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
is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of d
review.

[1]. Reading Data

```
%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
```

```
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

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

```
%cd ./drive/'My Drive'
```

```
[Errno 2] No such file or directory: './drive/My Drive'
/content/drive/My Drive
```

[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

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purpose above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
# 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 points
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# 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 50000
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50000""

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative ra
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be negative 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)
```

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

```
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()
```

```
(80668, 7)
```

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

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

	UserId	ProductId	ProfileName	Time	Score	
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I w

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

```
393063
```

[2] Exploratory Data Analysis

[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 nece
results for the analysis of the data. Following is an example:

```
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	Helpfuln
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	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, Helpfu
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 s

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same pro
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
eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one
without sorting would lead to possibility of different representatives still existing for the same product.

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

Double-click (or enter) to edit

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

```
(46072, 10)
```

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

```
92.144
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than H
possible hence these two rows too are removed from calcualtions

```
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	Helpfu
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

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


#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)
1    38479
0     7592
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with ana

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
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

```
# 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 anym
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. ho
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein po
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definit
==================================================
```

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags
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 anym
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. ho
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein po
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definit
```

```
# 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


sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usually protein po
==================================================
```

```
#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 anym
```

```
#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 powder
```

```
# 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', 'hi
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            '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', 'over'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', '
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'd
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn
            'won', "won't", 'wouldn', "wouldn't"])
```

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

```
100%|██████████| 46071/46071 [00:20<00:00, 2214.11it/s]
```

```
preprocessed_reviews[1500]
```

```
'great flavor low calories high nutrients high protein usually protein powders high p
```

[3.2] Converting list of sentances to ranked list by top features

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews, final['Score'].v
```

```
print(type(X_train))
print(type(y_train))
print(type(X_test))
print(type(y_test))
print(len(X_train))
print(y_train.shape)
print(len(X_test))
print(y_test.shape)
```

```
<class 'list'>
<class 'numpy.ndarray'>
<class 'list'>
<class 'numpy.ndarray'>
32249
(32249,)
13822
(13822,)
```

```
#splitting the sentances
i=0
list_of_sentance=[]
for sentance in X_train:
    list_of_sentance.append(sentance.split())
```

```
#flattening the lists to just words
import itertools
X=list(itertools.chain.from_iterable(list_of_sentance))
#unique words
s=set(X)
```

```
s=list(s)
```

```
#finding the frequency of unique words
a=[]
for word in tqdm(s) :
  a.append(X.count(word))
```

```
100%|██████████| 33269/33269 [11:12<00:00, 49.44it/s]
```

```
#cross checking
print(len(X))
print(len(s))
print(len(a))
sum=0
for i in a:
  sum=sum+i
print(sum)
```

```
1257151
33269
33269
1257151
```

```
sorted=np.argsort(a)[::-1][:5000]

top_features=[]
for i in sorted :
  top_features.append(s[i])
```

```
#Train
def func(i):
  m=[]
  for j in i:
    if j in top_features :
       m.append(top_features.index(j)+1)
  return m

l=[]
for i in tqdm(list_of_sentance):
  m=func(i)
  l.append(m)

X_train=l
```

```
100%|██████████| 32249/32249 [00:42<00:00, 763.34it/s]
```

```
list_of_sentance=[]
for sentance in X_test:
    list_of_sentance.append(sentance.split())
```

```
#Test
def func(i):
  m=[]
  for j in i:
    if j in top_features :
       m.append(top_features.index(j)+1)
  return m

l=[]
for i in tqdm(list_of_sentance):
  m=func(i)
  l.append(m)


X_test=l
```

```
100%|██████████| 13822/13822 [00:18<00:00, 764.57it/s]
```

LSTM

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic_acc(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Accuracy")
    ax.plot(x, ty, 'r', label="Train Accuracy")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

```
# Credits: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neura
# LSTM for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
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
# fix random seed for reproducibility
numpy.random.seed(7)
```

