```
%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 nltk.stem.porter import PorterStemmer
from nltk.stem import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
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
from nltk.stem import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
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
import os
```

How to read csv file from drive i followed this link https://stackoverflow.com/questions/53619189/rea

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
import pandas as pd
#data = pd.read_csv("/content/drive/My Drive/Reviews.csv")
```

```
con = sqlite3.connect(r"/content/drive/My Drive/database.sqlite")
data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""",con)
```

```
print (data)
```

```
             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             4  ...  If you are looking for the secret ingredient i...
4             5  ...  Great taffy at a great price.  There was a wid...
...         ...  ...                                                ...
525809   568450  ...  Great for sesame chicken..this is a good if no...
525810   568451  ...  I'm disappointed with the flavor. The chocolat...
525811   568452  ...  These stars are small, so you can give 10-15 o...
525812   568453  ...  These are the BEST treats for training and rew...
525813   568454  ...  I am very satisfied ,product is as advertised,...

[525814 rows x 10 columns]
```

The below code i am copying from my own assignment (Assignment number 4 from naive bayes whe cleaning,data deplication etc processes)

```
# Change Score with 1 n 2 as -ve and 4 n 5 as +ve

def chng_to_0_or_1 (Score):
    if Score ==4 or Score ==5:
        return 1
    elif Score ==1 or Score ==2:
        return 0
    else:# Thus in case by some mistake any data is their with rating 6 or 7 etc due to sc
        pass
currentScore = data["Score"]
new_Score = currentScore.map(chng_to_0_or_1)
data["Score"] = new_Score
print ("Number of data points available")
print (data.shape)#Gives original number of data points available


#2 Data Cleaning a.) Getting rid of duplicates and b.) if helpnessdenominator < helpfulnes


data = data.drop_duplicates(subset = ["UserId","ProfileName","HelpfulnessNumerator","Helpf
print ("Number of data points after removing duplicates")
print (data.shape)#Gives data points are deduplication

# Reference: Copied from above cell  final=final[final.HelpfulnessNumerator<=final.Helpful
data=data[data.HelpfulnessNumerator<=data.HelpfulnessDenominator]
print ("Number of data points after removing where HelpfulnessNumerator is more than Helpf
print (data.shape)
```

```
#3 Preprocessing begins

#Convert to lower case,convert shortcut words to proper words, remove Special Character

#i) Convert to lower case:
data["Text"] =  (data["Text"].str.lower())
data["Summary"] =  (data["Summary"].str.lower())

