OBJECTIVE :- Built LSTM model on Amazon Fine Food Reviews

```
# Installing package
!pip install gensim

# Importing libraries
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem.porter import PorterStemmer
import 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
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.6/dist-packages (3.
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: boto>=2.32 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: bz2file in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/di
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/di
Requirement already satisfied: botocore<1.13.0,>=1.12.130 in /usr/local/lib/python3
Requirement already satisfied: s3transfer<0.3.0,>=0.2.0 in /usr/local/lib/python3.6
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/python3.6/d
Requirement already satisfied: docutils>=0.10 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7"
paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install parami
```

Loading Data from Reviews.CSV file in Google Drive

```
# Install the PyDrive wrapper & import libraries.
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
```

```
from oauth2client.client import GoogleCredentials
```

```
     100% |████████████████████████████████| 993kB 25.1MB/s
  Building wheel for PyDrive (setup.py) ... done
```

```
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
# Downloading a file based on its file ID
#file_id = 'REPLACE_WITH_YOUR_FILE_ID'
#downloaded = drive.CreateFile({'id': file_id})
downloaded = drive.CreateFile({'id':'1dTt4BADLZcTEBxRX5lMjAgwYosP5iepz'})
downloaded.GetContentFile('Reviews.CSV')

# Getting data into a dataframe
df = pd.read_csv('Reviews.CSV')
```

```
df.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

```
# Eliminating neutral reviews i.e. those reviews with Score = 3
filtered_data = df[df['Score'] != 3]

# Converting Score variables to binary class variable (1-positive review and 0-negative re
# Give reviews with Score>3 a positive rating (1) , and reviews with a score<3 a negative
def polarity(x):
    if x < 3:
        return 0
    return 1

# Applying polarity function on Score column of filtered_data
filtered_data['Score'] = filtered_data['Score'].map(polarity)

print(filtered_data.shape)
filtered_data.head()
```

```
(525814, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuli
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	

NOTE :- In above table , in Score column 1 = positive review and 0 = negative review

Data Cleaning: Deduplication

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False,

#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='fir

# Removing rows where HelpfulnessNumerator is greater than HelpfulnessDenominator
final = final[final.HelpfulnessNumerator <= final.HelpfulnessDenominator]

print(final.shape)
final[30:50]
```

150499	150500	0006641040	A1IJKK6Q1GTEAY	A Customer	2
150498	150499	0006641040	A3E7R866M94L0C	L. Barker "simienwolf"	2
515425	515426	141278509X	AB1A5EGHHVA9M	CHelmic	1
24750	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0
24749	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1
308076	308077	2841233731	A3QD68O22M2XHQ	LABRNTH	0
171160	171161	7310172001	AFXMWPNS1BLU4	H. Sandler	0
171159	171160	7310172001	A74C7IARQEM1R	stucker	0
171143	171144	7310172001	A1V5MY8V9AWUQB	Cheryl Sapper "champagne girl"	0
171142	171143	7310172001	A2SWO60IW01VPX	Sam	0
171147	171148	7310172001	A3TFTWTG2CC1GA	J. Umphress	0

171141	171142	7310172001	A2ZO1AYFVQYG44	Cindy Rellie "Rellie"	0

OBSERVATION :- Here books with ProductId - 0006641040 and 2841233731 are also there so we have to remove all these rows with these ProductIds from the data

```
final = final[final['ProductId'] != '2841233731']
final = final[final['ProductId'] != '0006641040']
final.shape
```

```
(364136, 10)
```

Text Preprocessing: Stemming, stop-word removal and Lemmatization.¶

```
# Downloading stopwords
nltk.download('stopwords')

#set of stopwords in English
from nltk.corpus import stopwords
stop = set(stopwords.words('english'))
words_to_keep = set(('not'))
stop -= words_to_keep
#initialising the snowball stemmer
sno = nltk.stem.SnowballStemmer('english')

 #function to clean the word of any html-tags
def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

#function to clean the word of any punctuation or special characters
def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

