PERFORMANCE ASSESSMENT_ D213

NLM2 TASK 2: SENTIMENT ANALYSIS USING NEURAL NETWORKS

May 10th,2022

Table of Contents

Performance Evaluation of NLM2: SENTMENTAL ANALYSIS USING NEURAL NETWORKS, D213

Name: Rekha Alle
Student ID: 000778673
Course: Masters Data Analytics
Date: 05/10/2022
Program Mentor: Festus Elleh
Contact: 385-428-3501
Email: Festus.elleh@wgu.edu

PART I: Introduction to Scenario

Machine learning and AI have provided businesses with a priceless advantage: Amazon review analysis. Amazon and online shopping have become synonymous, owing to the platform's ability to allow small businesses to expand faster than they could through traditional retail outlets. A person can self-publish a copy of the book without having to worry about convincing a publisher. A small business in a faraway land can open an Amazon account and sell to people all over the world.

Because of its prominence and popularity, Amazon is the only site where people genuinely spend time and submit lengthy evaluations, as opposed to other platforms where customers must be prodded. As the case we'll look at later shows, Amazon review research process can reveal a lot about a product, including aspects that the company may not have considered.

A1. Analytical Question:

Analyzing Amazon review data can provide valuable customer response that can be used to improve products? Discover product features from Amazon reviews while also allowing brands to explore beyond star ratings. The sentimental analysis method will be used to solve this analysis.

A2. Goals and Objectives:

Understanding consumer behavior and demands in relation to a company and its products is critical. Sentiment analysis is one of the main aspects in which NLP has been widely applied. In general, client feedback on a product can be classified as Positive, Negative, or Neutral. Companies can determine how satisfied their clients are with their products/services by interpreting client feedback through product reviews.

A3. Sentimental Analysis:

Many of us, without even realizing it, engage in sentiment analysis on a regular basis. It's become a fundamental part of our culture; from the reviews we post on sites like Yelp and Amazon to the comments we provide on a purchase in an online store. Companies had a huge difficulty early on with how to respond to often contradicting customer feedback. Companies were able to deal with this issue thanks to the emergence of sentiment analysis, which allowed them to focus on the overall tone of the remarks rather than the specifics and modify their responses accordingly.

Identifying the emotions underlying each phrase - whether anger, contempt, fear, joy, sadness, or surprise - is a typical method. NLP, which evaluates the meaning of each word, is used to do this. Following that, machine learning and natural language comprehension are used to figure out what the text is about, such as what generated the feelings or how they are related to one another.

Sentimental Analysis classified into three categories:

- Predictive sentiment analysis
- Diagnostic sentiment analysis
- Sentiment classification

In this analysis, we used Recurrent Neural Network (RNN), A recurrent neural network is a type of neural network that is designed to handle a sequence of data x(t)= x(1), x(2),..., x() with a time step index t ranging from 1 to t. RNNs are often preferable for jobs that need sequential inputs, such as voice and language. RNNs are referred to as recurrent because they do the same task for each element of a sequence, with the outcome relying on past calculations. If you wish to anticipate the next word in a sentence in an NLP problem, you must first understand the words that precede it. RNNs also have a "memory" that stores information about previous calculations.

In RNN, sequence we're interested in is a three-word sentence, the network will be unrolled into a three-layer neural network, with one layer for each word.

- **Input:** At time step t, x(t) is used as the network's input. For instance, x1 could be a one-hot vector matching to a sentence's first word.
- **Hidden state:** h(t) represents a hidden state at time t and serves as the network's "memory." h(t) is calculated using the current input and the hidden state from the previous time step: h(t) = f(U x(t) + W h(t1)). A non-linear transformation, such as tanh or ReLU, is assumed for the function f.
- **Weights:** The RNN has input to hidden connections that are measured by a weight matrix U, hidden-to-hidden recurrent connections that are measured by a weight matrix W, and hidden-to-output connections that are parameterized by a weight matrix V, with all of these weights (U,V,W) being shared across time.

- **Output:** The output of the network is represented by o(t). In the diagram, I simply added an arrow after o(t), which is likewise prone to non-linearity, particularly when the network has further layers downstream.

  We use RNN with LSTM models for this analysis.

PART II: Data Objectives

Do the following to demonstrate the Data cleaning process:

B1. EDA is explained as follows:

Exploratory data analysis explores a series of stages and processes that involves the data preparation and cleaning to obtain clean data set. Lemmatizing, tokenization, and the removal of unclean data (tags, punctuation, stop words, and so on) were conducted in addition to the fundamental preprocessing and data cleaning had done to the text data. This alone resulted in an increase of over 18 points in accuracy. and visually plot the data. For the most part, EDA hasn't changed.

Changes to the hyperparameters of all evaluated models to enhance accuracy even slightly, inclusion of cross validation testing for all models, and an increase in dataset size are the primary differences of work.

We must assume that all customers who purchase a product leave a rating/review to ascertain the genuine general sentiment toward the product. In practice, this is not the case. Most people who purchase a product do not give it a review or a rating. The expectation is that the bulk of evaluations will be on either extreme, but it turns out that follow-up emails and marketing for a product can result in various reviews ranging from ordinary (3 stars) to exceptional (5 stars). When someone is writing a piece critiquing a product, on the other hand, the length of a text review is usually longer. Our EDA demonstrates this, and the dataset itself has a positive bias.

- The first stage is to check for duplicates, missing values, and data that is inconsistent.
- We estimate the importance of columns and drop/keep them as needed, mostly keeping only text data.
- Because the number of rows containing NANs was so little in comparison to the overall number of rows, we decided to remove them.
- The overall column (i.e., product rating) and the review column are the two key columns on which we focus our investigation. column of text (i.e., textual description and product review).
- Punctuation Elimination, the removal of redundant and special characters, such as punctuation, is a key duty in text normalization. The fundamental reason for this is that, while punctuation can provide useful insights into a review's emotion, even advanced language processing models such as BERT and GPT3 are unable to detect it. Because punctuation tends to throw the models off rather than aiding them in predicting the correct class, eliminating it is a popular approach.
- Num_words_text are removed, Num_words_text are the words that appear the most frequently when a corpus of text is aggregated based on solitary tokens and their frequencies are examined. Num_words_text include articles (a, the, and an), pronouns (you, them, they, me), and prepositions. They are insignificant at best. They are frequently eliminated from text during processing to preserve words with the most meaning and context.
- Stemming and lemmatization, Lemmatization is the process of reducing each word's inflectional forms to a single base or root. Stemming is a crude method that chops off the ends of words in

the hopes of reaching this goal most of the time, and it sometimes includes the removal of word formation units (the resulting part is known as the stem). Lemmatization is the process of returning the bottom or vocabulary sort of a word, that is known as the lemma, by appropriately using a vocabulary and morphological study of words.

Because the text is the least structured of all the access data, it contains numerous sorts of noise and is unusable without pre-processing. Text preprocessing refers to the process of cleaning and standardizing text to make it noise-free and ready for analysis. It is required to obtain any coherent results.

B2. Process of Tokenization:

Tokenization is the process of dividing or tokenizing a string of text into a list of tokens. Tokens are subunits of text document parts. They can be words, sentences, or phrases, among other things. For tokenization, we employed the Ngrams approach in our study. An n-gram is a contiguous sequence of n items from a text or audio sample, syllables, Phonemes, letters, words, and base pairs are common objects, depending on the appliance.

Tokenizer.sequences to matrix(x, mode='binary'): X = tokenizer.sequences to matrix(x, mode= If the word is present in the sample, the value of a word feature is 1, otherwise it is zero.

Count: X = tokenizer.sequences to matrix(x, mode='count'); the value represents the number of times a word appears in the phrase in this example.

Tokenizer.sequences to matrix(x, mode='tfidf'): TF-IDF: X = tokenizer.sequences to matrix(x, mode= In this case, we analyze the TF and IDF of the sample word in relation to the word's occurrence across the document.

B3. Process of padding:

In the padding process, each word is equal to a vector, each sample with a different number of words will have a different number of vectors. Now, to feed a model, each sample must have the same dimension, which necessitates padding to ensure that the number of words in each sample is equal.

Each text is converted into an integer sequence. In other words, if you had a sentence, it would assign an integer to each word in it. To validate the given integer to your word, use tokenizer.word index() (returns a dictionary). We're padding all sentences to a maximum length of 100 characters.

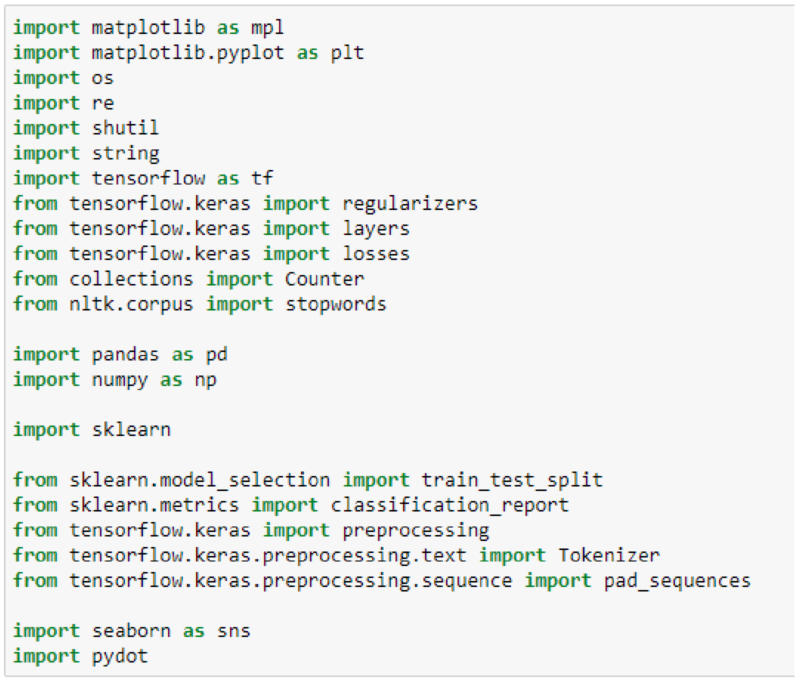
B4. Identifying Sentiments:

In this analysis, we used Recurrent neural network with LSTM model with sentiment processing and an embedding approach. The sentiments are categorized “Positive”, “Neutral” and “Negative”, “Overall” column has 1 to 5 ratings with Review text. “4 and 5” were Positive, 3 – Neutral and 1 and 2 were Negative” to get sentiment in out model, we created a simple method to get sentiments from overall reviews, basically “3, 4 and 5” rating is Positive and anything below which is “1 and 2: are considered “Negative”. Accuracy, a basic calculation of the percent of properly predicted labels, was utilized as the loss metric. Even with sentiment partitioning, the model accuracy on the test set never exceeded 56 percent, while approaching train accuracy.

B5. Data Preparation:

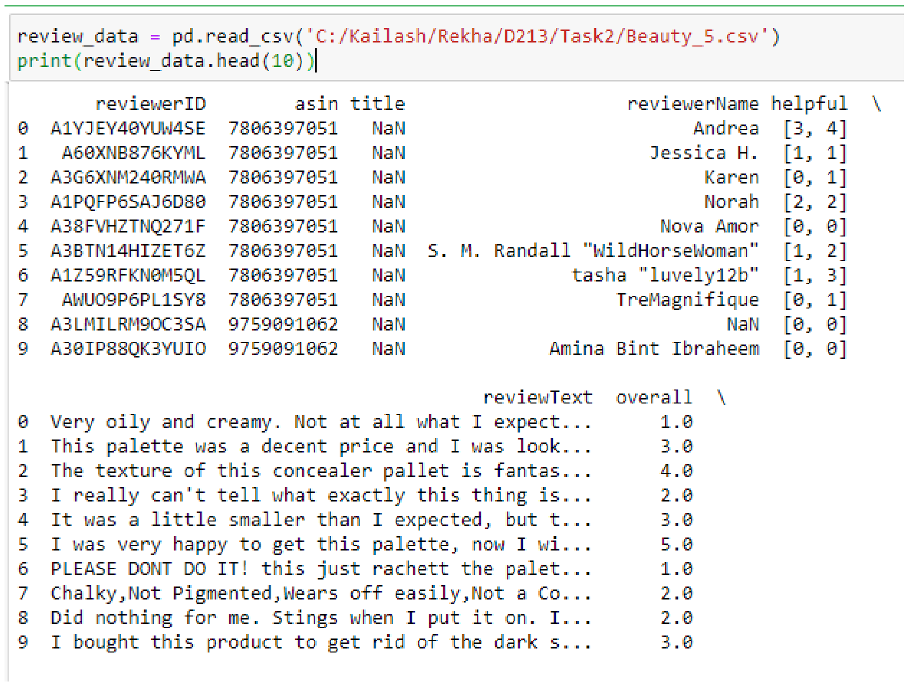
For this Sentimental analysis, we are using UCI data set from UCI Machine Learning Repository. This data set contains 3000 records with positive and negative sentiment available from 3 different websites: Amazon, IMDb, and Yelp. 500 positive sentences are labeled 1 and 500 negative sentences are labeled 0. Each website's information is saved in a separate txt file.

