LSTM_on_Amazon_fine_food_review

December 6, 2020

1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [ ]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm_notebook
import os
```

In [ ]:
```
# using SQLite Table to read data.
con = sqlite3.connect('/content/drive/MyDrive/Colab Notebooks/database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 1
```

```
        # for tsne assignment you can take 5k data points

        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

        # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
        def partition(x):
            if x < 3:
                return 0
            return 1

        #changing reviews with score less than 3 to be positive and vice-versa
        actualScore = filtered_data['Score']
        positiveNegative = actualScore.map(partition)
        filtered_data['Score'] = positiveNegative
        print("Number of data points in our data", filtered_data.shape)
        filtered_data.head(3)
```

```
Number of data points in our data (50000, 10)
```

Out[ ]:

	Id	...	Text
0	1	...	I have bought several of the Vitality canned d...
1	2	...	Product arrived labeled as Jumbo Salted Peanut...
2	3	...	This is a confection that has been around a fe...

[3 rows x 10 columns]

```
In [ ]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [ ]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

Out[ ]:

	UserId	...	COUNT(*)
0	#oc-R115TNMSPFT9I7	...	2
1	#oc-R11D9D7SHXIJB9	...	3
2	#oc-R11DNU2NBKQ23Z	...	2
3	#oc-R11O5J5ZVQE25C	...	3
4	#oc-R12KPBODL2B5ZD	...	2

[5 rows x 7 columns]

```
In [ ]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[ ]:              UserId  ... COUNT(*)
       80638  AZY10LLTJ71NX  ...        5

       [1 rows x 7 columns]
```

```
In [ ]: display['COUNT(*)'].sum()
```

```
Out[ ]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

```
In [ ]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

```
Out[ ]:        Id  ...                                                Text
       0   78445  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
       1  138317  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
       2  138277  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
       3   73791  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
       4  155049  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...

       [5 rows x 10 columns]
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

```
In [ ]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

```
In [ ]: sorted_data.shape

Out[ ]: (50000, 10)

In [ ]: #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
        final.shape

Out[ ]: (46072, 10)

In [ ]: #Checking to see how much % of data still remains
        (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100

Out[ ]: 92.144
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

```
In [ ]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND Id=44737 OR Id=64422
        ORDER BY ProductID
        """, con)

        display.head()

Out[ ]:       Id  ...                                                Text
        0  64422  ...  My son loves spaghetti so I didn't hesitate or...
        1  44737  ...  It was almost a 'love at first bite' - the per...

        [2 rows x 10 columns]

In [ ]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]

In [ ]: #Before starting the next phase of preprocessing lets see the number of entries left
        print(final.shape)

        #How many positive and negative reviews are present in our dataset?
        final['Score'].value_counts()

(46071, 10)


Out[ ]: 1    38479
        0     7592
        Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

```
In [ ]: # printing some random reviews
        sent_0 = final['Text'].values[0]
        print(sent_0)
        print("="*50)

        sent_1000 = final['Text'].values[1000]
        print(sent_1000)
        print("="*50)

        sent_1500 = final['Text'].values[1500]
        print(sent_1500)
        print("="*50)

        sent_4900 = final['Text'].values[4900]
        print(sent_4900)
        print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
==================================================
```

```
In [ ]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
        sent_0 = re.sub(r"http\S+", "", sent_0)
        sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

In [ ]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
```

In [ ]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
```

```
            phrase = re.sub(r"\'re", " are", phrase)
            phrase = re.sub(r"\'s", " is", phrase)
            phrase = re.sub(r"\'d", " would", phrase)
            phrase = re.sub(r"\'ll", " will", phrase)
            phrase = re.sub(r"\'t", " not", phrase)
            phrase = re.sub(r"\'ve", " have", phrase)
            phrase = re.sub(r"\'m", " am", phrase)
            return phrase
```

In [ ]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
```

In [ ]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

In [ ]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually protein powders are high
```

In [ ]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
```

```
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                    'won', "won't", 'wouldn', "wouldn't"])
```

In [ ]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm_notebook(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
    preprocessed_reviews.append(sentance.strip())
```