```
# truncate and/or pad input sequences
max_review_length = 200
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(type(X_train))
print(X_train[1])
```

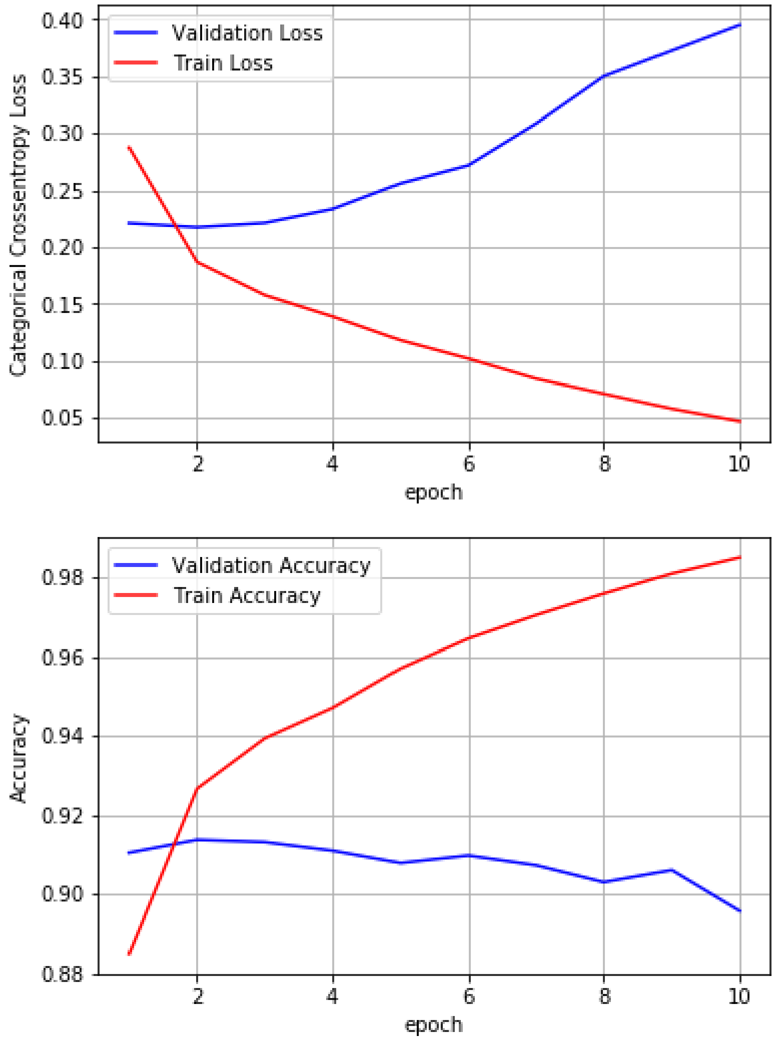
```
(32249, 200)
<class 'numpy.ndarray'>
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    2 2352  433  122    3
  667    1 1086   88]
```

```
# create the model
embedding_vector_length = 32
model = Sequential()
model.add(Embedding(5001, embedding_vector_length, input_length=max_review_length))
model.add(LSTM(256))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
history=model.fit(X_train, y_train, nb_epoch=10, batch_size=64,validation_data=(X_test,y_t
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_8 (Embedding)      (None, 200, 32)           160032    
_________________________________________________________________
lstm_10 (LSTM)               (None, 256)               295936    
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 257       
=================================================================
Total params: 456,225
Trainable params: 456,225
Non-trainable params: 0
_________________________________________________________________
None
Train on 32249 samples, validate on 13822 samples
Epoch 1/10
32249/32249 [==============================] - 226s 7ms/step - loss: 0.2873 - acc: 0.
Epoch 2/10
32249/32249 [==============================] - 225s 7ms/step - loss: 0.1867 - acc: 0.
Epoch 3/10
32249/32249 [==============================] - 224s 7ms/step - loss: 0.1576 - acc: 0.
Epoch 4/10
32249/32249 [==============================] - 224s 7ms/step - loss: 0.1388 - acc: 0.
Epoch 5/10
32249/32249 [==============================] - 225s 7ms/step - loss: 0.1179 - acc: 0.
Epoch 6/10
32249/32249 [==============================] - 225s 7ms/step - loss: 0.1019 - acc: 0.
Epoch 7/10
32249/32249 [==============================] - 224s 7ms/step - loss: 0.0843 - acc: 0.
Epoch 8/10
32249/32249 [==============================] - 224s 7ms/step - loss: 0.0709 - acc: 0.
Epoch 9/10
32249/32249 [==============================] - 225s 7ms/step - loss: 0.0578 - acc: 0.
Epoch 10/10
32249/32249 [==============================] - 225s 7ms/step - loss: 0.0469 - acc: 0.
Accuracy: 89.57%
```

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

# list of epoch numbers
x = list(range(1,11))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)

#accuracy
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Accuracy')
vy=history.history['val_acc']
ty=history.history['acc']
```

```
plt_dynamic_acc(x, vy, ty, ax)
```

