LSTM(Long Short Term Memory) on Amazon Fine Food Reviews

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:

Id ProductId - unique identifier for the product UserId - unqiue identifier for the user ProfileName HelpfulnessNumerator - number of users who found the review helpful HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not Score - rating between 1 and 5 Time - timestamp for the review Summary - brief summary of the review 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).

Loading,Cleaning & Preprocessing 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 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 id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".*italicized text*

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input director

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as output when you creat
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/amazon-fine-food-reviews/database.sqlite
/kaggle/input/amazon-fine-food-reviews/hashes.txt
/kaggle/input/amazon-fine-food-reviews/Reviews.csv
```

Import All Required Libraries

```
%matplotlib inline
```

```
import warnings

warnings.filterwarnings("ignore")
```

```
import sqlite3
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM,Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import string
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from tqdm import tqdm
import os
from tqdm import tqdm
import os
import pdb
import pickle
import re
import nltk
from collections import Counter
from itertools import islice
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.preprocessing import sequence
from keras.initializers import he_normal
from keras.layers import BatchNormalization, Dense, Dropout, Flatten, LSTM
from keras.layers.embeddings import Embedding
from keras.regularizers import L1L2
```

Connect to sqlite and fetch the data using SQL Query

```
con=sqlite3.connect('../input/amazon-fine-food-reviews/database.sqlite')

filtered_data=pd.read_sql_query("""SELECT * FROM Reviews WHERE Score !=3""",con)
filtered_data.head(3)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	13038

```
filtered_data.shape
```

```
(525814, 10)
```

```
def partition(x):
  if x < 3 :
    return 'negative'
  return 'positive'

actualScore=filtered_data['Score']
```

```
positive_negative=actualScore.map(partition)
filtered_data['Score']=positive_negative
print("Number of datapoints",filtered_data.shape)
filtered_data.head(3)
```

```
Number of datapoints (525814, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	positive	130

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

```
print(display.shape)
display.head(3)
```

```
(80668, 7)
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3

```
display[display["UserId"]=='AZY10LLTJ71NX']
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

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

```
393063
```

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199

```
sorted_data=filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=False,kind='quicksort',na_position

final_data=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep='first',inplace=False)
final_data.shape
```

```
(364173, 10)
```

```
(final_data['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
69.25890143662969
```

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

display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224

```
final_data=final_data[final_data.HelpfulnessNumerator<=final_data.HelpfulnessDenominator]
```

Count the Positive and Negative Review Counts

```
print(final_data.shape)

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

```
(364171, 10)
positive    307061
negative     57110
Name: Score, dtype: int64
```

Import nltk library

```

from nltk.corpus import stopwords
stopping_words=set(stopwords.words('english'))
print(stopping_words)
```

```
{'yourselves', "won't", 'ma', 'herself', 'them', 'didn', 'should', 'your', 'an', "haven't", 'wasn', 'his', 'who
```

```
def clean_html(text):
    clean_r = re.compile('<,*?>')
    clean_text = re.sub(clean_r,'',text)
    return clean_text

def Clean_punc(text):
    clean_sentence = re.sub(r'[?|!|\'|"|#]',r' ',text)
    clean_data = re.sub(r'[.|,|)|(|\|/)]',r' ',clean_sentence)
    return clean_data
```

```
stem_no = nltk.stem.SnowballStemmer('english')

if not os.path.isfile('final_data.sqlite'):
    final_string=[]
    all_positive_words=[]
    all_negative_words=[]
    for i,sentence in enumerate(tqdm(final_data['Text'].values)):
        filtered_sentence=[]
        sent_without_html_tags=clean_html(sentence)
        #pdb.set_trace()
        for w in sent_without_html_tags.split():
            for cleaned_words in Clean_punc(w).split():
                if ((cleaned_words.isalpha()) & (len(cleaned_words) > 2)):
                    if(cleaned_words.lower() not in stopping_words) :
                        stemming=(stem_no.stem(cleaned_words.lower())).encode('utf8')
                        filtered_sentence.append(stemming)
                        if(final_data['Score'].values)[i]=='positive':
                            all_positive_words.append(stemming)
                        if(final_data['Score'].values)[i]=='negative':
                            all_negative_words.append(stemming)
        str1 = b" ".join(filtered_sentence)
        final_string.append(str1)

    final_data['Cleaned_text']=final_string
    final_data['Cleaned_text']=final_data['Cleaned_text'].str.decode("utf-8")

    conn = sqlite3.connect('final_data.sqlite')
    cursor=conn.cursor
    conn.text_factory = str
    final_data.to_sql('Reviews',conn,schema=None,if_exists='replace',index=True,index_label=None,chunksize=None,dt
    conn.close()


    with open('positive_words.pkl','wb') as f :
        pickle.dump(all_positive_words,f)
    with open('negative_words.pkl','wb') as f :
        pickle.dump(all_negative_words,f)
```

```
100%|██████████| 364171/364171 [08:20<00:00, 727.25it/s]
```

```
final_data['total_words'] = [len(x.split()) for x in final_data['Cleaned_text'].tolist()]
```

```
final_data.head(3)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	positive
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1	positive