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

```
#Code for removing HTML tags , punctuations . Code for removing stopwords . Code for check
# also greater than 2 . Code for stemmimg and also to convert them to lowercase letters
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 1:
                        all_positive_words.append(s) #list of all words used to describe p
```

```
                    if(final['Score'].values)[i] == 0:
                        all_negative_words.append(s) #list of all words used to describe n
                else:
                    continue
            else:
                continue

        str1 = b" ".join(filtered_sentence) #final string of cleaned words


        final_string.append(str1)
        i+=1


#adding a column of CleanedText which displays the data after pre-processing of the review
final['CleanedText']=final_string
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
#below the processed review can be seen in the CleanedText Column
print('Shape of final',final.shape)
final.head()
```

```
Shape of final (364136, 11)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	He
515425	515426	141278509X	AB1A5EGHHVA9M	CHelmic	1	
24750	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	
24749	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	

Converting this data as IMDB dataset

```
##Sorting data according to Time in ascending order for Time Based Splitting
time_sorted_data = final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='

x = time_sorted_data['CleanedText'].values
y = time_sorted_data['Score']

# Finding all words in the vocabulary
count_vect = CountVectorizer()
count_vect.fit(x)

vocabulary = count_vect.get_feature_names()
print('No. of words in the Vocabulary : ',len(vocabulary))
```

```
No. of words in the Vocabulary :  71611
```

```
# Storing all words in the dictionary (words as keys and index as values)
corpus = dict()
```

```
ind = 0
for sent in x:
  for word in sent.split():
    corpus.setdefault(word,[])
    corpus[word].append(ind)
    ind += 1

# Getting frequency for each word of vocabulary and storing it in a list
freq = []
for w in vocabulary:
  freq.append(len(corpus[w]))



# Getting Index for each word in the vocabulary
# Sorting frequencies in decreasing order
inc_index =np.argsort(np.array(freq))[::-1]

# Allocating ranks to words of vocabulary in decreasing order of frequency and storing wor
word_rank = dict()
rank = 1
for i in inc_index:
  word_rank[vocabulary[i]] = rank
  rank +=1


# Converting full data into imdb format
data = []
for sent in x:
  row = []
  for word in sent.split():
    if(len(word)>1):
      row.append(word_rank[word])
  data.append(row)

# Splitting the data into 50-50 train_data and test_data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(data, y, test_size=0.5, random_state=4


print("No. of datapoints in X_train :",len(X_train))
print("No. of datapoints in X_test :",len(X_test))
print("Shape of Y_train :",Y_train.shape)
print("Shape of Y_test :",Y_test.shape)
```

```
No. of datapoints in X_train : 182068
No. of datapoints in X_test : 182068
Shape of Y_train : (182068,)
Shape of Y_test : (182068,)
```

```
182068+182068
```

```
364136
```

```
# Importing libraries
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 keras.layers import Dropout
# fix random seed for reproducibility
np.random.seed(7)
```

```
Using TensorFlow backend.
```

```
# truncate and/or pad input sequences
max_review_length = 100
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

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

```
(182068, 100)
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 152
  30 241  93 877 313 117  14 329  47  67]
```

```
# this function is used draw Binary Crossentropy Loss VS No. of epochs plot
def plt_dynamic(x, vy, ty):
  plt.figure(figsize=(10,5))
  plt.plot(x, vy, 'b', label="Validation Loss")
  plt.plot(x, ty, 'r', label="Train Loss")
  plt.xlabel('Epochs')
  plt.ylabel('Binary Crossentropy Loss')
  plt.title('\nBinary Crossentropy Loss VS Epochs')
  plt.legend()
  plt.grid()
  plt.show()
```