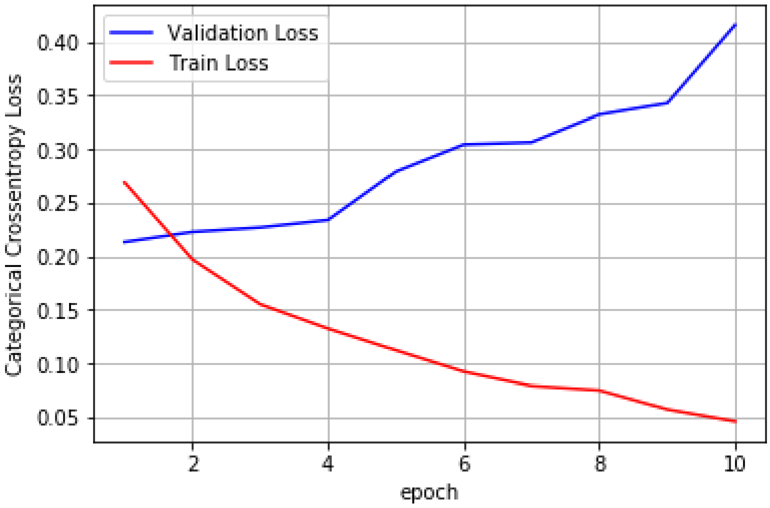


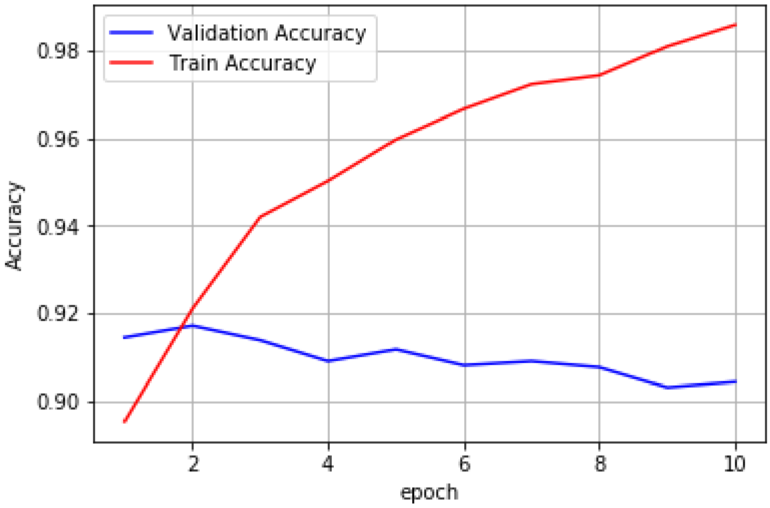
```
# create the model
embedding_vector_length = 64
model = Sequential()
model.add(Embedding(5001, embedding_vector_length, input_length=max_review_length))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())

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

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_9 (Embedding)      (None, 200, 64)           320064    
_________________________________________________________________
lstm_11 (LSTM)               (None, 128)               98816     
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129       
=================================================================
Total params: 419,009
Trainable params: 419,009
Non-trainable params: 0
_________________________________________________________________
None
Train on 32249 samples, validate on 13822 samples
Epoch 1/10
32249/32249 [==============================] - 230s 7ms/step - loss: 0.2689 - acc: 0.
Epoch 2/10
32249/32249 [==============================] - 227s 7ms/step - loss: 0.1970 - acc: 0.
Epoch 3/10
32249/32249 [==============================] - 226s 7ms/step - loss: 0.1552 - acc: 0.
Epoch 4/10
32249/32249 [==============================] - 227s 7ms/step - loss: 0.1325 - acc: 0.
Epoch 5/10
32249/32249 [==============================] - 225s 7ms/step - loss: 0.1122 - acc: 0.
Epoch 6/10
32249/32249 [==============================] - 224s 7ms/step - loss: 0.0924 - acc: 0.
Epoch 7/10
32249/32249 [==============================] - 225s 7ms/step - loss: 0.0791 - acc: 0.
Epoch 8/10
32249/32249 [==============================] - 224s 7ms/step - loss: 0.0750 - acc: 0.
Epoch 9/10
32249/32249 [==============================] - 222s 7ms/step - loss: 0.0572 - acc: 0.
Epoch 10/10
32249/32249 [==============================] - 223s 7ms/step - loss: 0.0462 - acc: 0.
Accuracy: 90.43%
```