```
final_data.shape
```

```
(364171, 12)
```

Pick the top 100K Data Points from the Dataset

```
final_data_100K=final_data[0:100000]
amazon_polarity_labels=final_data_100K['Score'].values
final_data_100K.head(2)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	positive

```
X_Data_Text=final_data_100K['Cleaned_text']
```

```
Y_Data_Score=final_data_100K['Score']
```

```
X_Data_Text.head(3)
```

```
    138706    witti littl book make son laugh loud recit car...
```

```
X_Data_Text.index=[x for x in range(0,10**5)]
```

```
    Name: cleaned_text, dtype: object
```

Get the list of all the words from the "Cleaned Text"

```
list_of_words = []
for i in tqdm(X_Data_Text):
    list_of_words += i
list_of_words = ''.join(list_of_words)
list_of_all_words=list_of_words.split()
```

```
100%|██████████| 100000/100000 [00:00<00:00, 318290.67it/s]
```

```
list_of_all_words[0]
```

```
'witti'
```

Convert the List of words into SET and List

```
vocab=set(list_of_all_words)
vocab_list=list(vocab)
```

```
type(vocab)
```

```
set
```

```
vocab_list[0]
```

```
'lisa'
```

```
dict_freq={}
for x in tqdm(vocab_list):
    dict_freq[x]=list_of_all_words.count(x)
```

```
100%|██████████| 103077/103077 [1:34:36<00:00, 18.16it/s]
```

```
dict_freq
```

SHOW HIDDEN OUTPUT

```
(pd.DataFrame.from_dict(data=dict_freq, orient='index')
   .to_csv('dict_file.csv', header=False))
```

Use Counter function to count the number of Unique words and pick the top 5000 words and add in the dictionary

```
print("Number of sentences in complete dataset : ",len(list_of_all_words))

counts_words = Counter(list_of_all_words)
print("Number of unique words present : ",len(counts_words.most_common()))
vocab_size = len(counts_words.most_common()) + 1
top_words_count = 5000
```

```
common_words = counts_words.most_common(top_words_count)

word_index = dict()
i = 1
for word,frequency in common_words:
    word_index[word] = i
    i += 1

print()
print("Top 25 words with their frequencies:")
print(counts_words.most_common(25))
print()
print("Top 25 words with their index:")
print(list(islice(word_index.items(), 25)))
```

```
Number of sentences in complete dataset :  3488783
Number of unique words present :  103077

Top 25 words with their frequencies:
[('like', 38877), ('tast', 37698), ('tea', 33156), ('good', 29934), ('product', 29213), ('use', 29058), ('flavor

Top 25 words with their index:
[('like', 1), ('tast', 2), ('tea', 3), ('good', 4), ('product', 5), ('use', 6), ('flavor', 7), ('one', 8), ('gre
```

```
type(X_Data_Text)
```

```
pandas.core.series.Series
```

Create a new Column called "CleanedText_Index" and add the index of each word occured in the "Cleaned_text"

```
def use_index(row):
    holder = []
    for word in row['Cleaned_text'].split():
        if word in word_index:
            holder.append(word_index[word])
        else:
            holder.append(0)
    return holder


final_data_100K['CleanedText_Index'] = final_data_100K.apply(lambda row: use_index(row),axis=1)
final_data_100K.head(5)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Scor
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positiv
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	positiv
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1	positiv

Convert the Score to 1 & 0.

positive - 1

negative - 0

```
final_data_100K['Score'] = final_data_100K['Score'].map(lambda x : 1 if x == 'positive' else 0)
final_data_100K.head(5)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1

```
final_data_100K['CleanedText_Index'].head(2)
```

```
138706    [0, 27, 932, 11, 384, 1976, 2578, 0, 1196, 123...
138688    [995, 247, 0, 932, 551, 23, 0, 988, 2594, 10, ...
Name: CleanedText_Index, dtype: object
```