1. **Include standard imports and creating data frames:**



```
import matplotlib as mpl
import matplotlib.pyplot as plt
import os
import re
import shutil
import string
import tensorflow as tf
from tensorflow.keras import regularizers
from tensorflow.keras import layers
from tensorflow.keras import losses
from collections import Counter
from nltk.corpus import stopwords

import pandas as pd
import numpy as np

import sklearn

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras import preprocessing
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

import seaborn as sns
import pydot
```

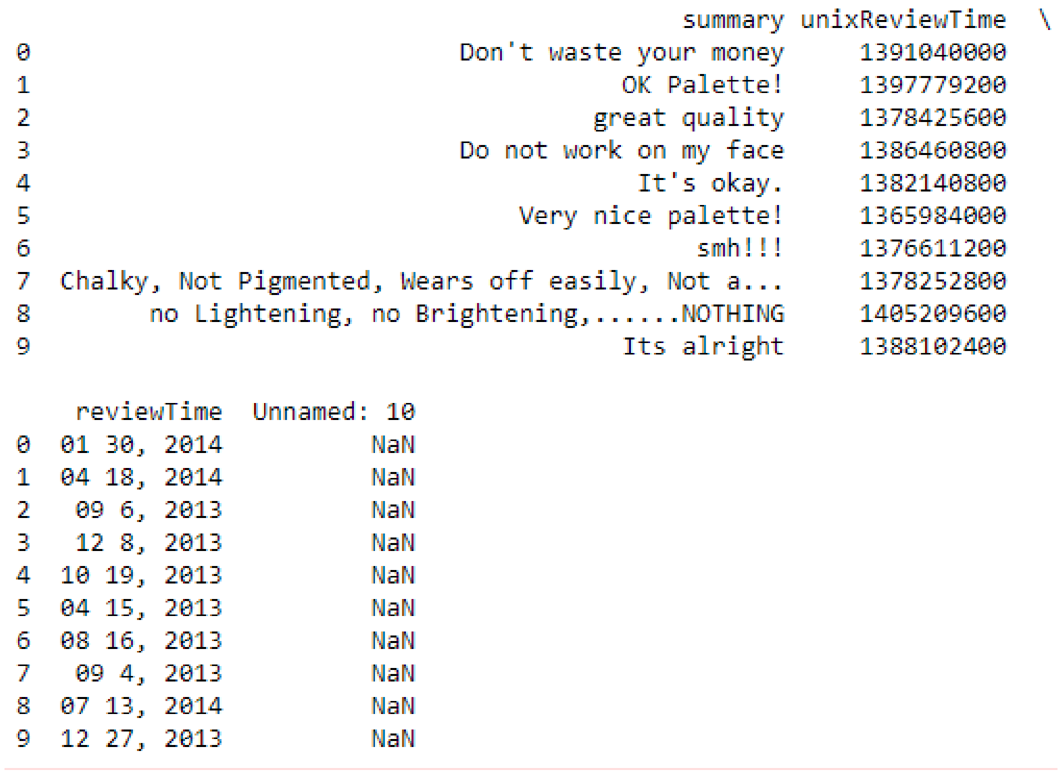
2. **Loaded the data set into pandas:**



```
review_data = pd.read_csv('C:/Kailash/Rekha/D213/Task2/Beauty_5.csv')
print(review_data.head(10))
```

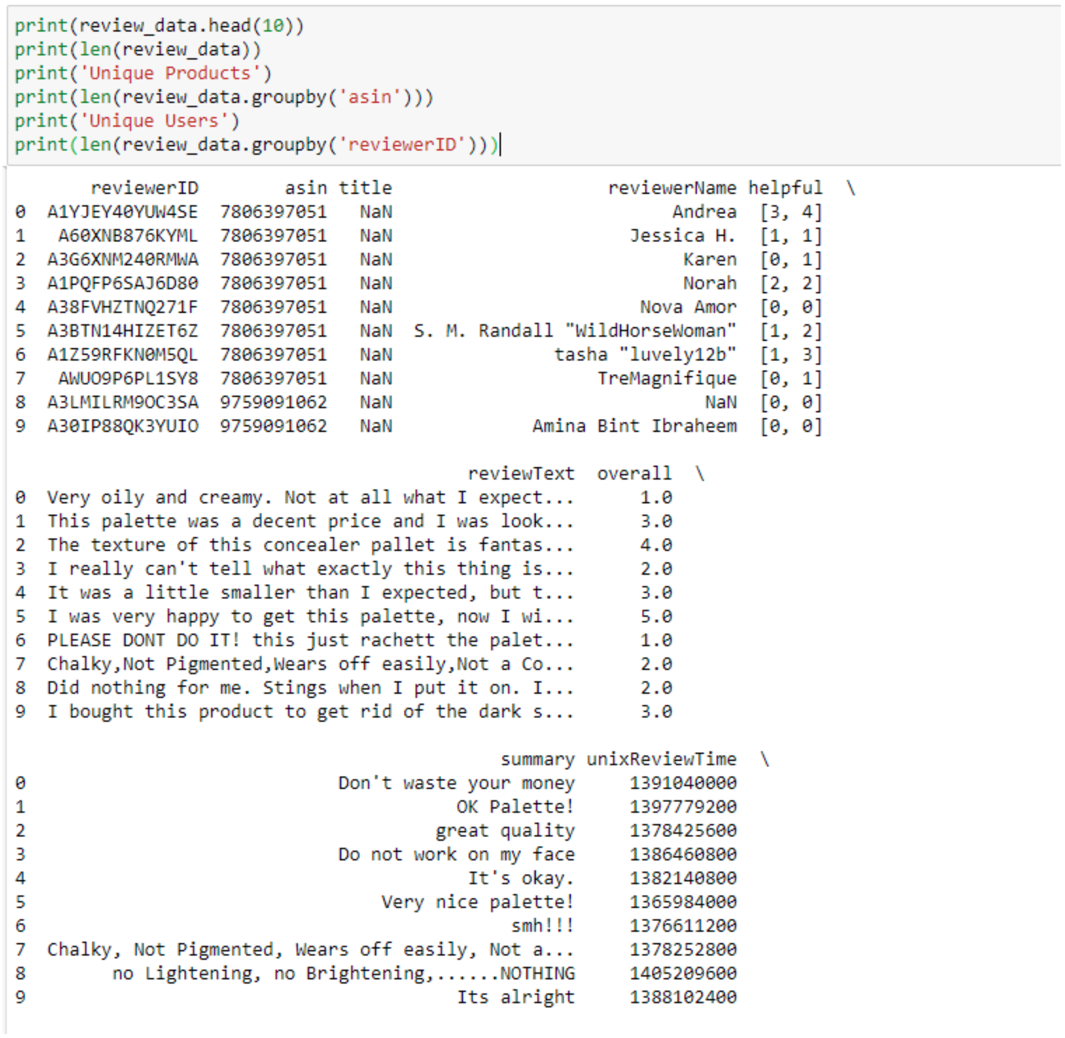
```
       reviewerID        asin title                      reviewerName helpful  \
0  A1YJEY40YUW4SE  7806397051   NaN                           Andrea  [3, 4]
1   A60XNB876KYML  7806397051   NaN                       Jessica H.  [1, 1]
2  A3G6XNM240RMWA  7806397051   NaN                            Karen  [0, 1]
3  A1PQFP6SAJ6D80  7806397051   NaN                            Norah  [2, 2]
4  A38FVHZTNQ271F  7806397051   NaN                        Nova Amor  [0, 0]
5  A3BTN14HIZET6Z  7806397051   NaN  S. M. Randall "WildHorseWoman"  [1, 2]
6  A1Z59RFKN0M5QL  7806397051   NaN                tasha "luvely12b"  [1, 3]
7   AWUO9P6PL1SY8  7806397051   NaN                     TreMagnifique  [0, 1]
8  A3LMILRM9OC3SA  9759091062   NaN                              NaN  [0, 0]
9  A30IP88QK3YUIO  9759091062   NaN               Amina Bint Ibraheem  [0, 0]

                                          reviewText  overall  \
0  Very oily and creamy. Not at all what I expect...      1.0
1  This palette was a decent price and I was look...      3.0
2  The texture of this concealer pallet is fantas...      4.0
3  I really can't tell what exactly this thing is...      2.0
4  It was a little smaller than I expected, but t...      3.0
5  I was very happy to get this palette, now I wi...      5.0
6  PLEASE DONT DO IT! this just rachett the palet...      1.0
7  Chalky,Not Pigmented,Wears off easily,Not a Co...      2.0
8  Did nothing for me. Stings when I put it on. I...      2.0
9  I bought this product to get rid of the dark s...      3.0
```

3. Loaded the data set into data frames:



```
                                           summary  unixReviewTime  \
0                            Don't waste your money      1391040000
1                                       OK Palette!      1397779200
2                                     great quality      1378425600
3                            Do not work on my face      1386460800
4                                        It's okay.      1382140800
5                                Very nice palette!      1365984000
6                                           smh!!!      1376611200
7  Chalky, Not Pigmented, Wears off easily, Not a...      1378252800
8        no Lightening, no Brightening,......NOTHING      1405209600
9                                       Its alright      1388102400

    reviewTime  Unnamed: 10
0  01 30, 2014          NaN
1  04 18, 2014          NaN
2   09 6, 2013          NaN
3   12 8, 2013          NaN
4  10 19, 2013          NaN
5  04 15, 2013          NaN
6  08 16, 2013          NaN
7   09 4, 2013          NaN
8  07 13, 2014          NaN
9  12 27, 2013          NaN
```

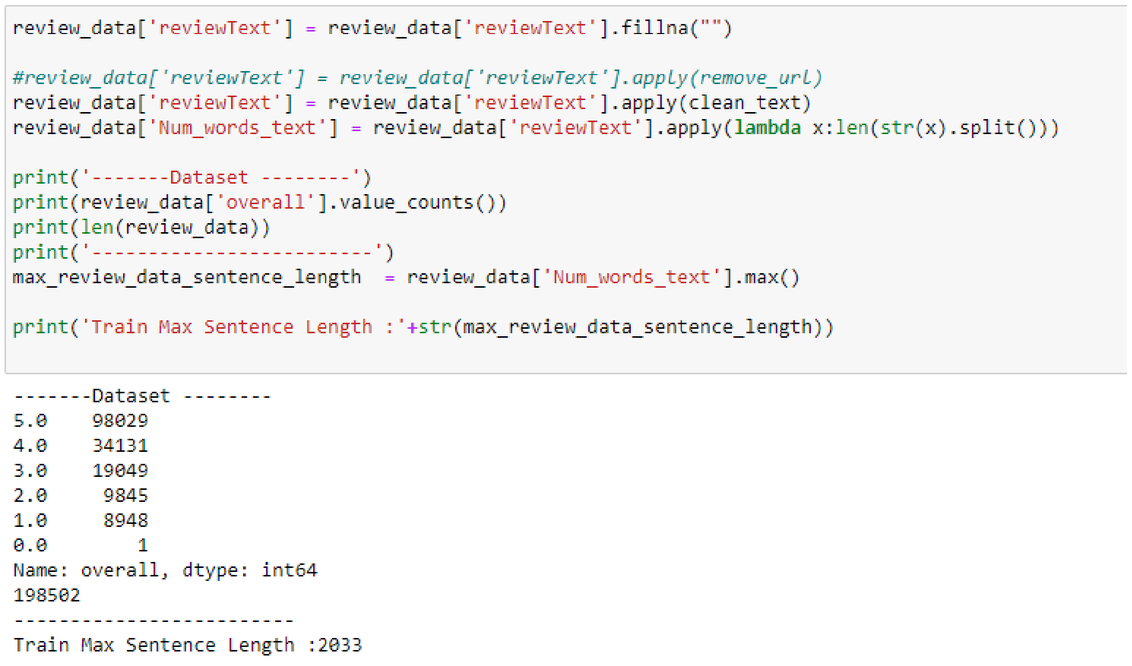
4. Examine the Amazon data frames and checking their dimensions:



```python
print(review_data.head(10))
print(len(review_data))
print('Unique Products')
print(len(review_data.groupby('asin')))
print('Unique Users')
print(len(review_data.groupby('reviewerID')))
```

```
       reviewerID        asin title                      reviewerName helpful  \
0  A1YJEY40YUW4SE  7806397051   NaN                            Andrea  [3, 4]
1   A60XNB876KYML  7806397051   NaN                        Jessica H.  [1, 1]
2  A3G6XNM240RMWA  7806397051   NaN                             Karen  [0, 1]
3  A1PQFP6SAJ6D80  7806397051   NaN                             Norah  [2, 2]
4  A38FVHZTNQ271F  7806397051   NaN                         Nova Amor  [0, 0]
5  A3BTN14HIZET6Z  7806397051   NaN  S. M. Randall "WildHorseWoman"  [1, 2]
6  A1Z59RFKN0M5QL  7806397051   NaN                 tasha "luvely12b"  [1, 3]
7   AWUO9P6PL1SY8  7806397051   NaN                      TreMagnifique  [0, 1]
8  A3LMILRM9OC3SA  9759091062   NaN                               NaN  [0, 0]
9  A3OIP88QK3YUIO  9759091062   NaN                Amina Bint Ibraheem  [0, 0]

                                          reviewText  overall  \
0  Very oily and creamy. Not at all what I expect...      1.0
1  This palette was a decent price and I was look...      3.0
2  The texture of this concealer pallet is fantas...      4.0
3  I really can't tell what exactly this thing is...      2.0
4  It was a little smaller than I expected, but t...      3.0
5  I was very happy to get this palette, now I wi...      5.0
6  PLEASE DONT DO IT! this just rachett the palet...      1.0
7  Chalky,Not Pigmented,Wears off easily,Not a Co...      2.0
8  Did nothing for me. Stings when I put it on. I...      2.0
9  I bought this product to get rid of the dark s...      3.0

                                             summary  unixReviewTime  \
0                               Don't waste your money      1391040000
1                                          OK Palette!      1397779200
2                                        great quality      1378425600
3                                  Do not work on my face      1386460800
4                                            It's okay.      1382140800
5                                      Very nice palette!      1365984000
6                                                smh!!!      1376611200
7  Chalky, Not Pigmented, Wears off easily, Not a...      1378252800
8        no Lightening, no Brightening,......NOTHING      1405209600
9                                          Its alright      1388102400
```