#ii) Convert Shortcuts words to proper words
#List of Words are:https://en.wikipedia.org/wiki/Wikipedia:List_of_English_contractions
#Reference:https://stackoverflow.com/questions/39602824/pandas-replace-string-with-another
data['Text'] = data['Text'].replace({"ain't":"am not","amn't":"am not","aren't":"are not",
"can't":"cannot","cause":"because","could've":"could have","couldn't":"could not","couldn'
"daren't":"dare not","daresn't":"dare not","dasn't":"dare not","didn't":"did not","doesn't
"don't":"do not","e'er":"ever","everyone's":"everyone is","finna":"fixing to","gimme":"giv
"gonna":"going to","gon't":"go not","gotta":"got to","hadn't":"had not","hasn't":"has not'
"he'd":"he had","he'll":"he shall","he's":"he has","he've":"he have","how'd":"how did","ho
"how're":"how are","how's":"how has","I'd":"I had","I'll":"I shall","I'm":"I am","I'm'a":'
"I'm'o":"I am going to","I've":"I have","isn't":"is not","it'd":"it would","it'll":"it sha
"let's":"let us","mayn't":"may not","may've":"may have","mightn't":"might not","might've":
"mustn't":"must not","mustn't've":"must not have","must've":"must have","needn't":"need no
"o'clock":"of the clock","o'er":"","ol'":"old","oughtn't":"ought not","shalln't":"shall no
"she'd":"she had","she'll":"she shall","she's":"she is","should've":"should have","shouldn
"shouldn't've":"should not have","somebody's":"somebody has","someone's":"someone has","so
"that'll":"that will","that're":"that are","that's":"that is","that'd":"that would","there
"there'll":"there shall","there're":"there are","there's":"there is","these're":"hese are'
"they'll":"they will","they're":"they are","they've":"they have","this's":"","those're":"t
"twas":"it was","wasn't":"was not","we'd":"we had","we'd've":"we would have","we'll":"we w
"we've":"we have","weren't":"were not","what'd":"what did","what'll":"what will","what're'
"what've":"what have","when's":"when is","where'd":"where did","where're":"where are","whe
"which's":"which has","who'd":"who would","who'd've":"who would have","who'll":"who shall'
"who's":"who has","who've":"who have","why'd":"why did","why're":"why are","why's":"why ha
"would've":"would have","wouldn't":"would not","y'all":"you all","you'd":"you had","you'll
"you've":"you have"})
##############Lets do the same for summary Text#####################################
data['Summary'] = data['Summary'].replace({"ain't":"am not","amn't":"am not","aren't":"are
"can't":"cannot","cause":"because","could've":"could have","couldn't":"could not","couldn'
"daren't":"dare not","daresn't":"dare not","dasn't":"dare not","didn't":"did not","doesn't
"don't":"do not","e'er":"ever","everyone's":"everyone is","finna":"fixing to","gimme":"giv
"gonna":"going to","gon't":"go not","gotta":"got to","hadn't":"had not","hasn't":"has not'
"he'd":"he had","he'll":"he shall","he's":"he has","he've":"he have","how'd":"how did","ho
"how're":"how are","how's":"how has","I'd":"I had","I'll":"I shall","I'm":"I am","I'm'a":'
"I'm'o":"I am going to","I've":"I have","isn't":"is not","it'd":"it would","it'll":"it sha
"let's":"let us","mayn't":"may not","may've":"may have","mightn't":"might not","might've":
"mustn't":"must not","mustn't've":"must not have","must've":"must have","needn't":"need no
"o'clock":"of the clock","o'er":"","ol'":"old","oughtn't":"ought not","shalln't":"shall no
"she'd":"she had","she'll":"she shall","she's":"she is","should've":"should have","shouldn
"shouldn't've":"should not have","somebody's":"somebody has","someone's":"someone has","so
"that'll":"that will","that're":"that are","that's":"that is","that'd":"that would","there
"there'll":"there shall","there're":"there are","there's":"there is","these're":"hese are'
```

```
 they'll : they will , they're : they are , they've : they have , this's :  , those're : t
"twas":"it was","wasn't":"was not","we'd":"we had","we'd've":"we would have","we'll":"we w
"we've":"we have","weren't":"were not","what'd":"what did","what'll":"what will","what're'
"what've":"what have","when's":"when is","where'd":"where did","where're":"where are","whe
"which's":"which has","who'd":"who would","who'd've":"who would have","who'll":"who shall'
"who's":"who has","who've":"who have","why'd":"why did","why're":"why are","why's":"why ha
"would've":"would have","wouldn't":"would not","y'all":"you all","you'd":"you had","you'll
"you've":"you have"})
#################################################################################
# iii) Remove Special Characters except alpahbets and numbers
#The reason i dont want to remove number people might write got five eggs as 5 eggs or vic
#that information which could be useful
#Ref:https://stackoverflow.com/questions/33257344/how-to-remove-special-characers-from-a-c
data["Text"]=data["Text"].map(lambda x: re.sub(r'[^a-zA-Z ]', '', x))
data["Summary_copy"]=data["Summary"].map(lambda x: re.sub(r'[^a-zA-Z ]', '', x))
import nltk
nltk.download("popular")