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

# list of epoch numbers
x = list(range(1,11))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)

#accuracy
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Accuracy')
vy=history.history['val_acc']
ty=history.history['acc']
```

```
plt_dynamic_acc(x, vy, ty, ax)
```



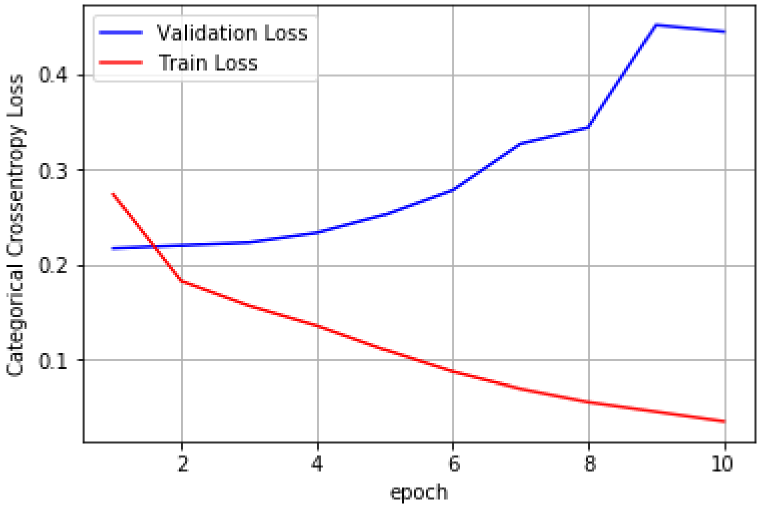


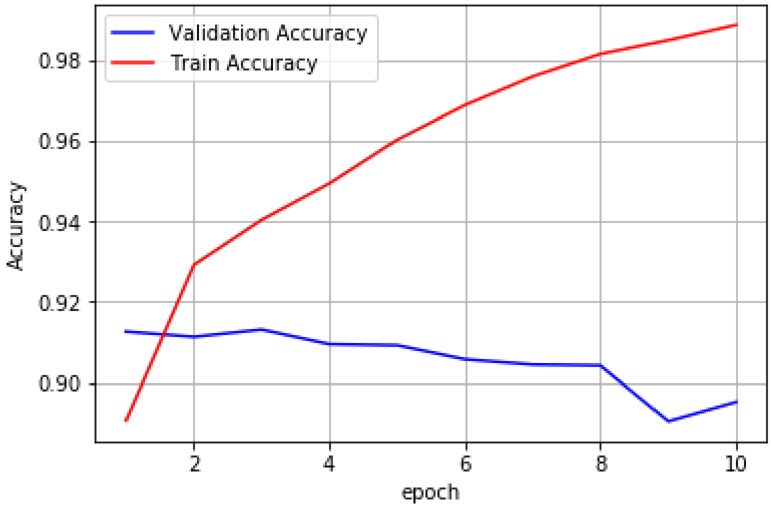
```
# create the model
embedding_vector_length = 32
model = Sequential()
model.add(Embedding(5001, embedding_vector_length, input_length=max_review_length),)
model.add(LSTM(128,return_sequences=True))
model.add(LSTM(64))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
history=model.fit(X_train, y_train, nb_epoch=10, batch_size=64,validation_data=(X_test,y_t
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_12 (Embedding)     (None, 200, 32)           160032    
_________________________________________________________________
lstm_14 (LSTM)               (None, 200, 128)          82432     
_________________________________________________________________
lstm_15 (LSTM)               (None, 64)                49408     
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65        
=================================================================
Total params: 291,937
Trainable params: 291,937
Non-trainable params: 0
_________________________________________________________________
None
Train on 32249 samples, validate on 13822 samples
Epoch 1/10
32249/32249 [==============================] - 454s 14ms/step - loss: 0.2740 - acc: 0
Epoch 2/10
32249/32249 [==============================] - 452s 14ms/step - loss: 0.1829 - acc: 0
Epoch 3/10
32249/32249 [==============================] - 461s 14ms/step - loss: 0.1570 - acc: 0
Epoch 4/10
32249/32249 [==============================] - 461s 14ms/step - loss: 0.1360 - acc: 0
Epoch 5/10
32249/32249 [==============================] - 459s 14ms/step - loss: 0.1107 - acc: 0
Epoch 6/10
32249/32249 [==============================] - 460s 14ms/step - loss: 0.0879 - acc: 0
Epoch 7/10
32249/32249 [==============================] - 462s 14ms/step - loss: 0.0699 - acc: 0
Epoch 8/10
32249/32249 [==============================] - 465s 14ms/step - loss: 0.0561 - acc: 0
Epoch 9/10
32249/32249 [==============================] - 464s 14ms/step - loss: 0.0460 - acc: 0
Epoch 10/10
32249/32249 [==============================] - 464s 14ms/step - loss: 0.0359 - acc: 0
Accuracy: 89.52%
```

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

# list of epoch numbers
x = list(range(1,11))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)

#accuracy
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Accuracy')
vy=history.history['val_acc']
```

```
vy=history.history['val_acc']
ty=history.history['acc']
plt_dynamic_acc(x, vy, ty, ax)
```