Split the Data into TEST and Train

138689 150507 0006641040 A1S4A3IQ2MU7V4 "sally sue" 1 1 1

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(final_data_100K['CleanedText_Index'].values,final_data_100K['S
```

```
X_Train[6]
```

```
[365,
 1773,
 266,
```

Apply Padding

```
 384,
```

```
from keras.preprocessing import sequence

max_review_length = 600
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])
```

```
(70000, 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  995
  247    0  932  551   23    0  988 2594   10  384   10  149  558  111
  562  366    0  113  211 3563   82   63  252   68 1751  166]
```

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 6202525998069733733
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 15694142288273005724
physical_device_desc: "device: XLA_CPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 15695549568
locality {
  bus_id: 1
  links {
  }
}
incarnation: 16992963004178590259
physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0'
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
```

Apply LSTM 1

```
physical_device_desc: "device: XLA_GPU device"
```

```
import numpy
numpy.random.seed(7)

embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words_count+1, 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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 600, 32)           160032    
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200     
_________________________________________________________________
dense (Dense)                (None, 1)                 101       
=================================================================
Total params: 213,333
Trainable params: 213,333
Non-trainable params: 0
_________________________________________________________________
None
```

```
model.fit(X_Train, Y_Train, epochs=10, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_Test, Y_Test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 1/10
1094/1094 [==============================] - 35s 32ms/step - loss: 0.2506 - accuracy: 0.9027
Epoch 2/10
1094/1094 [==============================] - 34s 31ms/step - loss: 0.1884 - accuracy: 0.9275
Epoch 3/10
1094/1094 [==============================] - 35s 32ms/step - loss: 0.1729 - accuracy: 0.9331
Epoch 4/10
1094/1094 [==============================] - 34s 31ms/step - loss: 0.1582 - accuracy: 0.9391
Epoch 5/10
1094/1094 [==============================] - 35s 32ms/step - loss: 0.1433 - accuracy: 0.9451
Epoch 6/10
1094/1094 [==============================] - 34s 32ms/step - loss: 0.1301 - accuracy: 0.9502
Epoch 7/10
1094/1094 [==============================] - 34s 31ms/step - loss: 0.1176 - accuracy: 0.9563
Epoch 8/10
```

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

```
Test score:  0.3073323667049408
Test accuracy:  0.9080333113670349
```

```
embedding_vecor_length = 32
model2 = Sequential()
model2.add(Embedding(top_words_count+1, embedding_vecor_length, input_length=max_review_length))
model2.add(LSTM(100,return_sequences=True))
model2.add(Dropout(0.25))
model2.add(LSTM(80))
model2.add(Dropout(0.5))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model2.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_2 (Embedding)      (None, 600, 32)           160032    
_________________________________________________________________
lstm_3 (LSTM)                (None, 600, 100)          53200     
_________________________________________________________________
dropout_1 (Dropout)          (None, 600, 100)          0         
_________________________________________________________________
lstm_4 (LSTM)                (None, 80)                57920     
_________________________________________________________________
dropout_2 (Dropout)          (None, 80)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 81        
=================================================================
Total params: 271,233
Trainable params: 271,233
Non-trainable params: 0
_________________________________________________________________
None
```

```
model2.fit(X_Train, Y_Train, epochs=10, batch_size=64)
# Final evaluation of the model
scores = model2.evaluate(X_Test, Y_Test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 1/10
1094/1094 [==============================] - 62s 56ms/step - loss: 0.2522 - accuracy: 0.9031
Epoch 2/10
1094/1094 [==============================] - 61s 56ms/step - loss: 0.1947 - accuracy: 0.9255
Epoch 3/10
1094/1094 [==============================] - 62s 56ms/step - loss: 0.1744 - accuracy: 0.9338
Epoch 4/10
1094/1094 [==============================] - 62s 57ms/step - loss: 0.1582 - accuracy: 0.9408
Epoch 5/10
1094/1094 [==============================] - 63s 57ms/step - loss: 0.1411 - accuracy: 0.9471
Epoch 6/10
1094/1094 [==============================] - 62s 57ms/step - loss: 0.1289 - accuracy: 0.9530
Epoch 7/10
```

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

```
Test score:  0.3317498564720154
Test accuracy:  0.9067999720573425
1094/1094 [==============================] - 62s 56ms/step - loss: 0.0800 - accuracy: 0.9733
```

CONCLUSION

After using LSTM Models , below are the conlusion made

1. LSTM without Dropouts --> Accuracy is 90.80%
2. LSTM with Dropout --> Accuracy is 90.67%