#The Summary are usually so small if we remove few stopwords the meaning itself would be c
# So let us see what all stopwords we have
#Ref:::::::::https://stackoverflow.com/questions/5511708/adding-words-to-nltk-stoplist
#https://chrisalbon.com/machine_learning/preprocessing_text/remove_stop_words/

stopwords = nltk.corpus.stopwords.words('english')
newStopWords = ['would','could','br','<br>','<','>','-','---br', '-----br','------br','---
#newStopWords = ['would','could','br','<br>','<','>','-','---br', '-----br','------br','--
notstopwords = ['not','no','nor']
stopwords.extend(newStopWords)
stopwords = [word for word in stopwords if word not in notstopwords]

#  iv) For now let us just go with flow will use default stopwords as creating our own stc
#Rather will use n-gram stratergy to get rid of problem of stopwords removal changing the
#Ref:https://stackoverflow.com/questions/43184364/python-remove-stop-words-from-pandas-dat
data["New_Text"]= data['Text'].apply(lambda x: [item for item in str.split(x) if item not
data["Summary"]= data['Summary'].apply(lambda x: [item for item in str.split(x) if item nc

#Ref:https://stackoverflow.com/questions/37347725/converting-a-panda-df-list-into-a-string
#we are creating new column New_summary so in case in future we need summary it is intact
data["New_Text"]=data["New_Text"].apply(' '.join)
data["Summary"]=data["Summary"].apply(' '.join)

# v) Now lets do Stemming
#https://stackoverflow.com/questions/48617589/beginner-stemming-in-pandas-produces-letters
english_stemmer=SnowballStemmer('english', ignore_stopwords=True)
data["New_Text"] = data["New_Text"].apply(english_stemmer.stem)
data["Summary"] = data["Summary"].apply(english_stemmer.stem)
data["New_Text"] = data["New_Text"].astype(str)
data["Summary"] = data["Summary"].astype(str)

#vi) stemming without removing stop words
english_stemmer=SnowballStemmer('english', ignore_stopwords=True)
#https://stackoverflow.com/questions/34724246/attributeerror-float-object-has-no-attribute
```

```
data["Text_with_stop"]=data["Text"].astype(str)
data["Summary"]=data["Summary"].astype(str)
data["Text_with_stop"]=data["Text_with_stop"].str.lower().map(english_stemmer.stem)
data["Summary"]=data["Summary"].str.lower().map(english_stemmer.stem)
data["Text_with_stop"]=data["Text_with_stop"].apply(''.join)
data["Summary"]=data["Summary"].apply(''.join)
data["Text_with_stop"] = data["Text_with_stop"].astype(str)
data["Summary"] = data["Summary"].astype(str)
################
# New_text_without_stop_convert as string
data["New_Text"] = data["New_Text"].astype(str)
data["New_Text"]=data["New_Text"].apply(''.join)
data["Summary"]=data["Summary"].apply(''.join)
data["Summary"] = data["Summary"].astype(str)
data["New_Text"] = data["New_Text"].astype(str)
print(data["Score"].value_counts())
print ("Thus we see there are 85% and 15% positive and negative reviews,thus a unbalanced
dataset we first copy negative dataset 6 times than we sample with same number of times as
# Let include another feature which is the length of the text
data_neg = data[data["Score"] == 0]
data_pos = data[data["Score"] == 1]
data = pd.concat([data_pos,data_neg])
#https://stackoverflow.com/questions/46429033/how-do-i-count-the-total-number-of-words-in-
data["Text_length"]= (data["New_Text"].str.count(' ') + 1)
data["Summary_length"]= (data["Summary"].str.count(' ') + 1)
data["Time_formatted"]= pd.to_datetime(data["Time"])
data.sort_values(by=['Time_formatted'], inplace=True)
```