(1) RNN with 1 LSTM layer

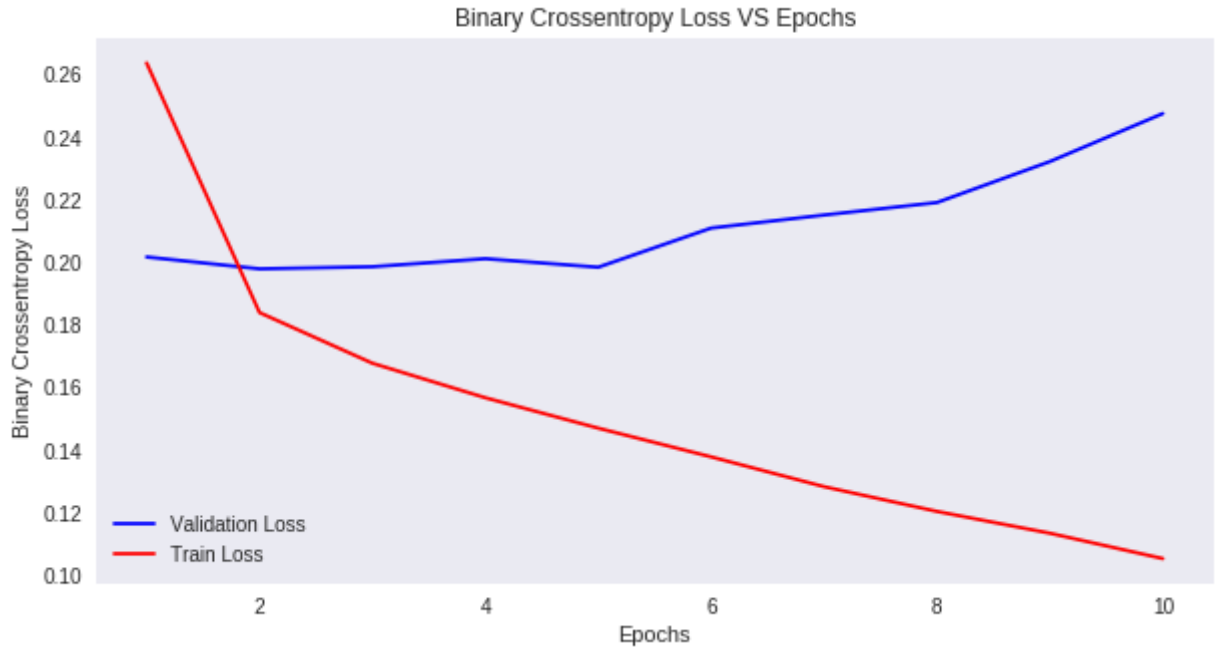
```
# create the model
embedding_vecor_length = 32

# Initialising the model
model_1 = Sequential()

# Adding embedding
model_1.add(Embedding(len(vocabulary), embedding_vecor_length, input_length=max_review_len

# Adding Dropout
model_1.add(Dropout(0.2))

# Adding first LSTM layer
model_1.add(LSTM(100))

# Adding Dropout
model_1.add(Dropout(0.2))

# Adding output layer
model_1.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_1.summary())

# Compiling the model
model_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_1 = model_1.fit(X_train, Y_train, nb_epoch=10, batch_size=512 ,verbose=1,validatio
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/fr
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensor
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_pr
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 32)           2291552
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 32)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 2,344,853
Trainable params: 2,344,853
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/op
Instructions for updating:
Use tf.cast instead.
Train on 182068 samples, validate on 182068 samples
Epoch 1/10
182068/182068 [==============================] - 77s 423us/step - loss: 0.2639 - ac
Epoch 2/10
182068/182068 [==============================] - 74s 409us/step - loss: 0.1841 - ac
Epoch 3/10
182068/182068 [==============================] - 74s 406us/step - loss: 0.1680 - ac
Epoch 4/10
182068/182068 [==============================] - 73s 403us/step - loss: 0.1569 - ac
Epoch 5/10
182068/182068 [==============================] - 73s 402us/step - loss: 0.1472 - ac
Epoch 6/10
182068/182068 [==============================] - 73s 400us/step - loss: 0.1381 - ac
Epoch 7/10
182068/182068 [==============================] - 75s 413us/step - loss: 0.1285 - ac
Epoch 8/10
182068/182068 [==============================] - 74s 407us/step - loss: 0.1206 - ac
Epoch 9/10
182068/182068 [==============================] - 74s 405us/step - loss: 0.1137 - ac
```

```
# Final evaluation of the model
scores = model_1.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Test and train accuracy of the model
model_1_test = scores[1]
model_1_train = max(history_1.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,11))

# Validation loss
vy = history_1.history['val_loss']
# Training loss
ty = history_1.history['loss']

# Calling the function to draw the plot
```

```
plt_dynamic(x, vy, ty)
```

```
Accuracy: 91.66%
```