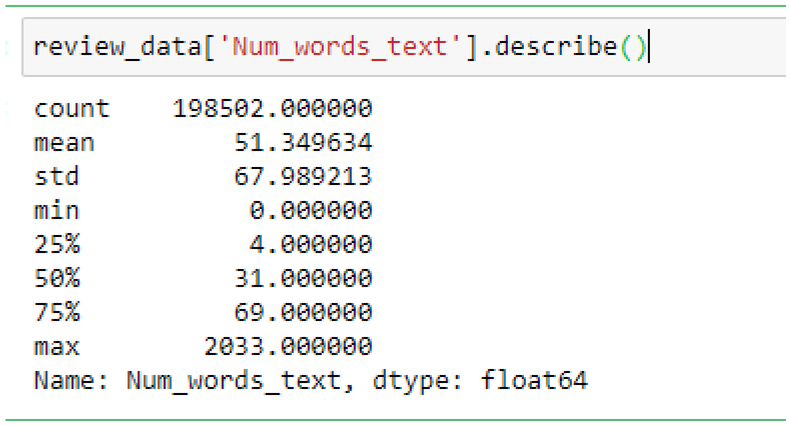
5. Clean the data set using clean_text function:



```
review_data['reviewText'] = review_data['reviewText'].fillna("")

#review_data['reviewText'] = review_data['reviewText'].apply(remove_url)
review_data['reviewText'] = review_data['reviewText'].apply(clean_text)
review_data['Num_words_text'] = review_data['reviewText'].apply(lambda x:len(str(x).split()))

print('-------Dataset --------')
print(review_data['overall'].value_counts())
print(len(review_data))
print('-------------------------')
max_review_data_sentence_length  = review_data['Num_words_text'].max()

print('Train Max Sentence Length :'+str(max_review_data_sentence_length))
```

```
-------Dataset --------
5.0    98029
4.0    34131
3.0    19049
2.0     9845
1.0     8948
0.0        1
Name: overall, dtype: int64
198502
-------------------------
Train Max Sentence Length :2033
```

6. Include descriptive statistics feature:



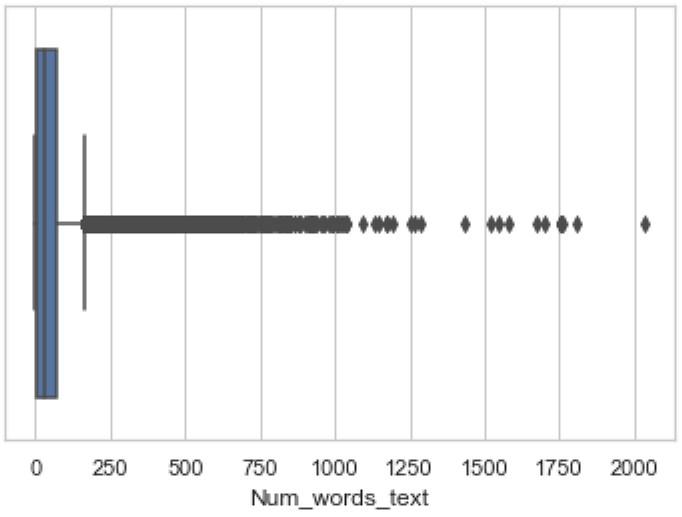
```
review_data['Num_words_text'].describe()
```

```
count    198502.000000
mean         51.349634
std          67.989213
min           0.000000
25%           4.000000
50%          31.000000
75%          69.000000
max        2033.000000
Name: Num_words_text, dtype: float64
```

7. Plot the sns:

```
sns.set(style="whitegrid")
sns.boxplot(x=review_data['Num_words_text'])
```

```
<AxesSubplot:xlabel='Num_words_text'>
```

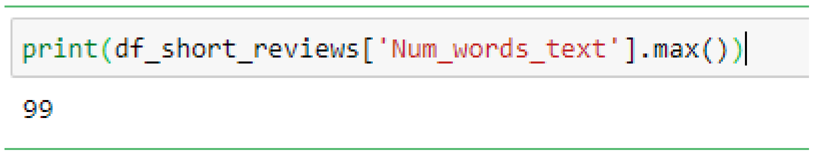


8. **Include the short reviews and long reviews:**

```
mask = (review_data['Num_words_text'] < 100) & (review_data['Num_words_text'] >=20)
df_short_reviews = review_data[mask]
print('No of Short reviews')
print(len(df_short_reviews))

mask = review_data['Num_words_text'] >= 100
df_long_reviews = review_data[mask]
print('No of Long reviews')
print(len(df_long_reviews))
```

```
No of Short reviews
99900
No of Long reviews
29478
```

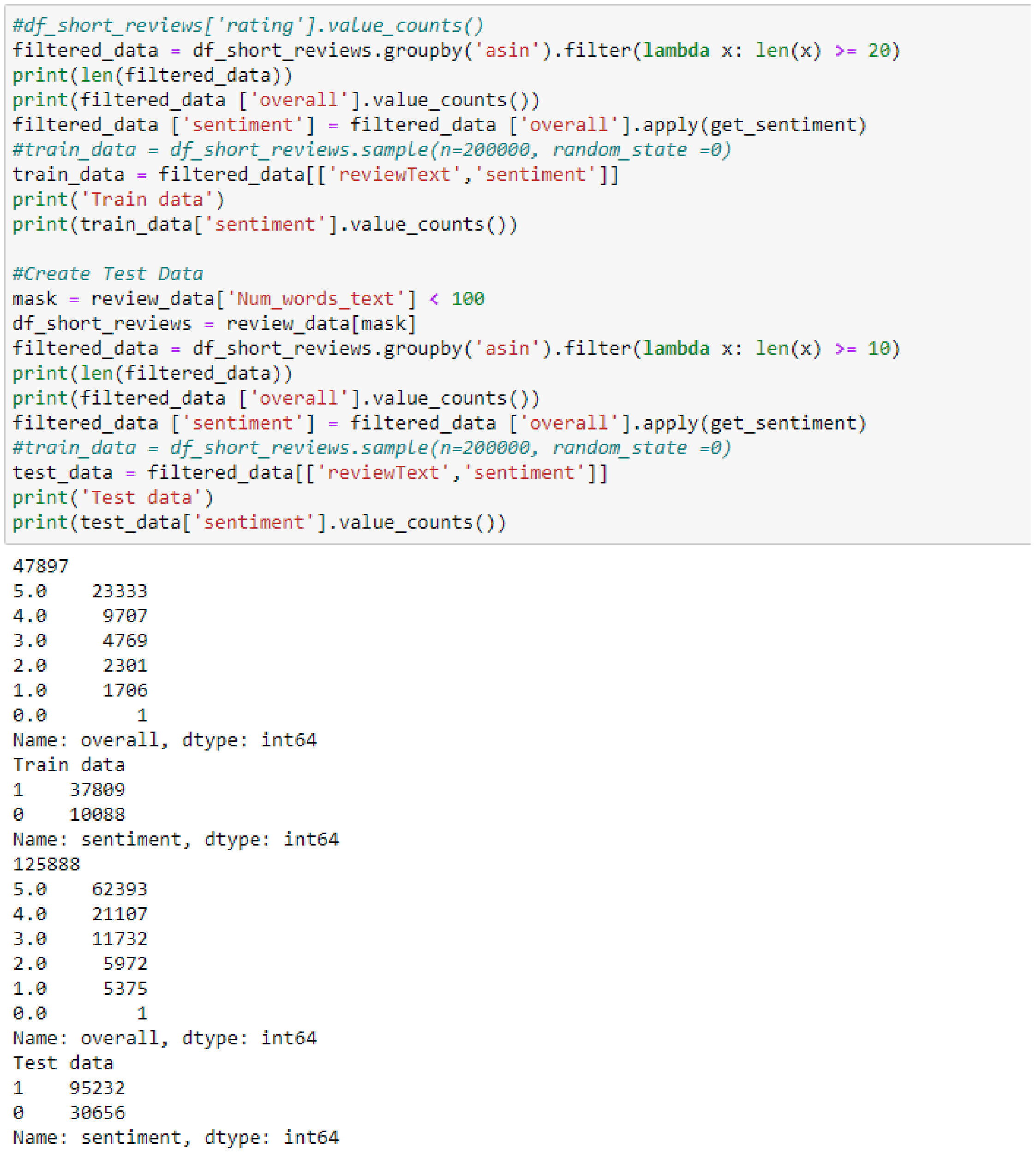
9. **Print the Short reviews:**



```
print(df_short_reviews['Num_words_text'].max())
```

```
99
```

10. **Split data set into train and test:**



```python
#df_short_reviews['rating'].value_counts()
filtered_data = df_short_reviews.groupby('asin').filter(lambda x: len(x) >= 20)
print(len(filtered_data))
print(filtered_data ['overall'].value_counts())
filtered_data ['sentiment'] = filtered_data ['overall'].apply(get_sentiment)
#train_data = df_short_reviews.sample(n=200000, random_state =0)
train_data = filtered_data[['reviewText','sentiment']]
print('Train data')
print(train_data['sentiment'].value_counts())

#Create Test Data
mask = review_data['Num_words_text'] < 100
df_short_reviews = review_data[mask]
filtered_data = df_short_reviews.groupby('asin').filter(lambda x: len(x) >= 10)
print(len(filtered_data))
print(filtered_data ['overall'].value_counts())
filtered_data ['sentiment'] = filtered_data ['overall'].apply(get_sentiment)
#train_data = df_short_reviews.sample(n=200000, random_state =0)
test_data = filtered_data[['reviewText','sentiment']]
print('Test data')
print(test_data['sentiment'].value_counts())
```

```
47897
5.0    23333
4.0     9707
3.0     4769
2.0     2301
1.0     1706
0.0        1
Name: overall, dtype: int64
Train data
1    37809
0    10088
Name: sentiment, dtype: int64
125888
5.0    62393
4.0    21107
3.0    11732
2.0     5972
1.0     5375
0.0        1
Name: overall, dtype: int64
Test data
1    95232
0    30656
Name: sentiment, dtype: int64
```

B6. Data set:

Please find the attached Prepared CSV

Beauty_review_Prep
ared.csv

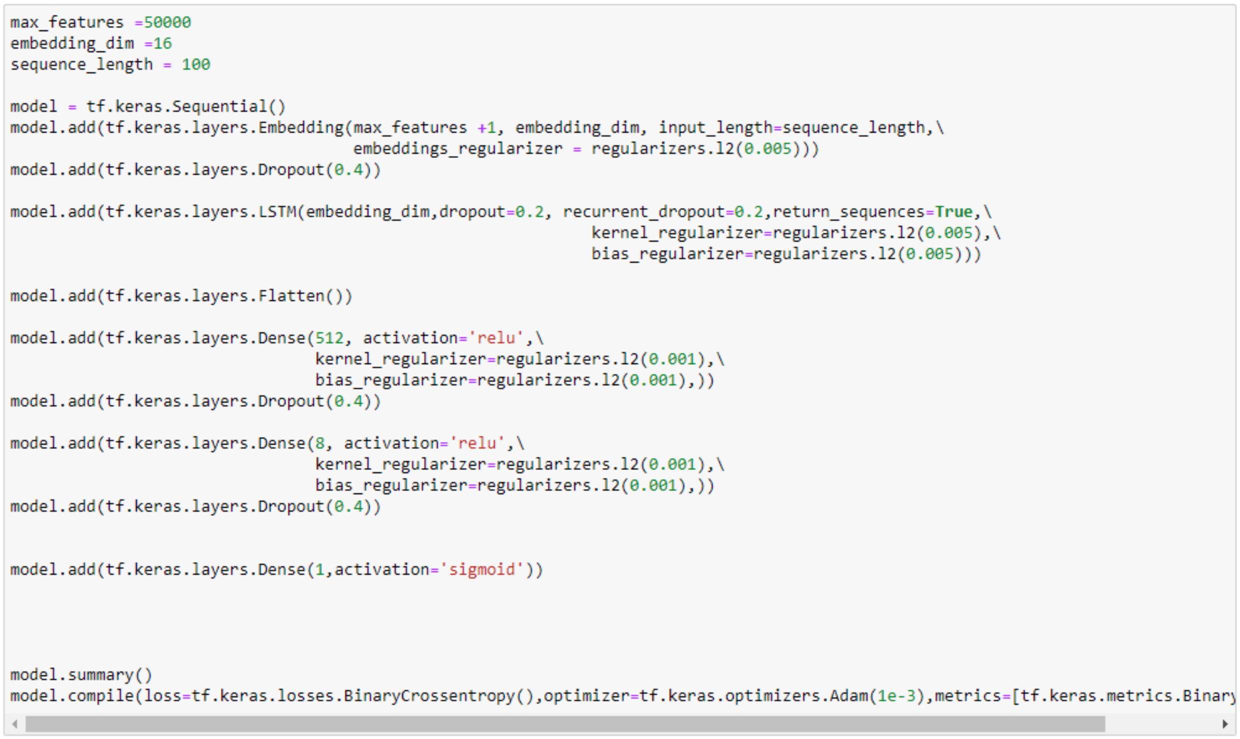
PART III: Architecture Network

C. Explain the Network Used for this Analysis:

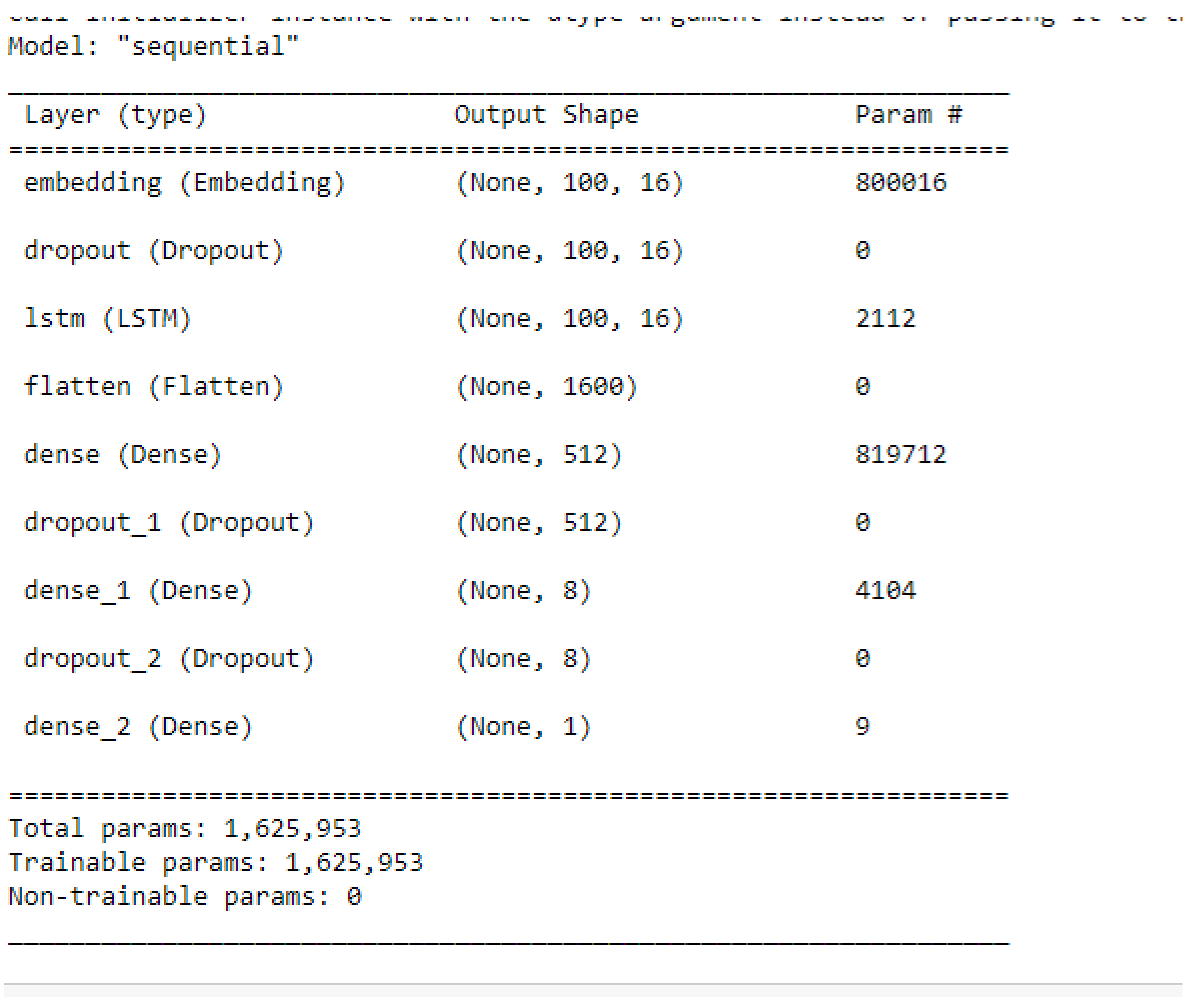
For this analysis, we used a bidirectional RNN with LSTMs to create our encoding layer. A recurrent neural network (RNN) is an artificial neural network that uses sequential or time series input to learn. We're using deep learning algorithms to process natural language (nlp). LSTM networks are a sort of recurrent neural network that can learn order dependence in sequence prediction issues.

C1. Overview of the Model TensorFlow:

LSTM (Long-Short Term Memory) neural networks are a sort of RNN that is built for long-term dependence and can store information for an extended length of time. When compared to traditional RNN, which may be used to predict future words based on previous word analysis, this method is more efficient.



```
max_features =50000
embedding_dim =16
sequence_length = 100

model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(max_features +1, embedding_dim, input_length=sequence_length,\
                                    embeddings_regularizer = regularizers.l2(0.005)))
model.add(tf.keras.layers.Dropout(0.4))

model.add(tf.keras.layers.LSTM(embedding_dim,dropout=0.2, recurrent_dropout=0.2,return_sequences=True,\
                                                             kernel_regularizer=regularizers.l2(0.005),\
                                                             bias_regularizer=regularizers.l2(0.005)))


model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(512, activation='relu',\
                                kernel_regularizer=regularizers.l2(0.001),\
                                bias_regularizer=regularizers.l2(0.001),))
model.add(tf.keras.layers.Dropout(0.4))

model.add(tf.keras.layers.Dense(8, activation='relu',\
                                kernel_regularizer=regularizers.l2(0.001),\
                                bias_regularizer=regularizers.l2(0.001),))
model.add(tf.keras.layers.Dropout(0.4))


model.add(tf.keras.layers.Dense(1,activation='sigmoid'))




model.summary()
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),optimizer=tf.keras.optimizers.Adam(1e-3),metrics=[tf.keras.metrics.Binary
```



```
call initializer instance with the dtype argument instead of passing it to t
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           800016

 dropout (Dropout)           (None, 100, 16)           0

 lstm (LSTM)                 (None, 100, 16)           2112

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 512)               819712

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 8)                 4104

 dropout_2 (Dropout)         (None, 8)                 0

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 1,625,953
Trainable params: 1,625,953
Non-trainable params: 0
_________________________________________________________________
```

For this project, an LSTM model was created for sentimental analysis.

- The output shape (100,16) is used to generate an embedding layer. This guarantees that the words are fed into the model according to the sentiment-specific vocabulary generated by word embeddings.
- To avoid the problem of overfitting when data advances further into the layers of the model, two dropout layers are added before and after the LSTM layer.
- The model generates an LSTM bidirectional layer of output shape (128) for the categorization of positive or negative emotion included in the words.
- To capture the output of the LSTM effectively and to stabilize the state of the model throughout 3 epochs, two dense layers of output shape (,64) and (,1) are built.

C2. Type of Layers:

Layers are the core features of the TensorFlow. Here is the list of different available layers

- **Core Layer:**
  input_data, fully_connected, dropout, custom_layer, reshape, flatten, activation, single_unit, highway, one_hot_encoding and time_distributed.
- **Conv Layer:**
  Conv_2d, Conv_2d_transpose_max_pool_2d, avg_pool_2d, upsample_2d, Conv_1d, max_pool_1d, avg_pool_1d, residual_block, residual_bottleneck, conv_3d, max_pool_3d, avg_pool_3d, highway_conv_1d, highway_conv_2d, global_avg_pool, and global_max_pool.

- **Recurrent Layer:**
  Simple_rnn, lstm, gru, bidirectional_rnn and dynamic_rnn

- **Embedding Layer:**
  Embedding

- **Normalization Layer:**
  Batch_normalization, local_response_normalization and 12_normalize

- **Merge Layer:**
  Merge and merge_outputs

- **Estimator Layer:**
  regression

In this analysis, we used Recurrent layer with

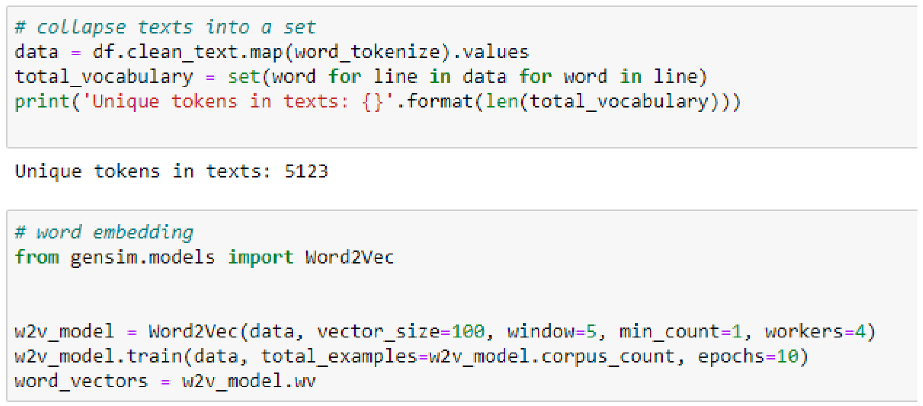
- lstm with (embedding_dim,dropout=0.2, recurrent_dropout=0.2,return_sequences=True) and
- Embedding with embedding_dim,dropout=0.2, recurrent_dropout=0.2,return_sequences=True,
- Dropout(0,4)
- Bidirectional layer with LSTM(64)
- Dense layer with (64, activation='relu')

C3. Hyper Parameters:

In this analysis, we used hyper parameter combination methods RNN-LSTM through token vectors

- Activation Functions:
  An artificial neural network's activation function is a function that is introduced to aid the network's learning of complicated patterns in data. When compared to a neuron-based model seen in our brains, the activation function determines what is to be fired to the following neuron at the end. In an ANN, an activation function performs the same job. It takes the preceding cell's output signal and turns it into a format that may be used as input to the next cell.

  In our analysis, we conducted, the Word2Vec neural network uses the softmax activation function for multiclass classification and consists of an input layer, a hidden layer, and an output layer. Each unique word in the lexicon is represented by a neuron in the network's input layer. Each of the input vocabulary items has a different weight, and the number of neurons in the hidden layer is determined by the required word vector size.



```
# collapse texts into a set
data = df.clean_text.map(word_tokenize).values
total_vocabulary = set(word for line in data for word in line)
print('Unique tokens in texts: {}'.format(len(total_vocabulary)))
```

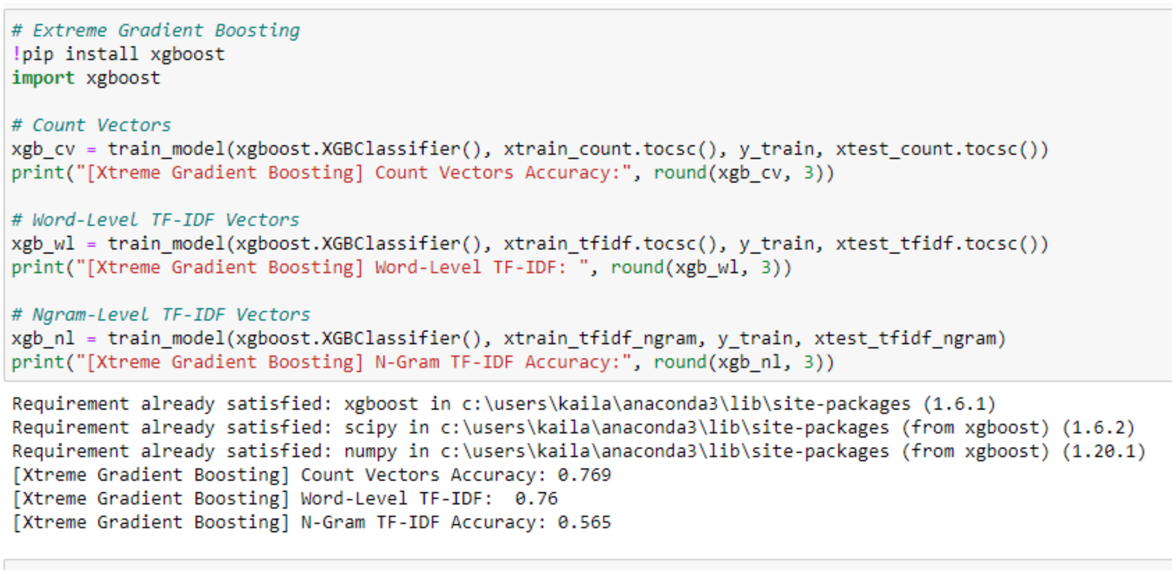
```
Unique tokens in texts: 5123
```

```
# word embedding
from gensim.models import Word2Vec


w2v_model = Word2Vec(data, vector_size=100, window=5, min_count=1, workers=4)
w2v_model.train(data, total_examples=w2v_model.corpus_count, epochs=10)
word_vectors = w2v_model.wv
```

- Number of nodes per layer
  You achieved greater training and accuracy when we reduced the number of neurons in each dense layer. In this situation, we were able to lessen the overfitting effect. Removing some neurons from our layers acted as a regularizer on our problem, reducing overfitting. This is a regular occurrence; depending on your dataset and overall neural network architecture, reducing the number of neurons in particular layers may result in improved generalization of your model.