```
Number of data points available
(525814, 10)
Number of data points after removing duplicates
(366392, 10)
Number of data points after removing where HelpfulnessNumerator is more than Helpfulr
(366390, 10)
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package gazetteers to /root/nltk_data...
[nltk_data]    |   Package gazetteers is already up-to-date!
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package shakespeare to /root/nltk_data...
[nltk_data]    |   Package shakespeare is already up-to-date!
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package twitter_samples is already up-to-date!
[nltk_data]    | Downloading package omw to /root/nltk_data...
[nltk_data]    |   Package omw is already up-to-date!
[nltk_data]    | Downloading package wordnet to /root/nltk_data...
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to /root/nltk_data...
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package punkt to /root/nltk_data...
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package snowball_data is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection popular
1    308679
0     57711
Name: Score, dtype: int64
```

Thus we see there are 85% and 15% positive and negative reviews,thus a unbalanced dat

```
print (data)
```

```
             Id   ProductId  ... Summary_length                Time_formatted
138706  150524  0006641040  ...              3 1970-01-01 00:00:00.939340800
138683  150501  0006641040  ...              7 1970-01-01 00:00:00.940809600
417839  451856  B00004CXX9  ...              2 1970-01-01 00:00:00.944092800
212472  230285  B00004RYGX  ...              4 1970-01-01 00:00:00.944438400
417838  451855  B00004CXX9  ...              1 1970-01-01 00:00:00.946857600
...        ...         ...  ...            ...                         ...
478157  517064  B007OYUZIM  ...              1 1970-01-01 00:00:01.351209600
457460  494625  B0012VSXIM  ...              5 1970-01-01 00:00:01.351209600
42269    45991  B007VQQT1K  ...              3 1970-01-01 00:00:01.351209600
15069    16426  B007TJGZ54  ...              2 1970-01-01 00:00:01.351209600
479033  518008  B00529PFX6  ...              1 1970-01-01 00:00:01.351209600

[366390 rows x 16 columns]
```

So we see we have 16 rows few of them are copy. The eason we made copy so in case we need to use corrupted due to some of our mistakes. But for now everything looks good. Lets make another datafra and now we can operate on this new dataframe

Also as number of rows is almost 3.5 lakhs which is taking us too long to compute lets us just take la

```
#data.sort_values(by=['Time_formatted'], inplace=True)
data = data.tail(150000)
print (data)
```

```
             Id   ProductId  ... Summary_length                Time_formatted
57121    61959  B000CQG8K8  ...              3 1970-01-01 00:00:01.320451200
503701  544672  B001P3PR54  ...              3 1970-01-01 00:00:01.320451200
144439  156703  B002L35LH6  ...              3 1970-01-01 00:00:01.320451200
100504  109185  B003SE8CN2  ...              5 1970-01-01 00:00:01.320451200
128467  139421  B000EM8308  ...              1 1970-01-01 00:00:01.320451200
...        ...         ...  ...            ...                         ...
478157  517064  B007OYUZIM  ...              1 1970-01-01 00:00:01.351209600
457460  494625  B0012VSXIM  ...              5 1970-01-01 00:00:01.351209600
42269    45991  B007VQQT1K  ...              3 1970-01-01 00:00:01.351209600
15069    16426  B007TJGZ54  ...              2 1970-01-01 00:00:01.351209600
479033  518008  B00529PFX6  ...              1 1970-01-01 00:00:01.351209600

[150000 rows x 16 columns]
```

```
#data.sort_values(by=['Time_formatted'], inplace=True)
data_text= data['New_Text']
#data.sort_values(by=['Time_formatted'], inplace=True)
data_summary= data['Summary_copy']
#data.sort_values(by=['Time_formatted'], inplace=True)
new_data_x = data[['Id','ProductId','UserId','ProfileName','HelpfulnessNumerator','Helpfu
new_data_y = data[['Score']]
data_text_summary = pd.concat([data_summary, data_text], axis=1, ignore_index=True)
```

```
data_text_summary = pd.concat([data_summary, data_text], axis=1, ignore_index=True)
data_text_summary.columns = ["Summary", "Text"]
print ("############################")
print ((data_text_summary))
merged_data_text_summary = pd.DataFrame()
merged_data_text_summary['Summary_Text'] = data_text_summary[['Summary', 'Text']].apply(la
print ("############################")
print ((merged_data_text_summary))
#X= new_data_x.values
```

