LSTM_IMDB

August 22, 2019

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 [1]:
```
%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 CountVectorizer
from sklearn import metrics
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
import os
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.layers import Input, Embedding, LSTM, Dense, Flatten, Dropout
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
```

```
Using TensorFlow backend.
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('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 5
# 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[2]:

```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      1  1303862400
1                     0                       0      0  1346976000
2                     1                       1      1  1219017600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

In [3]:

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

In [4]:

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

```
(80668, 7)
```

```
Out[4]:                UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

```
In [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[5]:               UserId   ProductId                       ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                              Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

```
In [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 393063
```

3 [2] Exploratory Data Analysis

4 [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 [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

```
Out[7]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
```

```
2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                             Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

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 [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

Out[9]: (46072, 10)

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:
```
      Id   ProductId          UserId              ProfileName  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                    Summary  \
0              Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                                Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#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[13]:
```
1    38479
0     7592
Name: Score, dtype: int64
```

5 [3] Preprocessing

5.1 [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 [14]:
```
# 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)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being
==================================================
It's Branston pickle, what is there to say. If you've never tried it you most likely wont like
==================================================
First Impression: The friendly folks over at "Exclusively Dog" heard about my website and sent
==================================================
It is hard to find candy that is overly sweet. My wife and Granddaughter both love Pink Grapef
==================================================
```

In [15]:
```
# 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)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all


         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)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being
==================================================
It's Branston pickle, what is there to say. If you've never tried it you most likely wont like
==================================================
First Impression: The friendly folks over at "Exclusively Dog" heard about my website and sent
==================================================
It is hard to find candy that is overly sweet. My wife and Granddaughter both love Pink Grapefr
```

```
In [14]: # 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 [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
First Impression: The friendly folks over at "Exclusively Dog" heard about my website and sent
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
First Impression The friendly folks over at Exclusively Dog heard about my website and sent me
```

In [15]:
```
# 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', 'ourselve
            "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', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
```

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

```
In [16]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(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 stopw
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 46071/46071 [00:18<00:00, 2475.85it/s]
```

```
In [17]: preprocessed_reviews[1500]
```

```
Out[17]: 'great flavor low calories high nutrients high protein usually protein powders high p
```

```
In [18]: x = preprocessed_reviews
         y = final["Score"].values
```

5.1.1 Splitting data in train and test data

```
In [19]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state
```

```
In [20]: print(len(x_train))
         print(len(x_test))
```

```
36856
9215
```

5.1.2 Tokenizing

```
In [21]: max_features = 5000
         t = Tokenizer(num_words=max_features, lower = False)
         t.fit_on_texts(x_train)

         x_train_encoded = t.texts_to_sequences(x_train)
```

```
In [22]: x_test_encoded = t.texts_to_sequences(x_test)
```

```
In [23]: print(type(x_train_encoded))
         print(len(x_train_encoded))
```

```
<class 'list'>
36856
```

5.1.3 Padding

```
In [24]: max_length   = 55
         x_train_padded = pad_sequences(x_train_encoded, maxlen= max_length, padding='post')
         print(x_train_padded.shape)
         print(x_train_padded)
```

```
(36856, 55)
[[  24   67   99 ...  234   26    8]
 [1100  130 1288 ...    0    0    0]
 [  77  972 1445 ...    0    0    0]
 ...
 [  16    1   30 ...    0    0    0]
 [2382   68  189 ...    0    0    0]
 [ 959  566  115 ...    0    0    0]]
```

```
In [25]: max_length   = 55
         x_test_padded = pad_sequences(x_test_encoded, maxlen= max_length, padding='post')
         print(x_test_padded.shape)
         print(x_test_padded)
```

```
(9215, 55)
[[  98  427    8 ...    0    0    0]
 [ 129   23  343 ...    0    0    0]
 [ 414  340   27 ...    0    0    0]
 ...
 [ 526  518  481 ...    0    0    0]
 [  44 1353  263 ...    0    0    0]
 [  12   80  848 ...    0    0    0]]
```

```
In [26]: print("shape of training data: ", x_train_padded.shape)
         print("shape of testing data: ", x_test_padded.shape)
```

```
shape of training data:  (36856, 55)
shape of testing data:  (9215, 55)
```

6 Model 1

```
In [28]: embedding_vecor_length = 32
         model = Sequential()
         model.add(Embedding(max_features, embedding_vecor_length, input_length=max_length))
```

```python
model.add(LSTM(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 55, 32)            160000
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                24832
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 184,897
Trainable params: 184,897
Non-trainable params: 0
_________________________________________________________________
None
```