In our model, When the number of nodes was reduced, the training accuracy improved, because overfitting improves training accuracy but decreases test/holdout accuracy.

- Loss function
  Whereas bagging selects a random sample of data for use in learning algorithms, boosting selects a sample with a higher level of inspection. Boosting is an ensemble strategy that uses a gradient descent algorithm to minimize a loss function in new models to correct for errors committed in previous models. Additional models are added until no further advancements are possible. XGB's predictions are represented by the categories in the final model.



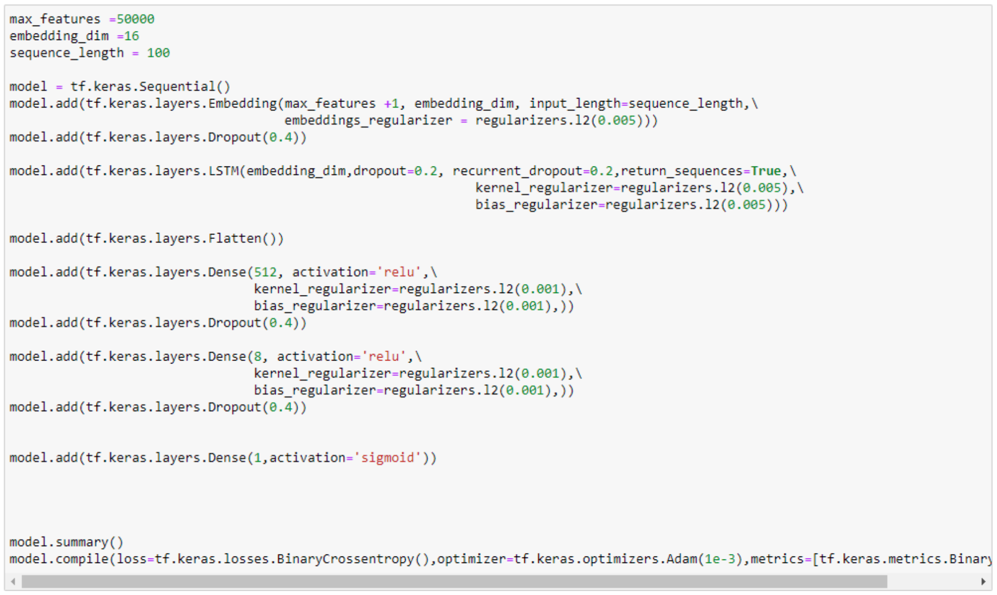
```
# Extreme Gradient Boosting
!pip install xgboost
import xgboost

# Count Vectors
xgb_cv = train_model(xgboost.XGBClassifier(), xtrain_count.tocsc(), y_train, xtest_count.tocsc())
print("[Xtreme Gradient Boosting] Count Vectors Accuracy:", round(xgb_cv, 3))

# Word-Level TF-IDF Vectors
xgb_wl = train_model(xgboost.XGBClassifier(), xtrain_tfidf.tocsc(), y_train, xtest_tfidf.tocsc())
print("[Xtreme Gradient Boosting] Word-Level TF-IDF: ", round(xgb_wl, 3))

# Ngram-Level TF-IDF Vectors
xgb_nl = train_model(xgboost.XGBClassifier(), xtrain_tfidf_ngram, y_train, xtest_tfidf_ngram)
print("[Xtreme Gradient Boosting] N-Gram TF-IDF Accuracy:", round(xgb_nl, 3))
```

```
Requirement already satisfied: xgboost in c:\users\kaila\anaconda3\lib\site-packages (1.6.1)
Requirement already satisfied: scipy in c:\users\kaila\anaconda3\lib\site-packages (from xgboost) (1.6.2)
Requirement already satisfied: numpy in c:\users\kaila\anaconda3\lib\site-packages (from xgboost) (1.20.1)
[Xtreme Gradient Boosting] Count Vectors Accuracy: 0.769
[Xtreme Gradient Boosting] Word-Level TF-IDF:  0.76
[Xtreme Gradient Boosting] N-Gram TF-IDF Accuracy: 0.565
```

In our RNN-LSTM model also, due to rising validation loss, the model was halted after 4 epochs. On the validation set, the model achieved an overall accuracy of around 64%. However, because there are four categories, we must dissect them to see how the model fared.

- Optimizer
  Optimisers are critical in improving the model's accuracy. There are a variety of optimiser options available. Descent of a Stochastic Gradient, Nesterov accelerated gradient, Adagrad, AdaDelta, RMSProp, `Adam.`

In our model we used adam optimizer.



```
max_features =50000
embedding_dim =16
sequence_length = 100

model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(max_features +1, embedding_dim, input_length=sequence_length,\
                                    embeddings_regularizer = regularizers.l2(0.005)))
model.add(tf.keras.layers.Dropout(0.4))

model.add(tf.keras.layers.LSTM(embedding_dim,dropout=0.2, recurrent_dropout=0.2,return_sequences=True,\
                                                             kernel_regularizer=regularizers.l2(0.005),\
                                                             bias_regularizer=regularizers.l2(0.005)))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(512, activation='relu',\
                                kernel_regularizer=regularizers.l2(0.001),\
                                bias_regularizer=regularizers.l2(0.001),))
model.add(tf.keras.layers.Dropout(0.4))

model.add(tf.keras.layers.Dense(8, activation='relu',\
                                kernel_regularizer=regularizers.l2(0.001),\
                                bias_regularizer=regularizers.l2(0.001),))
model.add(tf.keras.layers.Dropout(0.4))


model.add(tf.keras.layers.Dense(1,activation='sigmoid'))



model.summary()
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),optimizer=tf.keras.optimizers.Adam(1e-3),metrics=[tf.keras.metrics.Binary
```

- Stopping criteria
  Early stopping is a technique that allows you to provide an arbitrary large number of training epochs and then stop training when the model's performance on the validation dataset stops improving.
  In our model, is good fit on training as well as validation data set

- Evaluation metric
  Attached the Evaluation Metrix.

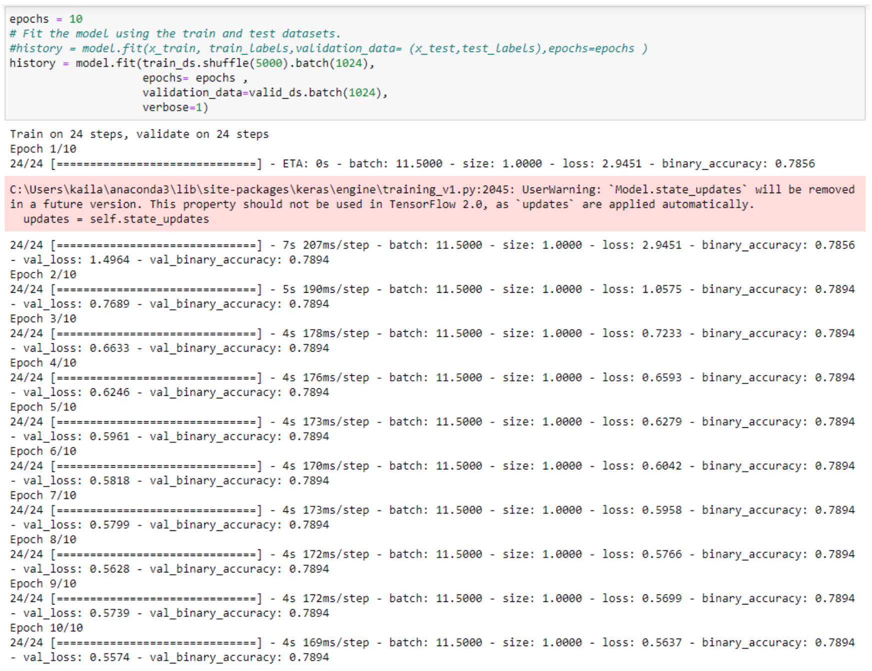
PART IV. Assessment of the Model

D1. Consequences of stopping criteria:

Early stopping prevents overfitting. Early stopping really tries to do is keep you from training the neural network too far. Early termination of training prevents the neural network from overfitting. In general, an overfit neural network will have too many epochs, while an underfit neural network will have too few.

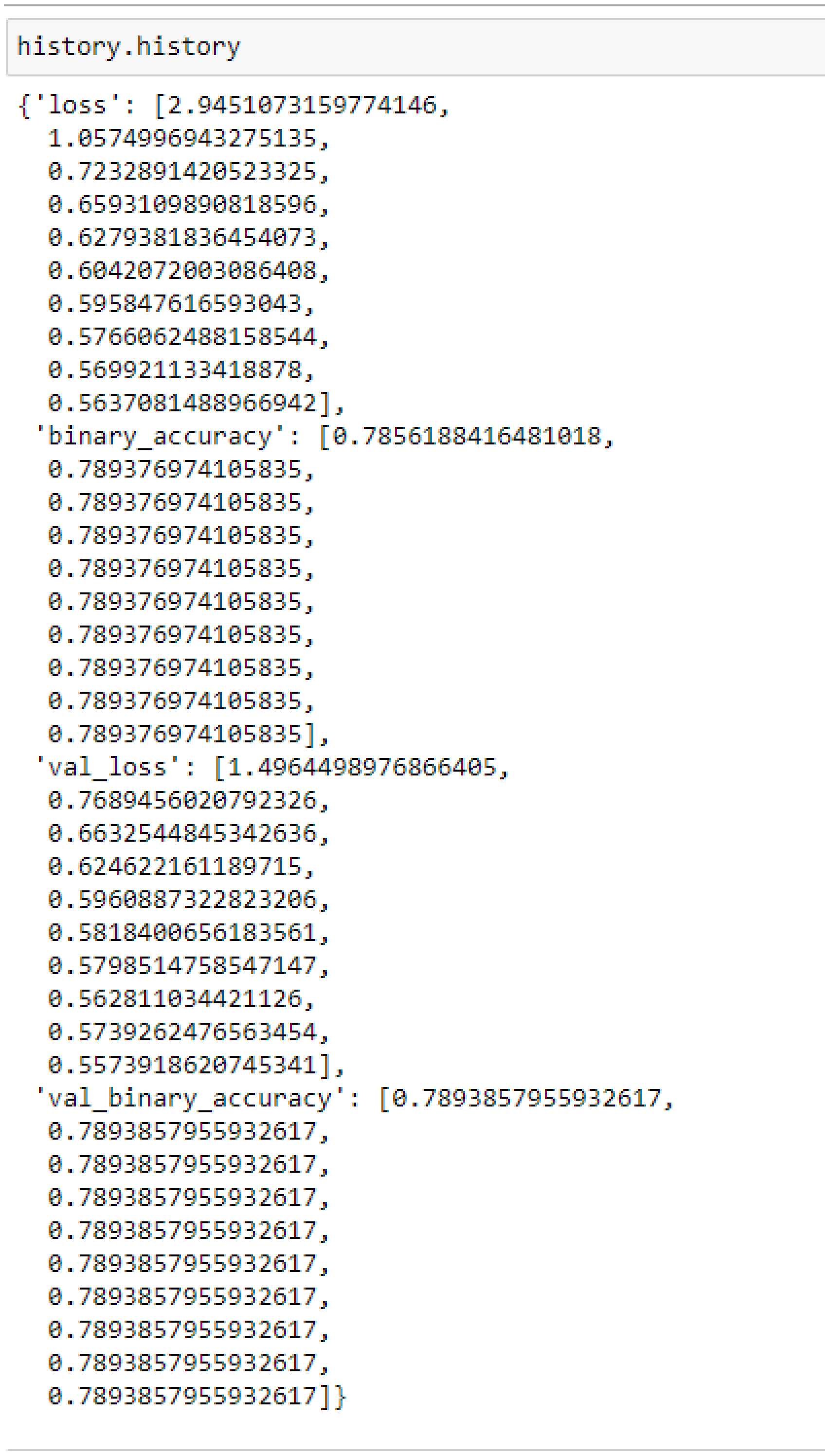
A stopping condition that forces your algorithm to end the process. It can be any meaningful condition you choose, such as the number of iterations, the quality of the solutions, statistical numbers, an external or internal condition, possibly a random termination key, and so on. You should judge an optimization algorithm's "average performance" over numerous independent runs with random initialization to acquire trustworthy findings. When you run your optimizer 100 times, the average fitness values across all runs can demonstrate the algorithm's capacity to solve that problem.