```
############################
                                Summary
57121            wonderful aromatic tea  tea wonderful treat cold night cinnamon arom
503701          great treat for my dogs  one best products market cleaning dogs teeth
144439                yummy lil things  not potato chip tortia chip seaweed paper fc
100504  its a hit   stopped smelly poop  product definitely hit two finicky cats one
128467                             yuck  bought pregnant terrible morning sickness th
...                                 ...
478157                         just bad  watery unpleasant like yoohoo mixed dirty di
457460      good  plenty licorice candy  like licorice love candy remember eating car
42269                 great irish tea  recently returned wonderful three week excur
15069                    super coffee  great coffee easy brew coffee great aroma gc
479033                      expensive  expensive tough salty tried piece not care m

[150000 rows x 2 columns]
############################
                                                Summary_Text
57121   wonderful aromatic tea tea wonderful treat col...
503701  great treat for my dogs one best products mark...
144439  yummy lil things not potato chip tortia chip s...
100504  its a hit   stopped smelly poop product defini...
128467  yuck bought pregnant terrible morning sickness...
...                                                   ...
478157  just bad watery unpleasant like yoohoo mixed d...
457460  good  plenty licorice candy like licorice love...
42269   great irish tea recently returned wonderful th...
15069   super coffee great coffee easy brew coffee gre...
479033  expensive expensive tough salty tried piece no...

[150000 rows x 1 columns]
```

```
data_text_summary_train,data_text_summary_test = train_test_split(merged_data_text_summary
print (data_text_summary_train)
```

```
                                                  Summary_Text
57121   wonderful aromatic tea tea wonderful treat col...
503701  great treat for my dogs one best products mark...
144439  yummy lil things not potato chip tortia chip s...
100504  its a hit   stopped smelly poop product defini...
128467  yuck bought pregnant terrible morning sickness...
...                                                   ...
88548   fantastic buy exact product large oriental mar...
15067   coffee flavored water excited see pack coffee ...
110968  too sweet and too artificial not impressed pro...
15033   very bland and weak coffee admit like strong f...
425885  sweet with a kick candies sweet chewy mean kic...

[100500 rows x 1 columns]
```

```
Y = new_data_y['Score'].values

y_train,y_test = train_test_split(Y, test_size=0.33, shuffle=False)
print ((y_train).shape)
print ((y_test).shape)
```

```
(100500,)
(49500,)
```

```
data_text_summary_train_val = data_text_summary_train.values
data_text_summary_test_val = data_text_summary_test.values


print (type(data_text_summary_train_val))
print ("Shape of data_text_summary_train is::")

print (data_text_summary_train_val.shape)
print ("First few row of data_text_summary_train looks like::")

print (data_text_summary_train_val)
print ("############################################################")
print (type(data_text_summary_train_val.ravel()))
print ("############################################################")
print (data_text_summary_train_val.ravel().shape)
print ("############################################################")
print (type(data_text_summary_train['Summary_Text'].astype(str).values.shape))
print ("############################################################")
#print (type(data_text_summary_train_val_xml))
print ("############################################################")
#print ((data_text_summary_train_val_xml))
```

```
<class 'numpy.ndarray'>
Shape of data_text_summary_train is::
(100500, 1)
First few row of data_text_summary_train looks like::
[['wonderful aromatic tea tea wonderful treat cold night cinnamon aroma enticing buyi
 ['great treat for my dogs one best products market cleaning dogs teeth opinion like
 ['yummy lil things not potato chip tortia chip seaweed paper formbr please dont crea
 ...
 ['too sweet and too artificial not impressed product fact sitting pantry might possi
 ['very bland and weak coffee admit like strong flavorful not burnt tasting coffee gi
 ['sweet with a kick candies sweet chewy mean kick towards end think helped little mc
############################################################
<class 'numpy.ndarray'>
############################################################
(100500,)
############################################################
<class 'tuple'>
############################################################
############################################################
```

https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

