Amazon_Food_Reviews_LSTM

July 8, 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).

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 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".

```
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 TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

# importing Cross validation libs
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import model_selection

from matplotlib import pyplot
import seaborn as sns

import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.layers import BatchNormalization, Dense, Dropout, Flatten, LSTM
```

```
Using TensorFlow backend.
```

1.2 Reading Data

```
In [2]: # Code to read csv file into colaboratory:
        !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

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



        id = '1VDFn9tpwOOecnr5DiC7TQwg-GVoPJUEK'

        downloaded = drive.CreateFile({'id':id})
        downloaded.GetContentFile('database.sqlite')

        # df2 = pd.read_csv(io.BytesIO(uploaded['Filename.csv']))

        # using the 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, and reviews with a score<3 a negative r
        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)
```

```
     || 993kB 5.1MB/s
  Building wheel for PyDrive (setup.py) ... done


WARNING: Logging before flag parsing goes to stderr.
W0707 19:22:06.110231 140405479520128 __init__.py:44] file_cache is unavailable when using oau
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/__init__.py", l
    from google.appengine.api import memcache
ModuleNotFoundError: No module named 'google.appengine'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/file_cache.py",
    from oauth2client.contrib.locked_file import LockedFile
ModuleNotFoundError: No module named 'oauth2client.contrib.locked_file'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/file_cache.py",
    from oauth2client.locked_file import LockedFile
ModuleNotFoundError: No module named 'oauth2client.locked_file'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/__init__.py", l
    from . import file_cache
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/file_cache.py",
    'file_cache is unavailable when using oauth2client >= 4.0.0 or google-auth')
ImportError: file_cache is unavailable when using oauth2client >= 4.0.0 or google-auth


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

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

        [3 rows x 10 columns]
```

```
In [0]: 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  ... COUNT(*)
        0  #oc-R115TNMSPFT9I7  ...        2
        1  #oc-R11D9D7SHXIJB9  ...        3
        2  #oc-R11DNU2NBKQ23Z  ...        2
        3  #oc-R11O5J5ZVQE25C  ...        3
        4  #oc-R12KPBODL2B5ZD  ...        2

        [5 rows x 7 columns]
```

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

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

        [1 rows x 7 columns]
```

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

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

2 [2] Exploratory Data Analysis

2.1 [2.1] Data Cleaning: Deduplication

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

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

```
Out[7]:       Id  ...                                           Text
        0   78445  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        1  138317  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        2  138277  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        3   73791  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        4  155049  ...  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...

        [5 rows x 10 columns]
```

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

```
In [9]: 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  ...                                               Text
         0  64422  ...  My son loves spaghetti so I didn't hesitate or...
         1  44737  ...  It was almost a 'love at first bite' - the per...

         [2 rows x 10 columns]
```

```
In [0]: 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
```

3 [3] Preprocessing

3.1 [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 [0]:

```
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)
```

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

```
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)
```

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

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
        from bs4 import BeautifulSoup

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

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

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

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

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

```
In [0]: # 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]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

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

```
In [19]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

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

```
In [0]: # https://gist.github.com/sebleier/554280
        # we are removing the words from the stop words list: 'no', 'nor', 'not'
        # <br /><br /> ==> after the above steps, we are getting "br br"
        # we are including them into stop words list
        # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

        stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves
                    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "t
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", '
                    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [0]: #filtered out whole reviews

        from bs4 import BeautifulSoup
        # Combining all the above stundents
        from tqdm import tqdm
        # tqdm is for printing the status bar
        word_counter = []
        def filterised_text(text):
            preprocessed_text = []
            for sentance in tqdm(text):
                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 sto
                count = len(sentance.split())
                word_counter.append(count)
                preprocessed_text.append(sentance.strip())
            return preprocessed_text
```

```
In [22]: preprocessed_reviews = filterised_text(final['Text'].values)
         final['preprocessed_reviews']  = preprocessed_reviews
         preprocessed_reviews[1822]
```

```
100%|| 46071/46071 [00:15<00:00, 3018.84it/s]
```

```
Out[22]: 'gobble one want dachsund mix horrible breath helps little definitely buy price right
```

```
In [23]: final['numbers_of_words']  = word_counter
         word_counter[1822]
```

```
Out[23]: 13
```

3.2 [3.2] Preprocessing Review Summary

```
In [24]: preprocessed_summary = filterised_text(final['Summary'].values)
         final['preprocessed_summary']  = preprocessed_summary
         preprocessed_summary[1822]
```

```
100%|| 46071/46071 [00:10<00:00, 4563.45it/s]
```

```
Out[24]: 'dog love'
```

3.2.1 Splitting data

```
In [25]: X = final['preprocessed_reviews']
         y = final['Score']
```

```
        # split the data set into train and test
        X_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0

        print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(32249,) (13822,) (32249,) (13822,)
```

4 [4] Featurization

4.1 [4.1] Vectorize Data