(2). RNN with 2 LSTM layers

```
# create the model
embedding_vecor_length = 32

# Initialising the model
model_2 = Sequential()

# Adding embedding
model_2.add(Embedding(len(vocabulary), embedding_vecor_length, input_length=max_review_len

# Adding first LSTM layer
model_2.add(LSTM(100,return_sequences=True, dropout=0.4, recurrent_dropout=0.4))

# Adding second LSTM layer
model_2.add(LSTM(100, dropout=0.4, recurrent_dropout=0.4))

# Adding output layer
model_2.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_2.summary())

# Compiling the model
model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_2 = model_2.fit(X_train, Y_train, nb_epoch=10, batch_size=512 ,verbose=1,validatio
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_2 (Embedding)      (None, 100, 32)           2291552   
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 100)          53200     
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400     
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101       
=================================================================
Total params: 2,425,253
Trainable params: 2,425,253
Non-trainable params: 0
_________________________________________________________________
None
Train on 182068 samples, validate on 182068 samples
Epoch 1/10
182068/182068 [==============================] - 165s 906us/step - loss: 0.2707 - a
Epoch 2/10
182068/182068 [==============================] - 164s 901us/step - loss: 0.1930 - a
Epoch 3/10
182068/182068 [==============================] - 164s 903us/step - loss: 0.1781 - a
Epoch 4/10
182068/182068 [==============================] - 166s 911us/step - loss: 0.1680 - a
Epoch 5/10
182068/182068 [==============================] - 163s 893us/step - loss: 0.1600 - a
Epoch 6/10
```

```
# Final evaluation of the model
scores = model_2.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Test and train accuracy of the model
model_2_test = scores[1]
model_2_train = max(history_2.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,11))

# Validation loss
vy = history_2.history['val_loss']
# Training loss
ty = history_2.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Accuracy: 92.07%
```

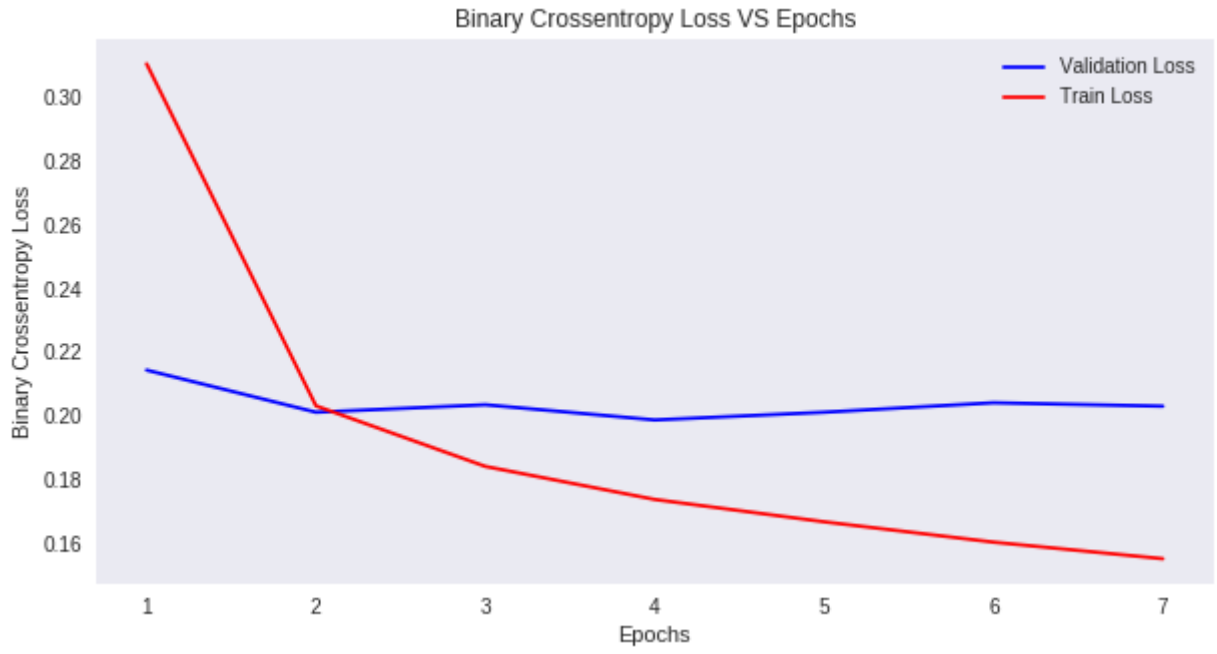
(3). RNN with 3 LSTM layers


```
# create the model
embedding_vecor_length = 32

# Initialising the model
model_3 = Sequential()

# Adding embedding
model_3.add(Embedding(len(vocabulary), embedding_vecor_length, input_length=max_review_len

# Adding first LSTM layer
model_3.add(LSTM(100,return_sequences=True, dropout=0.4, recurrent_dropout=0.4))

# Adding second LSTM layer
model_3.add(LSTM(100,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding third LSTM layer
model_3.add(LSTM(100, dropout=0.4, recurrent_dropout=0.4))

# Adding output layer
model_3.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_3.summary())

# Compiling the model
model_3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_3 = model_3.fit(X_train, Y_train, nb_epoch=7, batch_size=1024 ,verbose=1,validatio
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 100, 32)           2291552
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 100)          53200
_________________________________________________________________
lstm_5 (LSTM)                (None, 100, 100)          80400
```

```
# Final evaluation of the model
scores = model_3.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Test and train accuracy of the model
model_3_test = scores[1]
model_3_train = max(history_3.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,8))

# Validation loss
vy = history_3.history['val_loss']
# Training loss
ty = history_3.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Accuracy: 92.12%
```