In this analysis, we used to train and test the data set to fit the model, we used 10 epochs, and we don’t see any overfitting in our model.



```python
epochs = 10
# Fit the model using the train and test datasets.
#history = model.fit(x_train, train_labels,validation_data= (x_test,test_labels),epochs=epochs )
history = model.fit(train_ds.shuffle(5000).batch(1024),
                    epochs= epochs ,
                    validation_data=valid_ds.batch(1024),
                    verbose=1)
```

```
Train on 24 steps, validate on 24 steps
Epoch 1/10
24/24 [==============================] - ETA: 0s - batch: 11.5000 - size: 1.0000 - loss: 2.9451 - binary_accuracy: 0.7856

C:\Users\kaila\anaconda3\lib\site-packages\keras\engine\training_v1.py:2045: UserWarning: `Model.state_updates` will be removed
in a future version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  updates = self.state_updates

24/24 [==============================] - 7s 207ms/step - batch: 11.5000 - size: 1.0000 - loss: 2.9451 - binary_accuracy: 0.7856
- val_loss: 1.4964 - val_binary_accuracy: 0.7894
Epoch 2/10
24/24 [==============================] - 5s 190ms/step - batch: 11.5000 - size: 1.0000 - loss: 1.0575 - binary_accuracy: 0.7894
- val_loss: 0.7689 - val_binary_accuracy: 0.7894
Epoch 3/10
24/24 [==============================] - 4s 178ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.7233 - binary_accuracy: 0.7894
- val_loss: 0.6633 - val_binary_accuracy: 0.7894
Epoch 4/10
24/24 [==============================] - 4s 176ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.6593 - binary_accuracy: 0.7894
- val_loss: 0.6246 - val_binary_accuracy: 0.7894
Epoch 5/10
24/24 [==============================] - 4s 173ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.6279 - binary_accuracy: 0.7894
- val_loss: 0.5961 - val_binary_accuracy: 0.7894
Epoch 6/10
24/24 [==============================] - 4s 170ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.6042 - binary_accuracy: 0.7894
- val_loss: 0.5818 - val_binary_accuracy: 0.7894
Epoch 7/10
24/24 [==============================] - 4s 173ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5958 - binary_accuracy: 0.7894
- val_loss: 0.5799 - val_binary_accuracy: 0.7894
Epoch 8/10
24/24 [==============================] - 4s 172ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5766 - binary_accuracy: 0.7894
- val_loss: 0.5628 - val_binary_accuracy: 0.7894
Epoch 9/10
24/24 [==============================] - 4s 172ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5699 - binary_accuracy: 0.7894
- val_loss: 0.5739 - val_binary_accuracy: 0.7894
Epoch 10/10
24/24 [==============================] - 4s 169ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5637 - binary_accuracy: 0.7894
- val_loss: 0.5574 - val_binary_accuracy: 0.7894
```



```
history.history
```

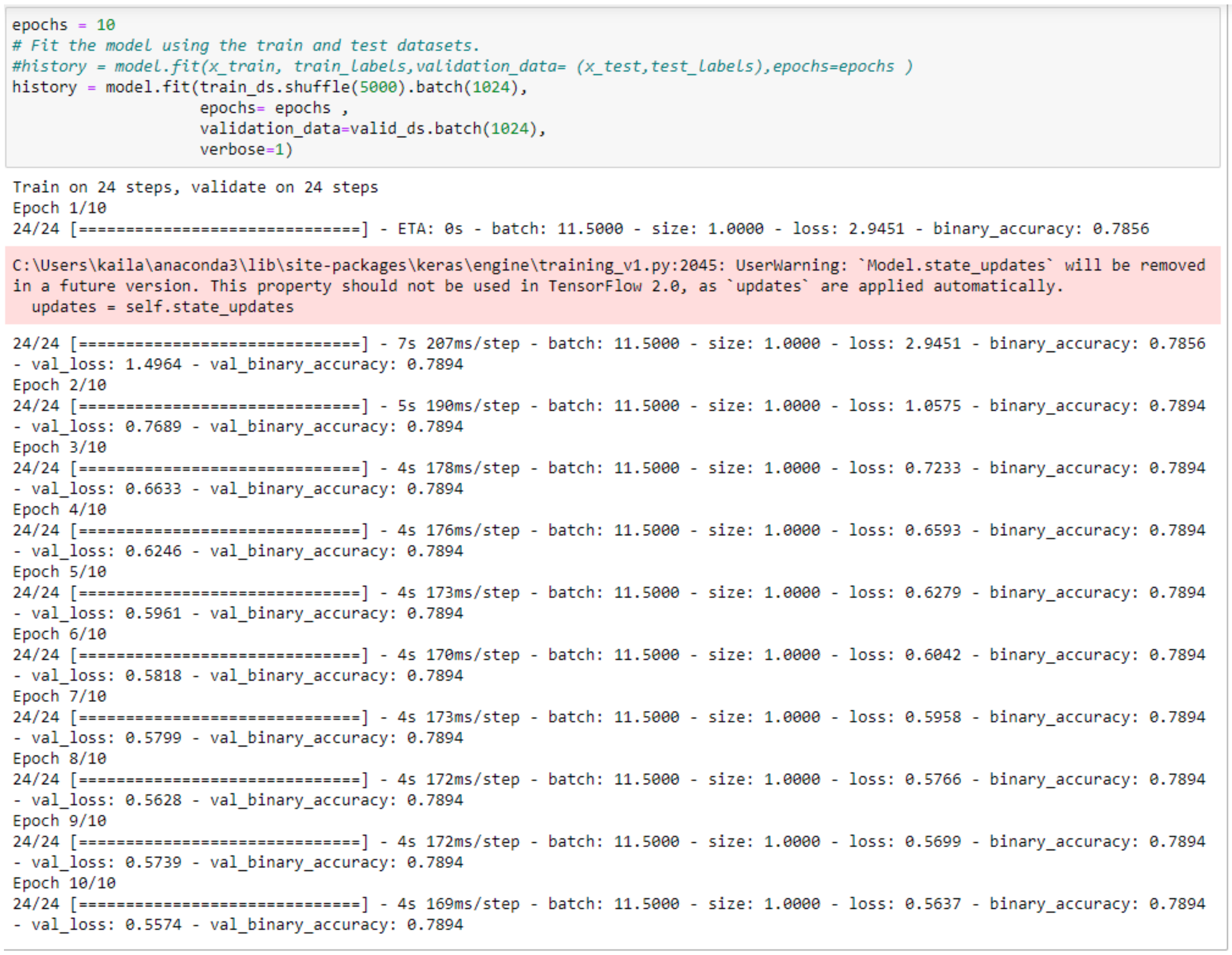
```
{'loss': [2.9451073159774146,
  1.0574996943275135,
  0.7232891420523325,
  0.6593109890818596,
  0.6279381836454073,
  0.6042072003086408,
  0.595847616593043,
  0.5766062488158544,
  0.569921133418878,
  0.5637081488966942],
 'binary_accuracy': [0.7856188416481018,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835],
 'val_loss': [1.4964498976866405,
  0.7689456020792326,
  0.6632544845342636,
  0.624622161189715,
  0.5960887322823206,
  0.5818400656183561,
  0.5798514758547147,
  0.562811034421126,
  0.5739262476563454,
  0.5573918620745341],
 'val_binary_accuracy': [0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617]}
```

D2. Model's Training Process Visualizations:

In this sentimental analysis, with our LSTM model training process, a a portion of the training dataset chosen at random from the entire dataset, which consist of 23,948 reviews was extracted.

In this we used 10 epochs, we used batch size (1024), we shuffled and pick this batch, we also have validation data set with similar batch size. We just called model. Fit on training as well as validation data set.

If you see the accuracy on train data is 0.78 and accuracy on validation data 0.7894, after 10 epochs accuracy on train data is 0.7894 and accuracy on validation data is 0.78, because we assigned model.fit to the history variable will get the dictionary where we have loss, training accuracy and validation accuracy. We plot the model.

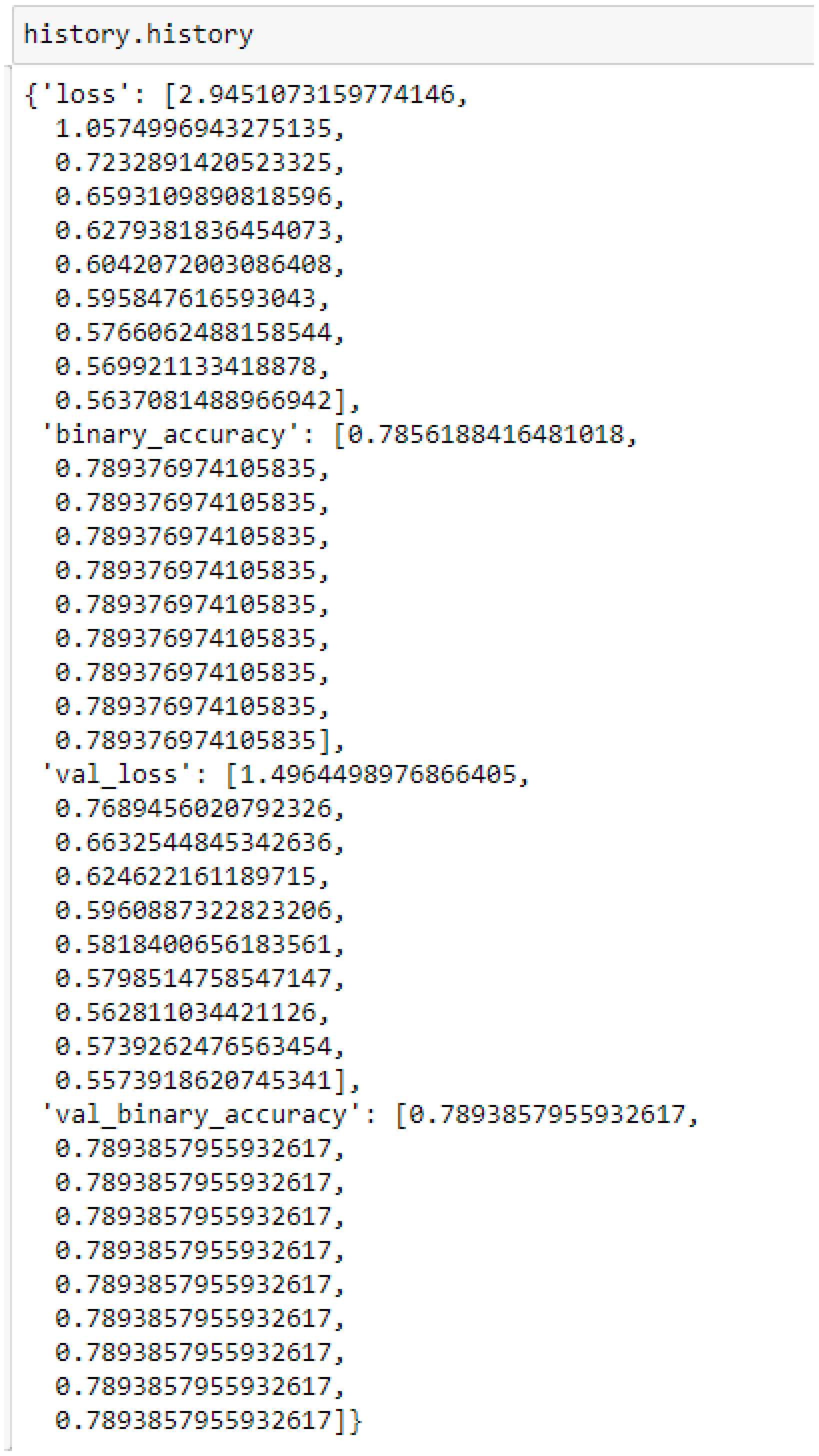


```
epochs = 10
# Fit the model using the train and test datasets.
#history = model.fit(x_train, train_labels,validation_data= (x_test,test_labels),epochs=epochs )
history = model.fit(train_ds.shuffle(5000).batch(1024),
                    epochs= epochs ,
                    validation_data=valid_ds.batch(1024),
                    verbose=1)
```

```
Train on 24 steps, validate on 24 steps
Epoch 1/10
24/24 [==============================] - ETA: 0s - batch: 11.5000 - size: 1.0000 - loss: 2.9451 - binary_accuracy: 0.7856
```

```
C:\Users\kaila\anaconda3\lib\site-packages\keras\engine\training_v1.py:2045: UserWarning: `Model.state_updates` will be removed
in a future version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  updates = self.state_updates
```

```
24/24 [==============================] - 7s 207ms/step - batch: 11.5000 - size: 1.0000 - loss: 2.9451 - binary_accuracy: 0.7856
- val_loss: 1.4964 - val_binary_accuracy: 0.7894
Epoch 2/10
24/24 [==============================] - 5s 190ms/step - batch: 11.5000 - size: 1.0000 - loss: 1.0575 - binary_accuracy: 0.7894
- val_loss: 0.7689 - val_binary_accuracy: 0.7894
Epoch 3/10
24/24 [==============================] - 4s 178ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.7233 - binary_accuracy: 0.7894
- val_loss: 0.6633 - val_binary_accuracy: 0.7894
Epoch 4/10
24/24 [==============================] - 4s 176ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.6593 - binary_accuracy: 0.7894
- val_loss: 0.6246 - val_binary_accuracy: 0.7894
Epoch 5/10
24/24 [==============================] - 4s 173ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.6279 - binary_accuracy: 0.7894
- val_loss: 0.5961 - val_binary_accuracy: 0.7894
Epoch 6/10
24/24 [==============================] - 4s 170ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.6042 - binary_accuracy: 0.7894
- val_loss: 0.5818 - val_binary_accuracy: 0.7894
Epoch 7/10
24/24 [==============================] - 4s 173ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5958 - binary_accuracy: 0.7894
- val_loss: 0.5799 - val_binary_accuracy: 0.7894
Epoch 8/10
24/24 [==============================] - 4s 172ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5766 - binary_accuracy: 0.7894
- val_loss: 0.5628 - val_binary_accuracy: 0.7894
Epoch 9/10
24/24 [==============================] - 4s 172ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5699 - binary_accuracy: 0.7894
- val_loss: 0.5739 - val_binary_accuracy: 0.7894
Epoch 10/10
24/24 [==============================] - 4s 169ms/step - batch: 11.5000 - size: 1.0000 - loss: 0.5637 - binary_accuracy: 0.7894
- val_loss: 0.5574 - val_binary_accuracy: 0.7894
```



```
history.history
```

