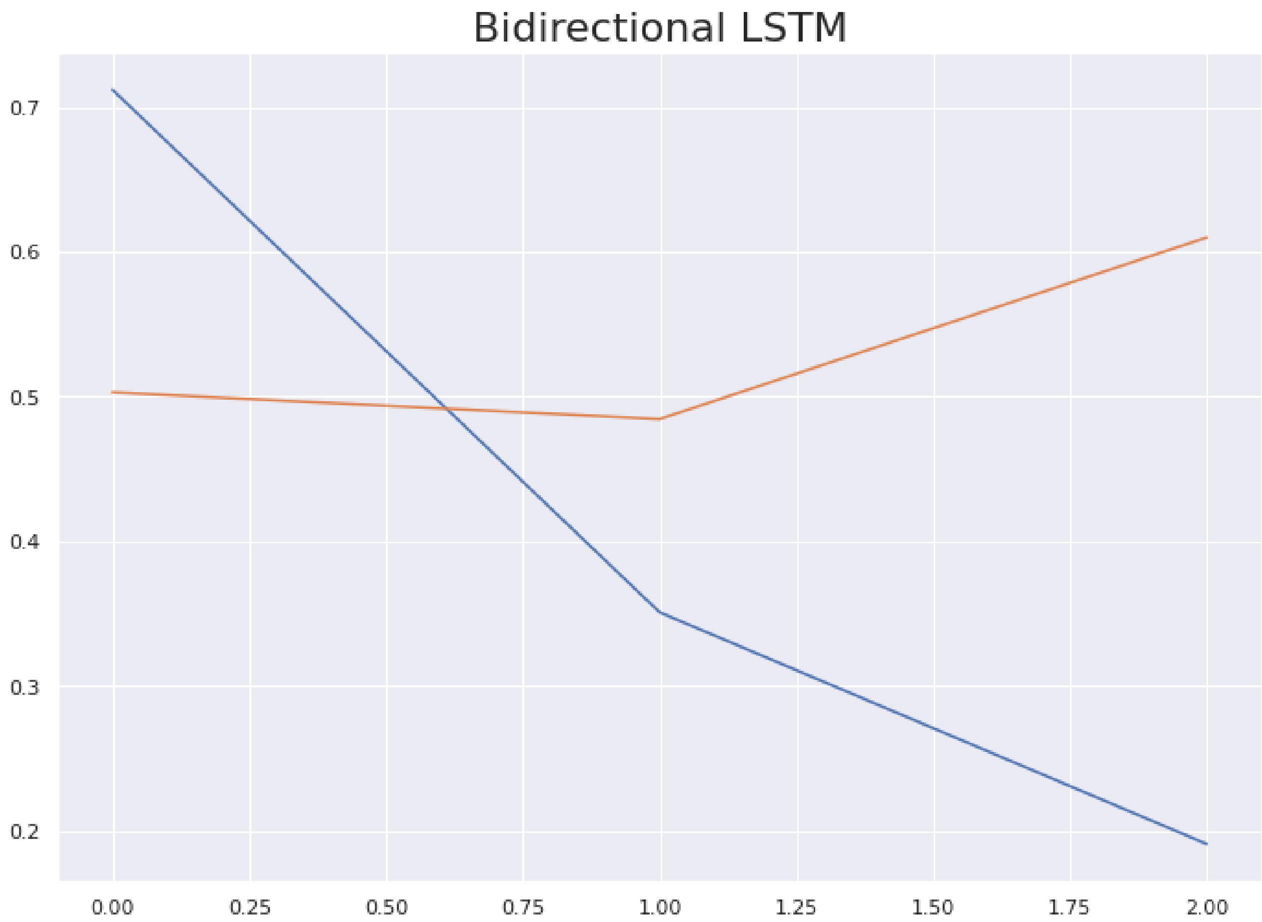
```
=================================================================
Total params: 1,789,475
Trainable params: 1,789,475
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
450/450 [==============================] - 16s 27ms/step - loss: 0.8836 - accuracy: 0.556
6 - val_loss: 0.5030 - val_accuracy: 0.8205
Epoch 2/20
450/450 [==============================] - 11s 24ms/step - loss: 0.3402 - accuracy: 0.890
5 - val_loss: 0.4845 - val_accuracy: 0.8289
Epoch 3/20
450/450 [==============================] - 11s 25ms/step - loss: 0.1760 - accuracy: 0.946
0 - val_loss: 0.6099 - val_accuracy: 0.7970
```

In [46]:

```
plt.plot(r.history['loss'])
plt.plot(r.history['val_loss'])
plt.title('Bidirectional LSTM',fontdict={'size':'22'})
plt.plot()
```

Out[46]:

```
[]
```



In [24]:

```
ypred = model.predict(test_x_padded)
ypred = [np.argmax(i) for i in ypred]
print(classification_report(yvalid,ypred,target_names=['Negative', 'Neutral', 'Positive'
]))
labels = ['Negative', 'Neutral', 'Positive']
conf = confusion_matrix(yvalid, ypred)
cm = pd.DataFrame(
    conf, index = [i for i in labels],
    columns = [i for i in labels]
)

plt.figure(figsize = (12,7))
sns.heatmap(cm, annot=True, fmt="d")
```

```
plt.title("LSTM - Confusion Matrix")
plt.show()
```

```
              precision    recall  f1-score   support

    Negative       0.80      0.83      0.81      3406
     Neutral       0.81      0.72      0.76      1667
    Positive       0.84      0.86      0.85      3918

    accuracy                           0.82      8991
   macro avg       0.82      0.80      0.81      8991
weighted avg       0.82      0.82      0.82      8991
```