```
In [26]: # https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/


         from keras.preprocessing.text import Tokenizer
         from keras.preprocessing.sequence import pad_sequences

         max_review_length = 500
         top_words = 5000

         tk = Tokenizer(lower = True, num_words= top_words)
         tk.fit_on_texts(X_train)
         X_train_seq = tk.texts_to_sequences(X_train)
         X_train_pad = pad_sequences(X_train_seq, maxlen=max_review_length)
         X_test_seq = tk.texts_to_sequences(x_test)
         X_test_pad = pad_sequences(X_test_seq, maxlen=max_review_length)


         print('train data shape ', X_train_pad.shape)
         print('test data shape', X_test_pad.shape)
```

```
train data shape  (32249, 500)
test data shape (13822, 500)
```

```
In [27]: # Sequence

         print(X_train_seq[1])
         print(type(X_train_seq[1]))
         print(len(X_train_seq[1]))
```

```
[3396, 7, 34, 1, 47, 442, 58, 2, 63, 992]
<class 'list'>
10
```

```
In [28]: # Padding
```

```python
print(X_train_pad[1])
print(type(X_train_pad[1]))
print(len(X_train_pad[1]))
```

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
 3396    7   34    1   47  442   58    2   63  992]
<class 'numpy.ndarray'>
500
```

5 Apply LSTM model

We will consider embedding vector length = 64

```
In [29]: # create the model
         embedding_vecor_length = 64
         model = Sequential()
         model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_leng
         model.add(LSTM(100))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         print(model.summary())
         #Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in
```

```
W0707 19:22:44.331145 140405479520128 deprecation_wrapper.py:119] From /usr/local/lib/python3.6

W0707 19:22:44.362855 140405479520128 deprecation_wrapper.py:119] From /usr/local/lib/python3.6

W0707 19:22:44.368639 140405479520128 deprecation_wrapper.py:119] From /usr/local/lib/python3.6

W0707 19:22:44.618789 140405479520128 deprecation_wrapper.py:119] From /usr/local/lib/python3.6

W0707 19:22:44.780548 140405479520128 deprecation_wrapper.py:119] From /usr/local/lib/python3.6

W0707 19:22:44.786787 140405479520128 deprecation.py:323] From /usr/local/lib/python3.6/dist-pa
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where


_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 500, 64)           320064    
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               66000     
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101       
=================================================================
Total params: 386,165
Trainable params: 386,165
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [30]: model.fit(X_train_pad, y_train, nb_epoch=10, batch_size=64)
         # Final evaluation of the model
         scores = model.evaluate(X_test_pad, y_test, verbose=0)
         print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
W0707 19:22:45.520249 140405479520128 deprecation_wrapper.py:119] From /usr/local/lib/python3.6
```

```
Epoch 1/10
32249/32249 [==============================] - 349s 11ms/step - loss: 0.2804 - acc: 0.8896
Epoch 2/10
32249/32249 [==============================] - 345s 11ms/step - loss: 0.1788 - acc: 0.9327
Epoch 3/10
32249/32249 [==============================] - 345s 11ms/step - loss: 0.1494 - acc: 0.9443
Epoch 4/10
32249/32249 [==============================] - 345s 11ms/step - loss: 0.1251 - acc: 0.9550
Epoch 5/10
32249/32249 [==============================] - 345s 11ms/step - loss: 0.1085 - acc: 0.9606
Epoch 6/10
32249/32249 [==============================] - 345s 11ms/step - loss: 0.0855 - acc: 0.9706
Epoch 7/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.0732 - acc: 0.9746
Epoch 8/10
32249/32249 [==============================] - 345s 11ms/step - loss: 0.0607 - acc: 0.9802
Epoch 9/10
32249/32249 [==============================] - 346s 11ms/step - loss: 0.0485 - acc: 0.9839
Epoch 10/10
32249/32249 [==============================] - 345s 11ms/step - loss: 0.0471 - acc: 0.9843
Accuracy: 90.39%
```

6 Apply LSTM Model 2

Here slightly modifying the model, we have considered embedding vector length is 32

```
In [31]: # create the model
         embedding_vecor_length = 32
         model = Sequential()
         model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_leng
         model.add(LSTM(100))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         print(model.summary())
         #Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in

         model.fit(X_train_pad, y_train, nb_epoch=10, batch_size=64)
         # Final evaluation of the model
         scores = model.evaluate(X_test_pad, y_test, verbose=0)
         print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           160032
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               53200
```

```
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,333
Trainable params: 213,333
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.2847 - acc: 0.8871
Epoch 2/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.1827 - acc: 0.9311
Epoch 3/10
32249/32249 [==============================] - 343s 11ms/step - loss: 0.1542 - acc: 0.9427
Epoch 4/10
32249/32249 [==============================] - 343s 11ms/step - loss: 0.1375 - acc: 0.9504
Epoch 5/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.1185 - acc: 0.9561
Epoch 6/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.1003 - acc: 0.9649
Epoch 7/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.0837 - acc: 0.9704
Epoch 8/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.0686 - acc: 0.9770
Epoch 9/10
32249/32249 [==============================] - 344s 11ms/step - loss: 0.0591 - acc: 0.9804
Epoch 10/10
32249/32249 [==============================] - 343s 11ms/step - loss: 0.0527 - acc: 0.9826
Accuracy: 90.18%
```