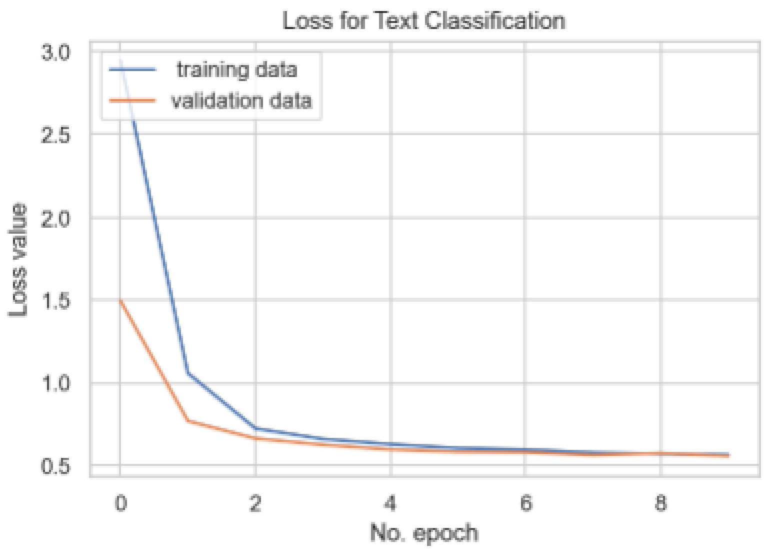
```
{'loss': [2.9451073159774146,
  1.0574996943275135,
  0.7232891420523325,
  0.6593109890818596,
  0.6279381836454073,
  0.6042072003086408,
  0.595847616593043,
  0.5766062488158544,
  0.569921133418878,
  0.5637081488966942],
 'binary_accuracy': [0.7856188416481018,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835,
  0.789376974105835],
 'val_loss': [1.4964498976866405,
  0.7689456020792326,
  0.6632544845342636,
  0.624622161189715,
  0.5960887322823206,
  0.5818400656183561,
  0.5798514758547147,
  0.562811034421126,
  0.5739262476563454,
  0.5573918620745341],
 'val_binary_accuracy': [0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617,
  0.7893857955932617]}
```

Then we did Loss for text classification

```
plt.plot(history.history['loss'], label=' training data')
plt.plot(history.history['val_loss'], label='validation data')
plt.title('Loss for Text Classification')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()
```

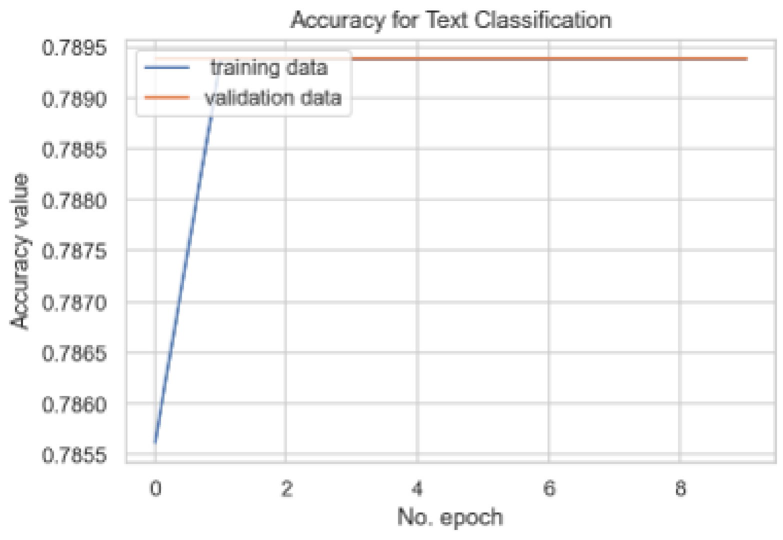
Plotting training data with validation data

```
plt.plot(history.history['loss'], label=' training data')
plt.plot(history.history['val_loss'], label='validation data')
plt.title('Loss for Text Classification')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()
```



Same way we can do the accuracy classification using training and validation data sets

```
plt.plot(history.history['binary_accuracy'], label=' training data')
plt.plot(history.history['val_binary_accuracy'], label='validation data')
plt.title('Accuracy for Text Classification')
plt.ylabel('Accuracy value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()
```

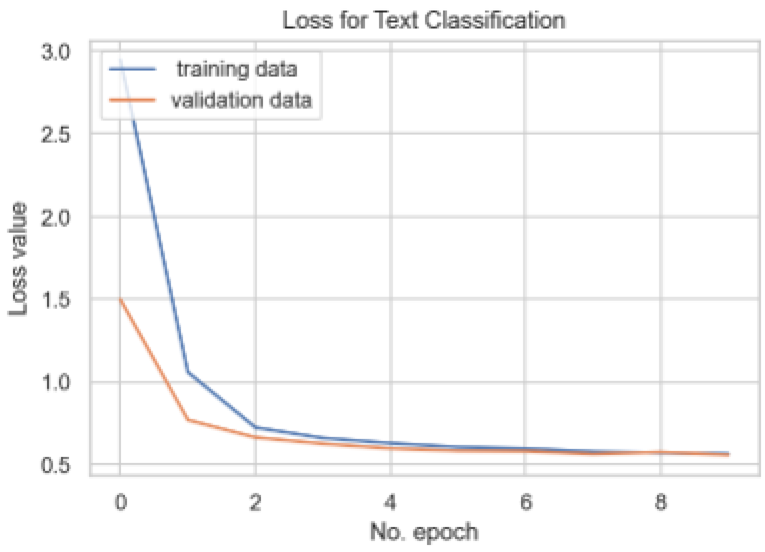


D3. Model's efficiency and any overfitting samples:

An overfit model is one in which performance on the train set is good and increases over time, whereas efficiency on the validation set increases to a point before degrading. When the train loss and validation loss sloping down, hitting an inflection point, and then sloping up again can be used to detect this. This could indicate that there are too many training epochs. The model training might be interrupted at the inflection point. Alternatively, you might increase the number of training examples in those situations.

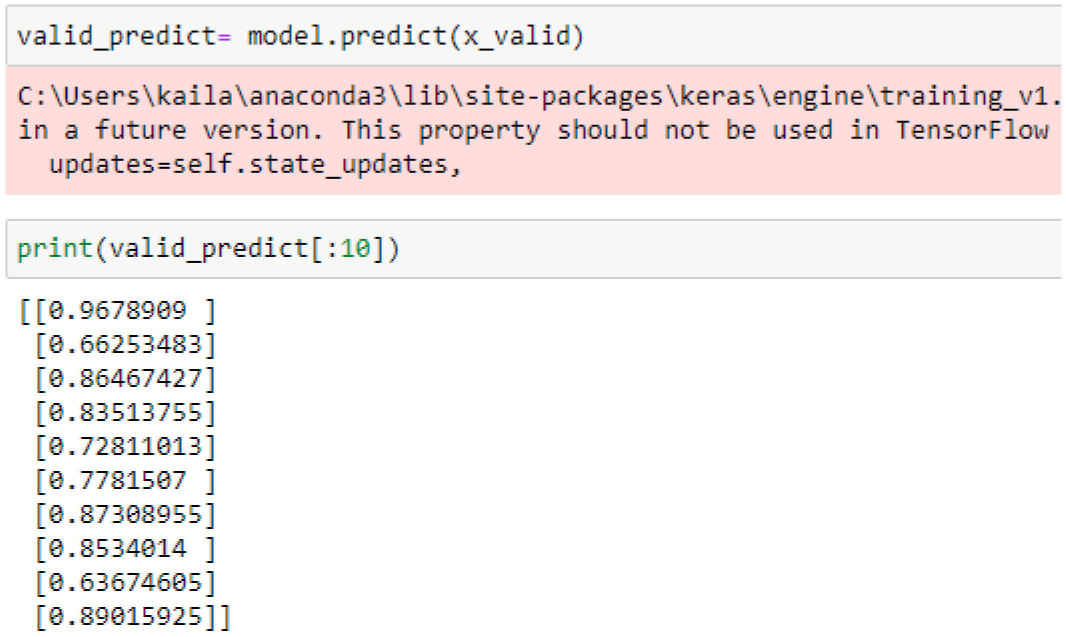
In this sentimental LSTM analysis, out model looks good fit. A good fit occurs when the model's performance is high on both the train and validation sets. the train and validation loss decreasing and stabilizing around the same point can be used to diagnose this

```
plt.plot(history.history['loss'], label=' training data')
plt.plot(history.history['val_loss'], label='validation data')
plt.title('Loss for Text Classification')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()
```



D4. Predictive Accuracy:

Once we save the model, we can predict on validation data the output is probabilities, if you see our predictions below



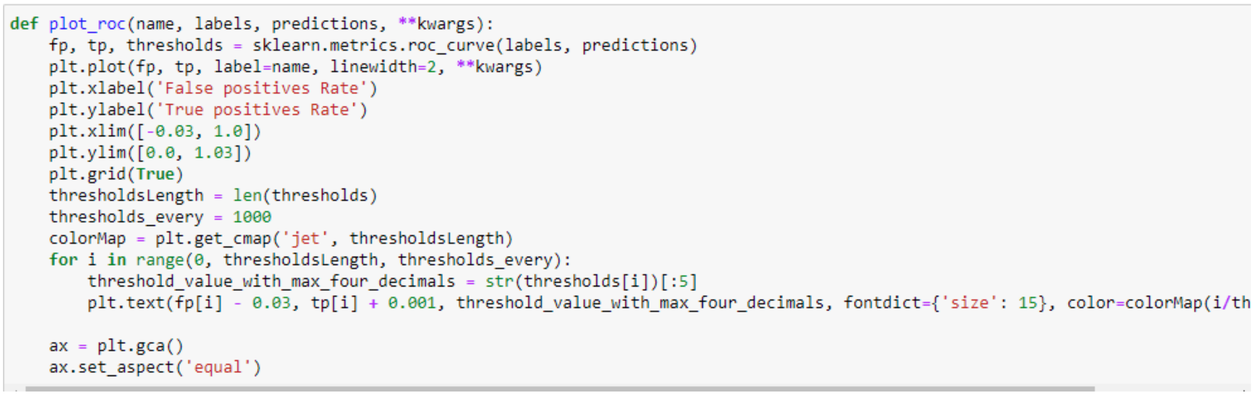
```
valid_predict= model.predict(x_valid)
```

```
C:\Users\kaila\anaconda3\lib\site-packages\keras\engine\training_v1.
in a future version. This property should not be used in TensorFlow
  updates=self.state_updates,
```

```
print(valid_predict[:10])
```