(4). RNN with 4 LSTM layers

```
# create the model
embedding_vecor_length = 64

# Initialising the model
model_4 = Sequential()

# Adding embedding
model_4.add(Embedding(len(vocabulary), embedding_vecor_length, input_length=max_review_len

# Adding first LSTM layer
```

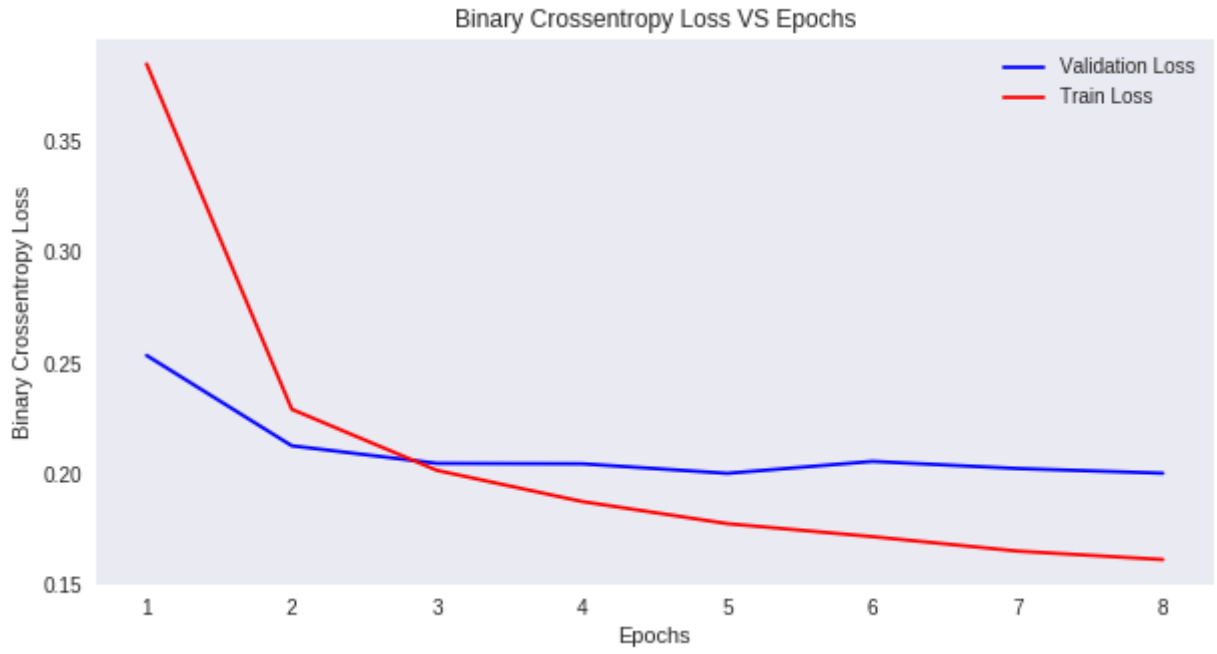
```
model_4.add(LSTM(120,return_sequences=True, dropout=0.6, recurrent_dropout=0.6))

# Adding second LSTM layer
model_4.add(LSTM(100,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding third LSTM layer
model_4.add(LSTM(80,return_sequences=True, dropout=0.4, recurrent_dropout=0.4))

# Adding fourth LSTM layer
model_4.add(LSTM(60, dropout=0.3, recurrent_dropout=0.3))

# Adding output layer
model_4.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_4.summary())

# Compiling the model
model_4.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_4 = model_4.fit(X_train, Y_train, nb_epoch=8, batch_size=2048 ,verbose=1,validatio
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 100, 64)           4583104
_________________________________________________________________
lstm_7 (LSTM)                (None, 100, 120)          88800
_________________________________________________________________
lstm_8 (LSTM)                (None, 100, 100)          88400
_________________________________________________________________
lstm_9 (LSTM)                (None, 100, 80)           57920
_________________________________________________________________
lstm_10 (LSTM)               (None, 60)                33840
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 61
=================================================================
Total params: 4,852,125
Trainable params: 4,852,125
Non-trainable params: 0
_________________________________________________________________
None
Train on 182068 samples, validate on 182068 samples
Epoch 1/8
182068/182068 [==============================] - 87s 480us/step - loss: 0.3853 - ac