```
from keras.preprocessing.text import Tokenizer
# create the tokenizer
t1 = Tokenizer(num_words=5000, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n', lower=True,
t1.fit_on_texts(data_text_summary_train['Summary_Text'].astype(str).values)

# summarize what was learned
t1_word_count = (t1.word_counts)
t1_document_count = (t1.document_count)
t1_word_index = (t1.word_index)
t1_word_docs=(t1.word_docs)

text_to_seq_train = t1.texts_to_sequences(data_text_summary_train['Summary_Text'].astype(s
text_to_seq_test = t1.texts_to_sequences(data_text_summary_test['Summary_Text'].astype(str



print (list(t1_word_count.items())[:100])
print ((t1_document_count))
print (list(t1_word_index.items())[:150])
print (list(t1_word_docs.items())[:500])
print ("##################")
print ((text_to_seq_train[:1]))
print (text_to_seq_test[:1])
```

```
[('wonderful', 5025), ('aromatic', 210), ('tea', 24812), ('treat', 5140), ('cold', 23
100500
[('not', 1), ('like', 2), ('good', 3), ('great', 4), ('taste', 5), ('product', 6), ('
[('night', 1398), ('always', 5901), ('aroma', 1417), ('wonderful', 4525), ('enticing'
##################
[[150, 2582, 11, 11, 150, 143, 363, 540, 323, 532, 157, 98, 25, 93, 16, 745]]
[[548, 1035, 498, 8, 67, 8, 310, 68, 296, 2683, 553, 278, 1425, 4609, 579, 3542, 278,
```

Lets see what is the average length of sentences.

```
print(sum(map(len, text_to_seq_train[:245481])))
print ("Average length of sentences are:")
print (4025586/100500)
print ("################")
```

```
4076411
Average length of sentences are:
40.05558208955224
################
```

Also lets see sentence with highset number of words

```
def FindMaxLength(lst):
    maxList = max((x) for x in lst)
    maxLength = max(len(x) for x in lst )

    return maxList, maxLength

print(FindMaxLength(text_to_seq_train))
```

```
([4997, 674, 3596, 4997, 3596, 693, 1037, 1193, 3523, 2236, 22, 1, 487, 450, 162, 136
```

```
max_review_length = 300
X_train = sequence.pad_sequences(text_to_seq_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(text_to_seq_test, maxlen=max_review_length)

print(X_train.shape)
print(X_train[1])
```

```
(100500, 300)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    4  143
   84  168   96    7   16  134  396 2006   96  544  672    2   74  143
   60  480  544  552  405  374]
```

Lets Try Single Layer LSTM first

```
import warnings

with warnings.catch_warnings():
    warnings.filterwarnings("ignore",category=DeprecationWarning)

epochs = 10
batch_size = 512

from keras.layers import Dense, Embedding, LSTM, Dropout
from keras.layers.normalization import BatchNormalization


embed_vector_length = 32
model1 = Sequential()
model1.add(Embedding(5000, embed_vector_length, input_length=max_review_length,embeddings_
model1.add(Dropout(0.3))
model1.add(LSTM(100))
model1.add(BatchNormalization())
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print("Printing the Model Summary")
print(model1.summary())
```