```
HBox(children=(FloatProgress(value=0.0, max=46071.0), HTML(value='')))
```

In [ ]:
```
preprocessed_reviews[1500]
```

Out[ ]:
```
'great flavor low calories high nutrients high protein usually protein powders high pr
```

Applying LSTM

In [ ]:
```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

In [ ]:
```
import numpy as np
from keras.layers import LSTM
from keras.models import Sequential
from keras.layers.embeddings import Embedding
from keras.layers import Dense
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
np.random.seed(7) #every time you call the numpy's other random function
```

In [ ]:
```
x_i=preprocessed_reviews
y_i=np.array(final['Score'])
```

Splitting data into Train and Test

```
In [ ]: from sklearn.model_selection import train_test_split
        x_train,x_test,y_train,y_test=train_test_split(x_i,y_i,test_size=0.4)

In [ ]: print("Size of x_train=",len(x_train))
        print("Size of y_train=",len(y_train))
        print("Size of x_test=",len(x_test))
        print("Size of y_test=",len(y_test))
```

```
Size of x_train= 27642
Size of y_train= 27642
Size of x_test= 18429
Size of y_test= 18429
```

```
In [ ]: x_train[0]
```

```
Out[ ]: 'daughter loves one gas station marietta georgia sells chage lolipop'
```

One-Hot representaition

```
In [ ]: from keras.preprocessing.text import one_hot
        one_hot_train=[one_hot(word,5000)for word in x_train]
        one_hot_test=[one_hot(word,5000)for word in x_test]

In [ ]: print(one_hot_train[0])
        print(one_hot_test[0])
```

```
[363, 1535, 100, 3152, 48, 4888, 274, 2567, 1018, 4146]
[1205, 1954, 4581, 548, 2368, 218, 1831, 2178, 2019, 2860, 821, 4761, 1952]
```

Padding Data

```
In [ ]: max_review_length = 600
        one_hot_train = sequence.pad_sequences(one_hot_train, maxlen=max_review_length)
        one_hot_test = sequence.pad_sequences(one_hot_test, maxlen=max_review_length)

        print(one_hot_train.shape)
        print(one_hot_train[1],y_train[1])
```

```
(27642, 600)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 3834 1107  893 4946 4868 1502  821  199 1634 2234  800 2544 3574 4162
  800 4061 2394 3959 1028 2224  199 1390  410  897  846 4130] 1
```

4.2 LSTM with one layer

```
In [ ]: # create the model
        embedding_vecor_length = 32
        model = Sequential()
        model.add(Embedding(5000, embedding_vecor_length, input_length=max_review_length))
        model.add(LSTM(100))
        model.add(Dense(1, activation='sigmoid'))
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 600, 32)           160000
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:
```
from keras.optimizers import Adam
batch_size=64
epoch=10
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history=model.fit(one_hot_train, y_train,batch_size=batch_size,epochs=epoch,verbose=1,v
```

```
Epoch 1/10
432/432 [==============================] - 16s 38ms/step - loss: 0.3030 - accuracy: 0.8782 - va
Epoch 2/10
432/432 [==============================] - 16s 36ms/step - loss: 0.1996 - accuracy: 0.9224 - va
Epoch 3/10
432/432 [==============================] - 16s 36ms/step - loss: 0.1719 - accuracy: 0.9339 - va
Epoch 4/10
432/432 [==============================] - 16s 36ms/step - loss: 0.1492 - accuracy: 0.9428 - va
Epoch 5/10
432/432 [==============================] - 16s 36ms/step - loss: 0.1245 - accuracy: 0.9550 - va
Epoch 6/10
432/432 [==============================] - 16s 37ms/step - loss: 0.1070 - accuracy: 0.9610 - va
Epoch 7/10
432/432 [==============================] - 16s 36ms/step - loss: 0.0919 - accuracy: 0.9676 - va
Epoch 8/10
432/432 [==============================] - 16s 37ms/step - loss: 0.0874 - accuracy: 0.9693 - va
Epoch 9/10
432/432 [==============================] - 16s 37ms/step - loss: 0.0667 - accuracy: 0.9772 - va
Epoch 10/10
432/432 [==============================] - 16s 37ms/step - loss: 0.0593 - accuracy: 0.9798 - va
```

In [ ]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
%matplotlib inline
```

```
        import numpy as np

        def plt_dynamic(x, vy, ty, ax, colors=['b']):
            ax.plot(x, vy, 'b', label="Validation Loss")
            ax.plot(x, ty, 'r', label="Train Loss")
            plt.legend()
            plt.grid()
            fig.canvas.draw()
```