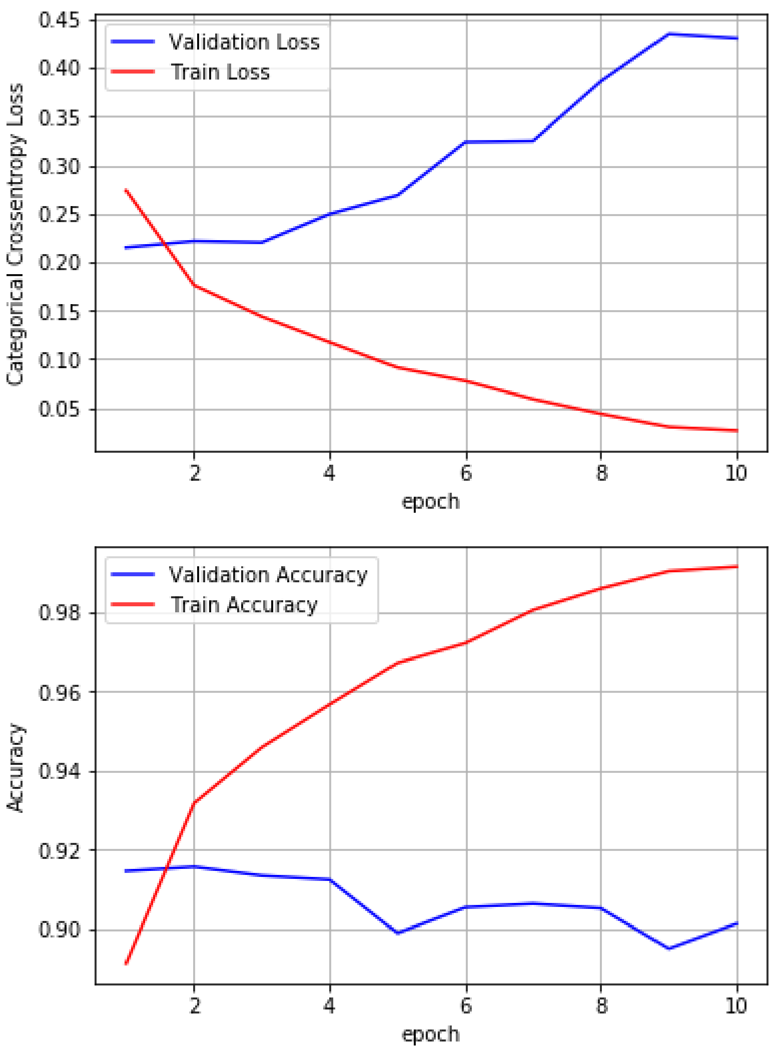


```
# create the model
embedding_vector_length = 64
model = Sequential()
model.add(Embedding(5001, embedding_vector_length, input_length=max_review_length))
model.add(LSTM(100,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
history=model.fit(X_train, y_train, nb_epoch=10, batch_size=64,validation_data=(X_test,y_t
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, 200, 64)           320064
_________________________________________________________________
lstm_16 (LSTM)               (None, 200, 100)          66000
_________________________________________________________________
lstm_17 (LSTM)               (None, 50)                30200
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 51
=================================================================
Total params: 416,315
Trainable params: 416,315
Non-trainable params: 0
_________________________________________________________________
None
Train on 32249 samples, validate on 13822 samples
Epoch 1/10
32249/32249 [==============================] - 460s 14ms/step - loss: 0.2737 - acc: 0
Epoch 2/10
32249/32249 [==============================] - 457s 14ms/step - loss: 0.1761 - acc: 0
Epoch 3/10
32249/32249 [==============================] - 459s 14ms/step - loss: 0.1439 - acc: 0
Epoch 4/10
32249/32249 [==============================] - 456s 14ms/step - loss: 0.1175 - acc: 0
Epoch 5/10
32249/32249 [==============================] - 456s 14ms/step - loss: 0.0915 - acc: 0
Epoch 6/10