```
Printing the Model Summary
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 300, 32)           160000
_________________________________________________________________
dropout_6 (Dropout)          (None, 300, 32)           0
_________________________________________________________________
lstm_6 (LSTM)                (None, 100)               53200
_________________________________________________________________
batch_normalization_4 (Batch (None, 100)               400
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,701
Trainable params: 213,501
Non-trainable params: 200
_________________________________________________________________
None
```

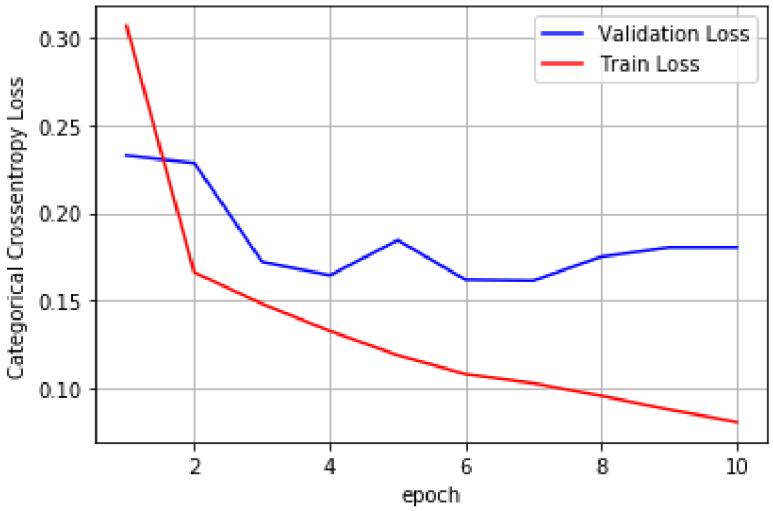
```
from keras.callbacks import EarlyStopping

callbacks = [EarlyStopping(monitor='val_loss', patience=3)]

import warnings

with warnings.catch_warnings():
    warnings.filterwarnings("ignore",category=DeprecationWarning)

history1=model1.fit(X_train, y_train,batch_size=batch_size,epochs=epochs,callbacks=callbac
```

```
Train on 100500 samples, validate on 49500 samples
Epoch 1/10
100500/100500 [==============================] - 104s 1ms/step - loss: 0.3067 - acc:
Epoch 2/10
100500/100500 [==============================] - 103s 1ms/step - loss: 0.1660 - acc:
Epoch 3/10
100500/100500 [==============================] - 104s 1ms/step - loss: 0.1481 - acc:
Epoch 4/10
100500/100500 [==============================] - 103s 1ms/step - loss: 0.1326 - acc:
Epoch 5/10
100500/100500 [==============================] - 104s 1ms/step - loss: 0.1188 - acc:
Epoch 6/10
100500/100500 [==============================] - 100s 995us/step - loss: 0.1080 - acc
Epoch 7/10
100500/100500 [==============================] - 98s 975us/step - loss: 0.1028 - acc:
Epoch 8/10
100500/100500 [==============================] - 98s 976us/step - loss: 0.0956 - acc:
Epoch 9/10
100500/100500 [==============================] - 97s 970us/step - loss: 0.0878 - acc:
Epoch 10/10
100500/100500 [==============================] - 98s 974us/step - loss: 0.0806 - acc:
```

```
def plt_dynamic(x, vy, ty, ax, colors=['b']):
```

```
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()


x = list(range(1,epochs+1))

%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt

scores1 = model1.evaluate(X_test, y_test, verbose=0)



print('Test score:', scores1[0])
print('Test accuracy:', scores1[1])

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

# list of epoch numbers
vy1 = history1.history['val_loss']
ty1 = history1.history['loss']
plt_dynamic(x, vy1, ty1, ax)
```

```
Test score: 0.1802538271431971
Test accuracy: 0.9388686868542373
```



Now lets dig in 2 layer LSTM model

```
import warnings

with warnings.catch_warnings():
    warnings.filterwarnings("ignore",category=DeprecationWarning)
```

```
epochs = 10
batch_size = 512

from keras.layers import Dense, Embedding, LSTM, Dropout
from keras.layers.normalization import BatchNormalization


embed_vector_length = 32
model2 = Sequential()
model2.add(Embedding(5000, embed_vector_length, input_length=max_review_length,embeddings_
model2.add(Dropout(0.3))
model2.add(LSTM(100, return_sequences=True))
model2.add(Dropout(0.3))
model2.add(LSTM(100))
model2.add(BatchNormalization())
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print("Printing the Model Summary")
print(model2.summary())
```