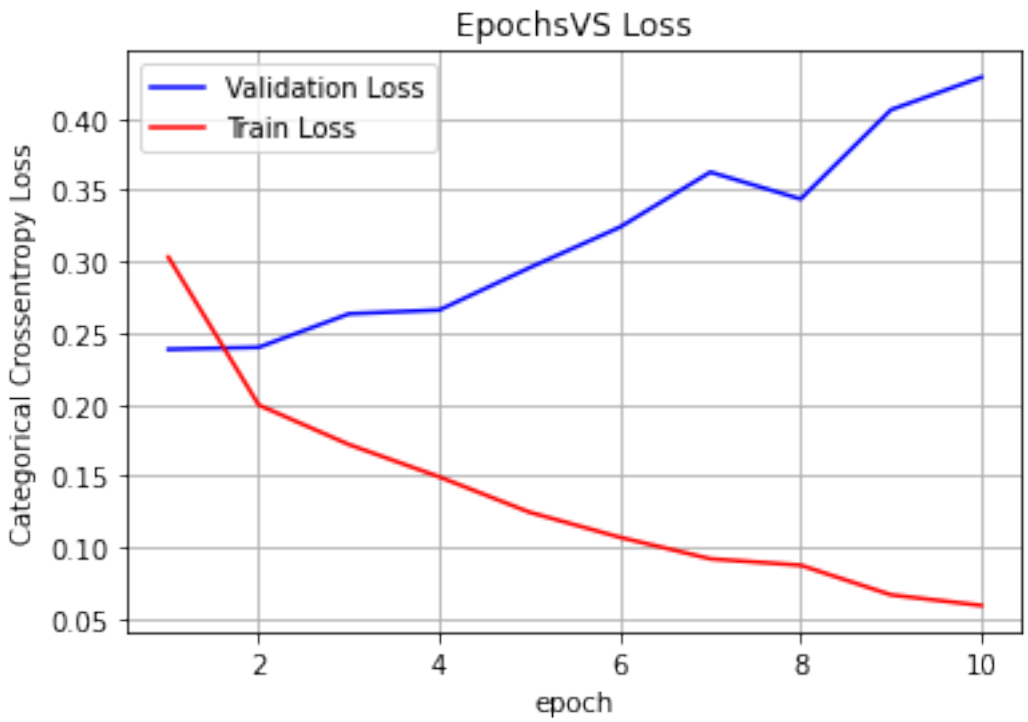
```
In [ ]: score = model.evaluate(one_hot_test, y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])


        fig,ax = plt.subplots(1,1)
        ax.set_title('EpochsVS Loss')
        ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

        # list of epoch numbers
        x = list(range(1,epoch+1))


        vy = history.history['val_loss']
        ty = history.history['loss']
        plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.42891067266464233
Test accuracy: 0.8754680156707764
```



4.3 LSTM with two layer

```
In [ ]: from keras.layers import Dropout
        embedding_vecor_length = 32
        model = Sequential()
        model.add(Embedding(5000, embedding_vecor_length, input_length=max_review_length))
        model.add(LSTM(100,return_sequences=True))
        model.add(LSTM(100))
        model.add(Dropout(0.5))
        model.add(Dense(1, activation='sigmoid'))
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 600, 32)           160000
_________________________________________________________________
lstm_3 (LSTM)                (None, 600, 100)          53200
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
```

```
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 293,701
Trainable params: 293,701
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [ ]: batch_size=64
        epoch=10
        model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
        history=model.fit(one_hot_train, y_train,batch_size=batch_size,epochs=epoch,verbose=1,v
```

```
Epoch 1/10
432/432 [==============================] - 31s 72ms/step - loss: 0.3013 - accuracy: 0.8820 - va
Epoch 2/10
432/432 [==============================] - 30s 69ms/step - loss: 0.2057 - accuracy: 0.9200 - va
Epoch 3/10
432/432 [==============================] - 30s 69ms/step - loss: 0.1801 - accuracy: 0.9306 - va
Epoch 4/10
432/432 [==============================] - 30s 69ms/step - loss: 0.1549 - accuracy: 0.9409 - va
Epoch 5/10
432/432 [==============================] - 30s 68ms/step - loss: 0.1307 - accuracy: 0.9522 - va
Epoch 6/10
432/432 [==============================] - 30s 69ms/step - loss: 0.1138 - accuracy: 0.9596 - va
Epoch 7/10
432/432 [==============================] - 30s 69ms/step - loss: 0.0896 - accuracy: 0.9685 - va
Epoch 8/10
432/432 [==============================] - 30s 69ms/step - loss: 0.0730 - accuracy: 0.9738 - va
Epoch 9/10
432/432 [==============================] - 30s 69ms/step - loss: 0.0550 - accuracy: 0.9815 - va
Epoch 10/10
432/432 [==============================] - 30s 69ms/step - loss: 0.0472 - accuracy: 0.9842 - va
```