32249/32249 [==============================] - 455s 14ms/step - loss: 0.0777 - acc: 0
Epoch 7/10
32249/32249 [==============================] - 454s 14ms/step - loss: 0.0591 - acc: 0
Epoch 8/10
32249/32249 [==============================] - 451s 14ms/step - loss: 0.0440 - acc: 0
Epoch 9/10
32249/32249 [==============================] - 452s 14ms/step - loss: 0.0306 - acc: 0
Epoch 10/10
32249/32249 [==============================] - 453s 14ms/step - loss: 0.0269 - acc: 0
Accuracy: 90.13%
```

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

# list of epoch numbers
x = list(range(1,11))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)

#accuracy
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Accuracy')
vy=history.history['val_acc']
```

```
vy=history.history['val_acc']
ty=history.history['acc']
plt_dynamic_acc(x, vy, ty, ax)
```



Summary

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Embedded_vector_length", "No of LSTM layers", "Accuracy"]

x.add_row([32, 1,89.57 ])
x.add_row([64, 1,90.43])
x.add_row([32, 2,89.52 ])
x.add_row([64, 2,90.13])

print(x)
```

```
+------------------------+-------------------+----------+
| Embedded_vector_length | No of LSTM layers | Accuracy |
+------------------------+-------------------+----------+
|           32           |         1         |  89.57   |
|           64           |         1         |  90.43   |
|           32           |         2         |  89.52   |
|           64           |         2         |  90.13   |
+------------------------+-------------------+----------+
```