Epoch 2/8
182068/182068 [==============================] - 84s 459us/step - loss: 0.2293 - ac
Epoch 3/8
182068/182068 [==============================] - 83s 455us/step - loss: 0.2016 - ac
Epoch 4/8
182068/182068 [==============================] - 83s 455us/step - loss: 0.1876 - ac
Epoch 5/8
182068/182068 [==============================] - 83s 457us/step - loss: 0.1776 - ac
Epoch 6/8
182068/182068 [==============================] - 84s 461us/step - loss: 0.1717 - ac
Epoch 7/8
182068/182068 [==============================] - 83s 455us/step - loss: 0.1651 - ac
Epoch 8/8
182068/182068 [==============================] - 82s 451us/step - loss: 0.1613 - ac
```

```
# Final evaluation of the model
scores = model_4.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
# Test and train accuracy of the model
model_4_test = scores[1]
model_4_train = max(history_4.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,9))

# Validation loss
vy = history_4.history['val_loss']
# Training loss
ty = history_4.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Accuracy: 92.21%
```




```
# create the model
embedding_vecor_length = 64

# Initialising the model
model_5 = Sequential()

# Adding embedding
model_5.add(Embedding(len(vocabulary), embedding_vecor_length, input_length=max_review_len

# Adding first LSTM layer
model_5.add(LSTM(120,return_sequences=True, dropout=0.6, recurrent_dropout=0.6))

# Adding second LSTM layer
model_5.add(LSTM(100,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding third LSTM layer
model_5.add(LSTM(80,return_sequences=True, dropout=0.4, recurrent_dropout=0.4))

# Adding third LSTM layer
model_5.add(LSTM(60,return_sequences=True, dropout=0.3, recurrent_dropout=0.3))

# Adding fourth LSTM layer
model_5.add(LSTM(40, dropout=0.2, recurrent_dropout=0.2))

# Adding output layer
model_5.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_5.summary())
```