```
Printing the Model Summary
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 300, 32)           160000
_________________________________________________________________
dropout_3 (Dropout)          (None, 300, 32)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 300, 100)          53200
_________________________________________________________________
dropout_4 (Dropout)          (None, 300, 100)          0
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               80400
_________________________________________________________________
batch_normalization_2 (Batch (None, 100)               400
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 294,101
Trainable params: 293,901
Non-trainable params: 200
_________________________________________________________________
None
```

https://chrisalbon.com/deep_learning/keras/neural_network_early_stopping/

```
from keras.callbacks import EarlyStopping

callbacks = [EarlyStopping(monitor='val_loss', patience=3)]

import warnings
```

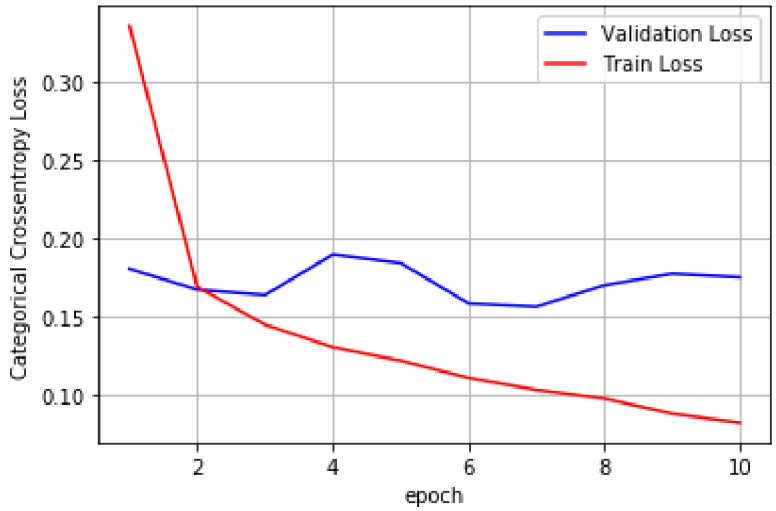
```
import warnings

with warnings.catch_warnings():
    warnings.filterwarnings("ignore",category=DeprecationWarning)

history2=model2.fit(X_train, y_train,batch_size=batch_size,epochs=epochs,callbacks=callbac
```

```
Train on 100500 samples, validate on 49500 samples
Epoch 1/10
100500/100500 [==============================] - 197s 2ms/step - loss: 0.3352 - acc:
Epoch 2/10
100500/100500 [==============================] - 195s 2ms/step - loss: 0.1688 - acc:
Epoch 3/10
100500/100500 [==============================] - 197s 2ms/step - loss: 0.1446 - acc:
Epoch 4/10
100500/100500 [==============================] - 200s 2ms/step - loss: 0.1303 - acc:
Epoch 5/10
100500/100500 [==============================] - 203s 2ms/step - loss: 0.1217 - acc:
Epoch 6/10
100500/100500 [==============================] - 204s 2ms/step - loss: 0.1108 - acc:
Epoch 7/10
100500/100500 [==============================] - 206s 2ms/step - loss: 0.1032 - acc:
Epoch 8/10
100500/100500 [==============================] - 206s 2ms/step - loss: 0.0976 - acc:
Epoch 9/10
100500/100500 [==============================] - 206s 2ms/step - loss: 0.0880 - acc:
Epoch 10/10
100500/100500 [==============================] - 206s 2ms/step - loss: 0.0822 - acc:
```

```
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()


x = list(range(1,epochs+1))

%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt

scores2 = model2.evaluate(X_test, y_test, verbose=0)



print('Test score:', scores2[0])
print('Test accuracy:', scores2[1])

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

# list of epoch numbers
vy2 = history2.history['val_loss']
```

```
ty2 = history2.history['loss']
plt_dynamic(x, vy2, ty2, ax)
```

```
Test score: 0.1749505036221911
Test accuracy: 0.9403434343289847
```



Conclusion:

Both the model single and double layer LSTM performed very well. Both being around 95% accurate