7 Apply Multiple LSTM model

Here we will try to slightly different model, we will test with stacking of LSTM model.

```
In [32]: # https://keras.io/getting-started/sequential-model-guide/

         embedding_vecor_length = 32

         # create the model
         model = Sequential()
         model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_leng
         model.add(LSTM(100, return_sequences=True))
         model.add(LSTM(100))

         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
print(model.summary())
#Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in

model.fit(X_train_pad, y_train, nb_epoch=10, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test_pad, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 500, 32)           160032    
_________________________________________________________________
lstm_3 (LSTM)                (None, 500, 100)          53200     
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               80400     
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101       
=================================================================
Total params: 293,733
Trainable params: 293,733
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
32249/32249 [==============================] - 690s 21ms/step - loss: 0.2745 - acc: 0.8921
Epoch 2/10
32249/32249 [==============================] - 688s 21ms/step - loss: 0.1793 - acc: 0.9316
Epoch 3/10
32249/32249 [==============================] - 689s 21ms/step - loss: 0.1553 - acc: 0.9429
Epoch 4/10
32249/32249 [==============================] - 688s 21ms/step - loss: 0.1281 - acc: 0.9542
Epoch 5/10
32249/32249 [==============================] - 688s 21ms/step - loss: 0.1039 - acc: 0.9647
Epoch 6/10
32249/32249 [==============================] - 685s 21ms/step - loss: 0.0833 - acc: 0.9722
Epoch 7/10
32249/32249 [==============================] - 688s 21ms/step - loss: 0.0674 - acc: 0.9779
Epoch 8/10
32249/32249 [==============================] - 691s 21ms/step - loss: 0.0548 - acc: 0.9828
Epoch 9/10
32249/32249 [==============================] - 698s 22ms/step - loss: 0.0419 - acc: 0.9873
Epoch 10/10
32249/32249 [==============================] - 698s 22ms/step - loss: 0.0342 - acc: 0.9897
Accuracy: 90.64%
```

8 LSTM Model with different parameter

Here we have considered single LSTM model, with various input parameter like Dropout, Batch Normalization

```
In [33]: # create the model
         embedding_vecor_length = 32
         model = Sequential()
         model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_leng
         model.add(Dropout(0.25))
         model.add(LSTM(100))
         model.add(BatchNormalization())
         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         print(model.summary())
         #Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in

         model.fit(X_train_pad, y_train, nb_epoch=10, batch_size=64)
         # Final evaluation of the model
         scores = model.evaluate(X_test_pad, y_test, verbose=0)
         print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
W0707 23:19:37.184394 140405479520128 deprecation.py:506] From /usr/local/lib/python3.6/dist-pa
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.


_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 500, 32)           160032
_________________________________________________________________
dropout_1 (Dropout)          (None, 500, 32)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 100)               53200
_________________________________________________________________
batch_normalization_1 (Batch (None, 100)               400
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,733
Trainable params: 213,533
Non-trainable params: 200

_________________________________________________________________
None
Epoch 1/10
32249/32249 [==============================] - 355s 11ms/step - loss: 0.2925 - acc: 0.8800
Epoch 2/10
32249/32249 [==============================] - 354s 11ms/step - loss: 0.1860 - acc: 0.9273
```

```
Epoch 3/10
32249/32249 [==============================] - 353s 11ms/step - loss: 0.1535 - acc: 0.9421
Epoch 4/10
32249/32249 [==============================] - 354s 11ms/step - loss: 0.1272 - acc: 0.9510
Epoch 5/10
32249/32249 [==============================] - 354s 11ms/step - loss: 0.1090 - acc: 0.9579
Epoch 6/10
32249/32249 [==============================] - 353s 11ms/step - loss: 0.0890 - acc: 0.9664
Epoch 7/10
32249/32249 [==============================] - 350s 11ms/step - loss: 0.0787 - acc: 0.9700
Epoch 8/10
32249/32249 [==============================] - 350s 11ms/step - loss: 0.0673 - acc: 0.9744
Epoch 9/10
32249/32249 [==============================] - 350s 11ms/step - loss: 0.0616 - acc: 0.9765
Epoch 10/10
32249/32249 [==============================] - 351s 11ms/step - loss: 0.0538 - acc: 0.9798
Accuracy: 88.66%
```

9 Conclusion

We can achieve better accuracy by stacking multiple LSTMs. With single LSTM accuracy was 90.18 and by stacking the LSTM model its slightly increase 90.67.