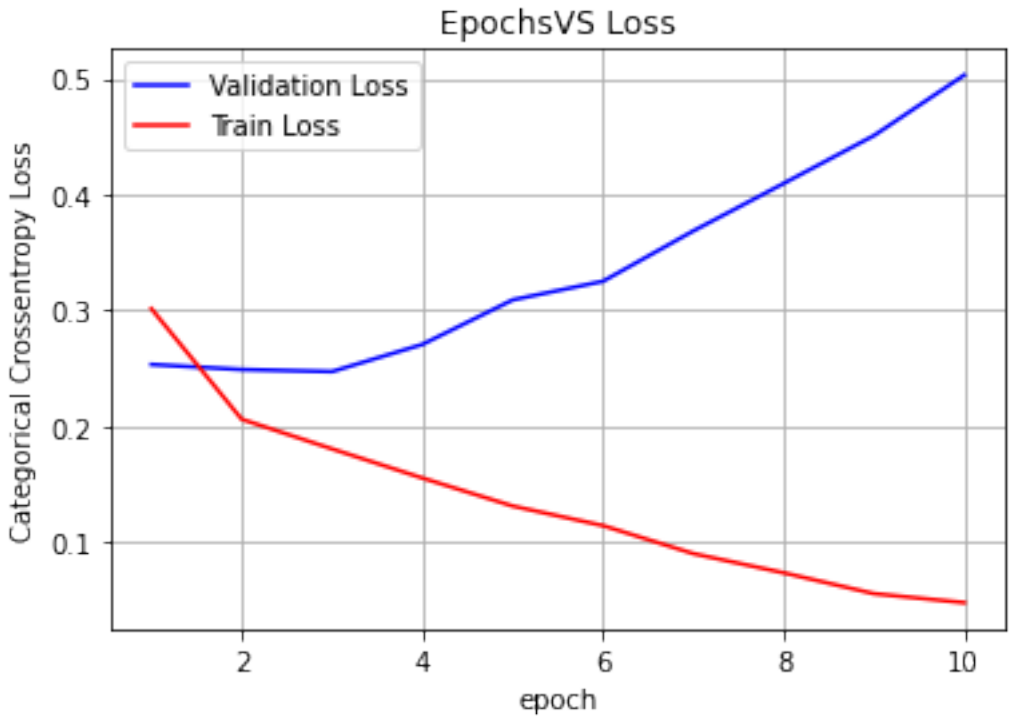
```
In [ ]: score = model.evaluate(one_hot_test, y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])


        fig,ax = plt.subplots(1,1)
        ax.set_title('EpochsVS Loss')
        ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

        # list of epoch numbers
        x = list(range(1,epoch+1))
```

```
        vy = history.history['val_loss']
        ty = history.history['loss']
        plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.5031416416168213
Test accuracy: 0.8881654143333435
```



5 Conclusion

```
In [ ]: from prettytable import PrettyTable
        x = PrettyTable()
        x.field_names = ["No of LSTM layers", "Accuracy %"]

        x.add_row(["1","87.54"])
        x.add_row(["2","88.81"])

        print(x)
```

```
+-------------------+------------+
| No of LSTM layers | Accuracy % |
```

1	87.54
2	88.81

Observed that double layer LSTM has given better accuracy than single layer LSTM