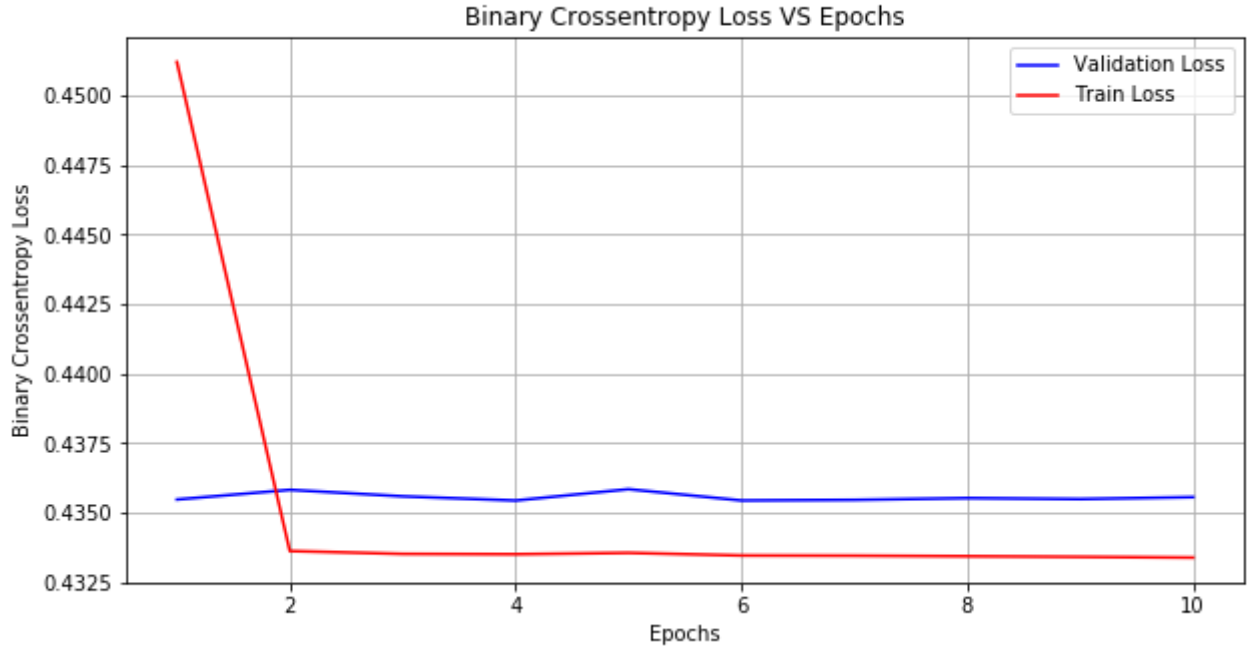
```
# Compiling the model
model_5.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_5 = model_5.fit(X_train, Y_train, nb_epoch=10, batch_size=2048 ,verbose=1,validati
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/fr
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensor
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_pr
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 64)           4583104
_________________________________________________________________
lstm_1 (LSTM)                (None, 100, 120)          88800
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 100)          88400
_________________________________________________________________
lstm_3 (LSTM)                (None, 100, 80)           57920
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 60)           33840
_________________________________________________________________
lstm_5 (LSTM)                (None, 40)                16160
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 41
=================================================================
Total params: 4,868,265
Trainable params: 4,868,265
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/op
Instructions for updating:
Use tf.cast instead.
Train on 182068 samples, validate on 182068 samples
Epoch 1/10
182068/182068 [==============================] - 118s 649us/step - loss: 0.4512 - a
Epoch 2/10
182068/182068 [==============================] - 110s 605us/step - loss: 0.4336 - a
Epoch 3/10
182068/182068 [==============================] - 111s 608us/step - loss: 0.4335 - a
Epoch 4/10
182068/182068 [==============================] - 111s 607us/step - loss: 0.4335 - a
Epoch 5/10
182068/182068 [==============================] - 108s 594us/step - loss: 0.4336 - a
Epoch 6/10
182068/182068 [==============================] - 110s 606us/step - loss: 0.4335 - a
Epoch 7/10
182068/182068 [==============================] - 108s 595us/step - loss: 0.4335 - a
Epoch 8/10
182068/182068 [==============================] - 107s 589us/step - loss: 0.4334 - a
Epoch 9/10
182068/182068 [==============================] - 109s 600us/step - loss: 0.4334 - a
Epoch 10/10
182068/182068 [==============================] - 108s 592us/step - loss: 0.4334 - a
```

```
# Final evaluation of the model
```

```
scores = model_5.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Test and train accuracy of the model
model_5_test = scores[1]
model_5_train = max(history_5.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,11))

# Validation loss
vy = history_5.history['val_loss']
# Training loss
ty = history_5.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Accuracy: 84.25%
```




```
from keras.layers.normalization import BatchNormalization


# create the model
embedding_vecor_length = 64

# Initialising the model
model_6 = Sequential()

# Adding embedding
model_6.add(Embedding(len(vocabulary), embedding_vecor_length, input_length=max_review_len

# Adding first LSTM layer
model_6.add(LSTM(150,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding second LSTM layer
model_6.add(LSTM(120,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding third LSTM layer
model_6.add(LSTM(100,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding Batch normalisation
model_6.add(BatchNormalization())

# Adding forth LSTM layer
model_6.add(LSTM(80,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))
```

```
# Adding fifth LSTM layer
model_6.add(LSTM(60,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding sixth LSTM layer
model_6.add(LSTM(40, dropout=0.5, recurrent_dropout=0.5))

# Adding output layer
model_6.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_6.summary())

# Compiling the model
model_6.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_6 = model_6.fit(X_train, Y_train, nb_epoch=10, batch_size=2048 ,verbose=1,validati
```

```
# Final evaluation of the model
scores = model_6.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Test and train accuracy of the model
model_6_test = scores[1]
model_6_train = max(history_6.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,11))

# Validation loss
vy = history_6.history['val_loss']
# Training loss
ty = history_6.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Accuracy: 92.15%
```