In [29]:
```python
model.fit(x_train_padded, y_train, epochs = 10,verbose = 2, batch_size = 128,validatio
```

```
Train on 36856 samples, validate on 9215 samples
Epoch 1/10
 - 20s - loss: 0.4848 - acc: 0.8305 - val_loss: 0.3536 - val_acc: 0.8479
Epoch 2/10
 - 18s - loss: 0.2788 - acc: 0.8805 - val_loss: 0.2381 - val_acc: 0.9046
Epoch 3/10
 - 19s - loss: 0.2096 - acc: 0.9231 - val_loss: 0.2187 - val_acc: 0.9169
Epoch 4/10
 - 17s - loss: 0.1808 - acc: 0.9333 - val_loss: 0.2451 - val_acc: 0.9159
Epoch 5/10
 - 17s - loss: 0.1722 - acc: 0.9374 - val_loss: 0.2307 - val_acc: 0.9162
Epoch 6/10
 - 17s - loss: 0.1657 - acc: 0.9427 - val_loss: 0.2419 - val_acc: 0.9037
Epoch 7/10
 - 16s - loss: 0.1433 - acc: 0.9486 - val_loss: 0.2478 - val_acc: 0.8942
Epoch 8/10
 - 16s - loss: 0.1355 - acc: 0.9524 - val_loss: 0.2863 - val_acc: 0.9022
Epoch 9/10
 - 16s - loss: 0.1662 - acc: 0.9466 - val_loss: 0.2953 - val_acc: 0.9104
Epoch 10/10
 - 17s - loss: 0.1343 - acc: 0.9546 - val_loss: 0.2984 - val_acc: 0.9122
```

Out[29]:
```
<keras.callbacks.History at 0x2bfd0cd1fd0>
```

6.0.1 Model 2

```
In [30]: embedding_vecor_length = 32
         model_2 = Sequential()
         model_2.add(Embedding(max_features, embedding_vecor_length, input_length=max_length))
         model_2.add(LSTM(64, return_sequences=True, activation='relu'))
         model_2.add(LSTM(64, activation='relu'))
         model_2.add(Dropout(0.5))
         model_2.add(Dense(1, activation='sigmoid'))
         model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         print(model_2.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 55, 32)            160000    
_________________________________________________________________
lstm_3 (LSTM)                (None, 55, 64)            24832     
_________________________________________________________________
lstm_4 (LSTM)                (None, 64)                33024     
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
=================================================================
Total params: 217,921
Trainable params: 217,921
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [33]: model_2.fit(x_train_padded, y_train, epochs = 10,verbose = 2, batch_size = 128,valida
```

```
Train on 36856 samples, validate on 9215 samples
Epoch 1/10
 - 37s - loss: 1.3150 - acc: 0.8621 - val_loss: 0.7897 - val_acc: 0.8697
Epoch 2/10
 - 37s - loss: 0.5445 - acc: 0.8925 - val_loss: 0.7327 - val_acc: 0.8906
Epoch 3/10
 - 37s - loss: 0.5033 - acc: 0.9146 - val_loss: 0.7785 - val_acc: 0.8800
Epoch 4/10
 - 37s - loss: 0.5605 - acc: 0.9195 - val_loss: 1.0056 - val_acc: 0.8698
Epoch 5/10
 - 37s - loss: 0.5171 - acc: 0.9154 - val_loss: 0.4909 - val_acc: 0.8877
Epoch 6/10
 - 37s - loss: 0.3480 - acc: 0.9310 - val_loss: 0.6089 - val_acc: 0.8981
Epoch 7/10
 - 37s - loss: 0.3429 - acc: 0.9376 - val_loss: 0.5881 - val_acc: 0.8909
```

```
Epoch 8/10
 - 37s - loss: 0.3164 - acc: 0.9438 - val_loss: 0.5963 - val_acc: 0.8862
Epoch 9/10
 - 37s - loss: 0.3030 - acc: 0.9467 - val_loss: 0.5716 - val_acc: 0.8921
Epoch 10/10
 - 37s - loss: 0.2829 - acc: 0.9507 - val_loss: 0.5775 - val_acc: 0.8830
```

```
Out[33]: <keras.callbacks.History at 0x2bfdf4d1710>
```

6.0.2 Conclusions

6.0.3 1. I took 50k reviews for this assignment.

6.0.4 2. Two models are made Model_1 only have one LSTM layer, where as Model_2 includes two LSTM layers.

6.0.5 3. After we preprocess the reviews, we tokenize the data as we can't directly pass text data into the model.

6.0.6 4. So after we tokenize we pad the data as all the data which we input should be similar in dimensions.

6.0.7 5. Then for modeling we used sequential models, which are simple and pretty straight forward.

6.0.8 6. Model 1 is containing only one LSTM layer with relu activation, it also consists of one dropout layer as well.

6.0.9 7. Model 2 is containing two LSTM layers and one dropout layer.

6.0.10 8. For final output layer dense layer is used with sigmoid activation.

6.0.11 8. Loss function used is binary_crossentrop here as we have a binary classification problem at hand.

6.0.12 10. Optimizer used for both the models is adam and performace metric we used for both models is accuracy.