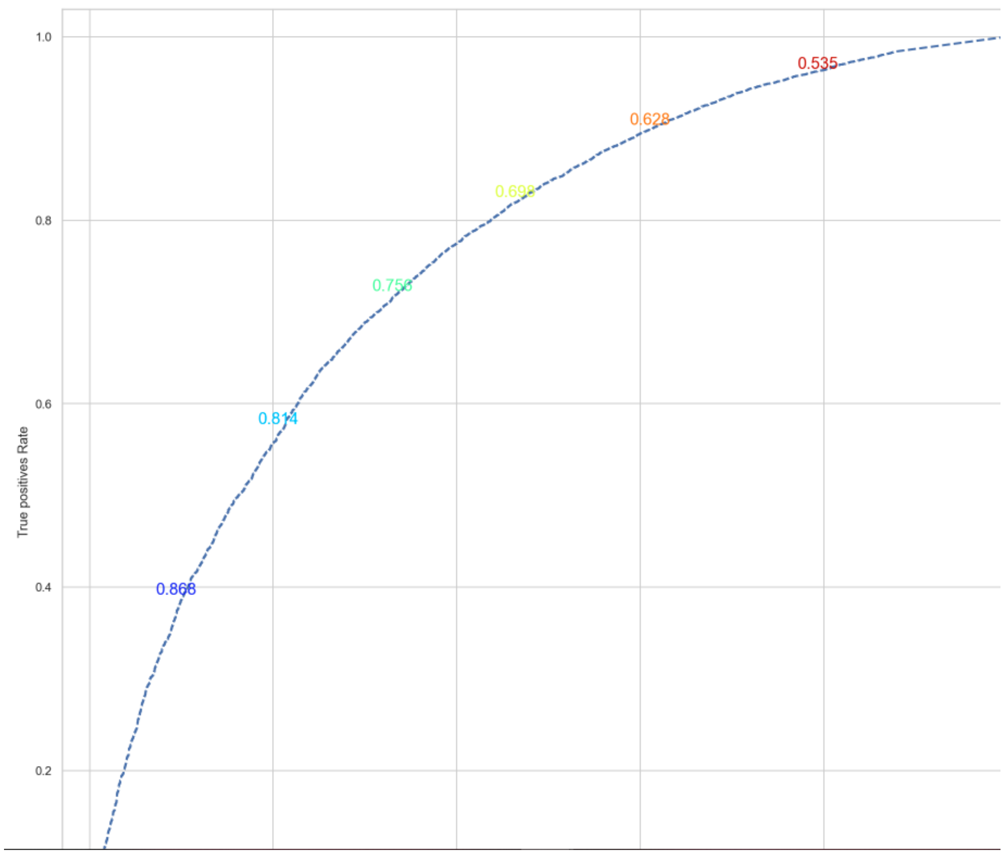
```
[[0.9678909 ]
 [0.66253483]
 [0.86467427]
 [0.83513755]
 [0.72811013]
 [0.7781507 ]
 [0.87308955]
 [0.8534014 ]
 [0.63674605]
 [0.89015925]]
```

We need to apply thresholds to classify Positive points and negative points, the positive sentiments, or negative sentiments. For that we used ROC curve. ROC curves illustrate the diagnostic ability of binary classifier system as its discrimination threshold is varied.



```
def plot_roc(name, labels, predictions, **kwargs):
    fp, tp, thresholds = sklearn.metrics.roc_curve(labels, predictions)
    plt.plot(fp, tp, label=name, linewidth=2, **kwargs)
    plt.xlabel('False positives Rate')
    plt.ylabel('True positives Rate')
    plt.xlim([-0.03, 1.0])
    plt.ylim([0.0, 1.03])
    plt.grid(True)
    thresholdsLength = len(thresholds)
    thresholds_every = 1000
    colorMap = plt.get_cmap('jet', thresholdsLength)
    for i in range(0, thresholdsLength, thresholds_every):
        threshold_value_with_max_four_decimals = str(thresholds[i])[:5]
        plt.text(fp[i] - 0.03, tp[i] + 0.001, threshold_value_with_max_four_decimals, fontdict={'size': 15}, color=colorMap(i/th

    ax = plt.gca()
    ax.set_aspect('equal')
```

```
<matplotlib.legend.Legend at 0x2b2af74e280>
```



In ROC curve you will see True Positive Rate and False Positive Rate

Reloading the saved model and generating Predictions

```
new_model = tf.keras.models.load_model('C:/Kailash/Rekha/D213/Task2/Beauty_5/savedTFLSTMModel/tf_lstmmodel.h5')
new_model.summary()
```

```
WARNING:tensorflow:From C:\Users\kaila\anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:93: calling GlorotUniform.
__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From C:\Users\kaila\anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:93: calling Orthogonal.__i
nit__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From C:\Users\kaila\anaconda3\lib\site-packages\tensorflow\python\ops\init_ops.py:93: calling Zeros.__init__
(from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           800016

 dropout (Dropout)           (None, 100, 16)           0

 lstm (LSTM)                 (None, 100, 16)           2112

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 512)               819712

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 8)                 4104

 dropout_2 (Dropout)         (None, 8)                 0

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 1,625,953
Trainable params: 1,625,953
Non-trainable params: 0
_________________________________________________________________
```

```
with open('C:/Kailash/Rekha/D213/Task2/Beauty_5/savedTFLSTMModel/tokenizer.json') as json_file:
    json_string = json.load(json_file)
tokenizer1 = tf.keras.preprocessing.text.tokenizer_from_json(json_string)
```

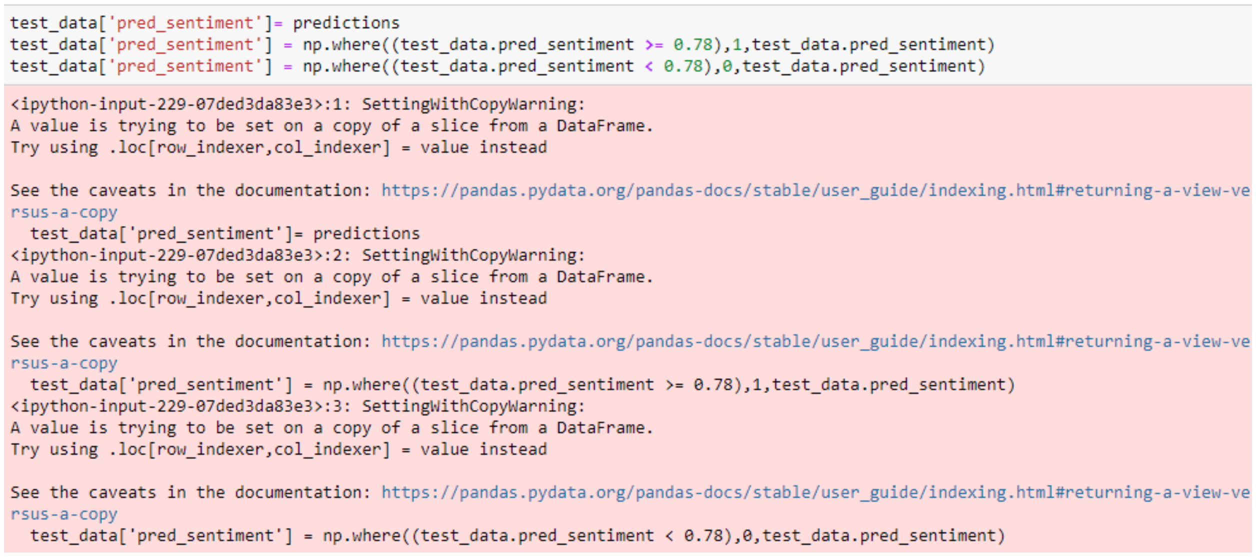
```
x_test  = np.array( tokenizer.texts_to_sequences(test_data['reviewText'].tolist()) )
x_test = pad_sequences(x_test, padding='post', maxlen=100)
```

```
<ipython-input-227-8bdeadb671a6>:1: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a lis
t-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must sp
ecify 'dtype=object' when creating the ndarray.
  x_test  = np.array( tokenizer.texts_to_sequences(test_data['reviewText'].tolist()) )
```

```
# Generate predictions (probabilities -- the output of the last layer)
# on test  data using `predict`
print("Generate predictions for all samples")
predictions = new_model.predict(x_test)
```

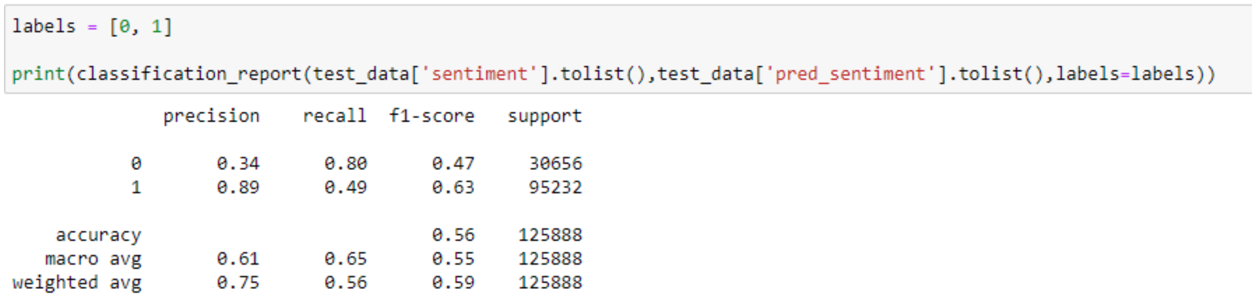
```
Generate predictions for all samples
```

```
C:\Users\kaila\anaconda3\lib\site-packages\keras\engine\training_v1.py:2067: UserWarning: `Model.state_updates` will be removed
in a future version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  updates=self.state_updates,
```



```
test_data['pred_sentiment']= predictions
test_data['pred_sentiment'] = np.where((test_data.pred_sentiment >= 0.78),1,test_data.pred_sentiment)
test_data['pred_sentiment'] = np.where((test_data.pred_sentiment < 0.78),0,test_data.pred_sentiment)
```

```
<ipython-input-229-07ded3da83e3>:1: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  test_data['pred_sentiment']= predictions
<ipython-input-229-07ded3da83e3>:2: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  test_data['pred_sentiment'] = np.where((test_data.pred_sentiment >= 0.78),1,test_data.pred_sentiment)
<ipython-input-229-07ded3da83e3>: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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  test_data['pred_sentiment'] = np.where((test_data.pred_sentiment < 0.78),0,test_data.pred_sentiment)
```



```
labels = [0, 1]

print(classification_report(test_data['sentiment'].tolist(),test_data['pred_sentiment'].tolist(),labels=labels))
```

```
              precision    recall  f1-score   support

           0       0.34      0.80      0.47     30656
           1       0.89      0.49      0.63     95232

    accuracy                           0.56    125888
   macro avg       0.61      0.65      0.55    125888
weighted avg       0.75      0.56      0.59    125888
```

If you see accuracy is 0.56 and F1 score is 0.47.

PART V. Conclusions

E. Neural Networks are being saved:

We are saving Neural network on below location

```
model.save('C:/Kailash/Rekha/D213/Task2/Beauty_5/savedTFLSTMModel/tf_lstmmodel.h5')
json_string = tokenizer.to_json()
```

```
import json
with open('C:/Kailash/Rekha/D213/Task2/Beauty_5/savedTFLSTMModel/tokenizer.json', 'w') as outfile:
    json.dump(json_string, outfile)
```

Attached the files on Submission.

F. Neural Networks and affects:

In this sentimental analysis, we used RNN network. A feedforward neural network with an internal memory is known as a recurrent neural network. RNN is recurrent in nature since it performs the same function for each data input, and the current input's outcome is dependent on the previous one's computation. The output is replicated and transmitted back into the recurrent network when it is created. It examines the current input as well as the output from the prior input when deciding. RNNs can model data sequences so that each sample is thought to be reliant on the ones before it. Even convolutional layers and recurrent neural networks are employed to extend the effective pixel neighborhood.

We used LSTM model for this analysis, Long-term memory (LSTM) networks are a modified version of recurrent neural networks that make it easier to recall past material. RNN's vanishing gradient problem is overcome in this paper. Given time lags of uncertain duration, LSTM is well suited to identify, analyses, and predict time series. Using backpropagation, it trains the model. Three gates are included in an LSTM network.

G. Suggestions:

According to the above classification report, the model had the simplest time classifying the sentiment of Bad and Excellent reviews but had a considerably more difficult difficulty classifying the sentiment of Fair and Good ratings. The accuracy of Fair and Good predictions was slightly higher than randomly selecting a class.

If we ran this model again, it would be interesting to see if changing the categories affected the model's accuracy. If we combined 4- and 5-star ratings together, we believe the algorithm would be reasonably effective in predicting good reviews. Also, combining Bad and Fair reviews together as negative would be the same. Another project concept would be to use sentiment analysis to forecast the actual star rating.

PART VI. Documentary Evidence

H. Notebook Code:

Task2-Sentimental Analysis.ipynb

I. Third Party Evidence:

Title: (Recurrent Neural Network), GeeksForGeeks.
URL: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
Date: July 4th, 2019

Title: SuperDataScience, (Machine Learning A-Z: Hands-On Python & R in Data Science)
Date: August 15th, 2021
URL: https://www.superdatascience.com/

Title: Sentimental Analysis in Python)
Author: Violeta Misheva.
URL: https://app.datacamp.com/learn/courses/sentiment-analysis-in-python
Date: 2021

Title: (Python code examples of)
Author: Y Zhang.
URL: https://zhang-yang.medium.com/deep-learning-model-for-product-category-prediction-40600747b22e
Date: 2018

J. Sources:

Title: (Machine Learning A-Z: Hands-On Python & R in Data Science), SuperDataScience
Date: August 15th, 2021
URL: https://www.superdatascience.com/