CONCLUSION

(a). Procedure Followed :

(b) Table (Different models with their train and test accuracy):

1. Upload Amazon Fine Food Reviews dataset on google drive to run it on GOOGLE colab.
2. Load Amazon Fine Food Reviews dataset from google drive.
3. Perform text pre-processing on text data.
4. Perform following 3 tasks to convert it into IMDB data format.

a. Sort the dataset on the basis of time and after that find vocabulary for all the reviews in the dataset
b. Now compute frequencies for each word of vocabulary.
c. Index each word in the decreasing order of frequencies (Word with max frequency will have rank 1 or index 1).

5. Convert the dataset into imdb dataset format.
6. Split whole dataset into 50-50 for training_data and test_data randomly.
7. Now pad or truncate each review intpo sequences of length 100.
8. Now implement RNN with 1 , 2 ,3,4,5 and 6 LSTM layers and 6th with batch normalisation.
9. Find accuracy for each above model.
10. Draw Binary Crossentropy Loss VS No.of Epochs plot.

PRETTY TABLE TO COMPAIR THE MODELS

```
# Installing the library prettytable
!pip install prettytable

# Creating table using PrettyTable library
from prettytable import PrettyTable

# Names of models
names = ['RNN With 1 LSTM Layer','RNN With 2 LSTM Layers','RNN With 3 LSTM Layers','RNN Wi

# Training accuracies
train_acc = [0.9604,0.9515,0.9414,0.9389,.8438,0.9447]

# Test accuracies
test_acc = [0.9166,0.9206,0.9211,0.9389,.8438,.9215]

numbering = [1,2,3,4,5,6]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Training Accuracy",train_acc)
ptable.add_column("Test Accuracy",test_acc)

# Printing the Table
print(ptable)
```

```
Requirement already satisfied: prettytable in /usr/local/lib/python3.6/dist-package
+-------+------------------------+--------------------+---------------+
| S.NO. |         MODEL          | Training Accuracy | Test Accuracy |
+-------+------------------------+--------------------+---------------+
|   1   | RNN With 1 LSTM Layer  |       0.9604       |     0.9166    |
|   2   | RNN With 2 LSTM Layers |       0.9515       |     0.9206    |
|   3   | RNN With 3 LSTM Layers |       0.9414       |     0.9211    |
|   4   | RNN With 4 LSTM Layers |       0.9389       |     0.9389    |
|   5   | RNN With 5 LSTM Layers |       0.8438       |     0.8438    |
|   6   | RNN With 6 LSTM Layers |       0.9447       |     0.9215    |
+-------+------------------------+--------------------+---------------+
```

OBSERVATION: