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.

[1]. Reading Data

[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 [0]:

```
%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
```

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [3]:

```
# using SQLite Table to read data.
con = sqlite3.connect('./drive/My Drive/Copy of 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
# 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""", con)
# for tsne assignment you can take 5k data points

#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con
)
#filtered_data=pd.read_csv('Reviews.csv')

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
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[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

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 [5]:

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

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:

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

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [7]:

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

Out[7]:

```
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 necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [8]:

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

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE

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 [0]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [10]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

Out[10]:

```
(46072, 10)
```

In [11]:

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

Out[11]:

```
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 [12]:

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

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [0]:

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

In [14]:

```
#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[14]:

```
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 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 [15]:

```
# 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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is be
tter, but a heck of a lot more work. this is great to have on hand for last minute dessert needs w
here you really want to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig
h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo
r the lady gym rats, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give this
one a try. Let me first start by saying that everyone is looking for something different for their
ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range
of tea drinkers (whether you are a beginner or long-time tea enthusiast).  I have gone through ove
r 12 boxes of this tea myself, and highly recommend it for the following reasons:<br /><br />-Qual
ity:  First, this tea offers a smooth quality without any harsh or bitter after tones, which often
turns people off from many green teas.  I've found my ideal brewing time to be between 3-5
minutes, giving you a light but flavorful cup of tea.  However, if you get distracted or forget ab
out your tea and leave it brewing for 20+ minutes like I sometimes do, the quality of this tea is
such that you still get a smooth but deeper flavor without the bad after taste.  The leaves themse
lves are whole leaves (not powdered stems, branches, etc commonly found in other brands), and the
high-quality nylon bags also include chunks of tropical fruit and other discernible ingredients.
This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground d
own to a fine powder, leaving you to wonder what it is you are actually drinking.<br /><br />-Tast
e:  This tea offers notes of real pineapple and other hints of tropical fruits, yet isn't sweet or
artificially flavored.  You have the foundation of a high-quality young hyson green tea for those
true "tea flavor" lovers, yet the subtle hints of fruit make this a truly unique tea that I believ
e most will enjoy.  If you want it sweet, you can add sugar, splenda, etc but this really is not n
ecessary as this tea offers an inherent warmth of flavor through it's ingredients.<br /><br />-Pri
```

ce: This tea offers an excellent product at an exceptional price (especially when purchased at th
e prices Amazon offers). Compared to other brands which I believe to be of similar quality
(Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product at an outstanding pri
ce. I have been purchasing this through Amazon for less per box than I would be paying at my loca
l grocery store for Lipton, etc.

Overall, this is a wonderful tea that is comparable, a
nd even better than, other teas that are priced much higher. It offers a well-balanced cup of gre
en tea that I believe many will enjoy. In terms of taste, quality, and price, I would argue you w
on't find a better combination that that offered by Revolution's Tropical Green Tea.
==

In [16]:

```
# 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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt. It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.

In [17]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an
-element
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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt. It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is be
tter, but a heck of a lot more work. this is great to have on hand for last minute dessert needs w
here you really want to impress wih your creativity in cooking! recommended.
==
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig
h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo
r the lady gym rats, probably not "macho" enough for guys since it is soy based...
==
For those of you wanting a high-quality, yet affordable green tea, you should definitely give this
one a try. Let me first start by saying that everyone is looking for something different for their
ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range
of tea drinkers (whether you are a beginner or long-time tea enthusiast). I have gone through ove
r 12 boxes of this tea myself, and highly recommend it for the following reasons:-Quality: First,
this tea offers a smooth quality without any harsh or bitter after tones, which often turns people
off from many green teas. I've found my ideal brewing time to be between 3-5 minutes, giving you
a light but flavorful cup of tea. However, if you get distracted or forget about your tea and lea
ve it brewing for 20+ minutes like I sometimes do, the quality of this tea is such that you still
get a smooth but deeper flavor without the bad after taste. The leaves themselves are whole leave
s (not powdered stems, branches, etc commonly found in other brands), and the high quality nylon b

s (not powdered stems, branches, etc commonly found in other brands), and the high-quality nylon b
ags also include chunks of tropical fruit and other discernible ingredients. This isn't your
standard cheap paper bag with a mix of unknown ingredients that have been ground down to a fine po
wder, leaving you to wonder what it is you are actually drinking.-Taste: This tea offers notes of
real pineapple and other hints of tropical fruits, yet isn't sweet or artificially flavored. You
have the foundation of a high-quality young hyson green tea for those true "tea flavor" lovers, ye
t the subtle hints of fruit make this a truly unique tea that I believe most will enjoy. If you w
ant it sweet, you can add sugar, splenda, etc but this really is not necessary as this tea offers
an inherent warmth of flavor through it's ingredients.-Price: This tea offers an excellent
product at an exceptional price (especially when purchased at the prices Amazon offers). Compared
to other brands which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), R
evolution offers a superior product at an outstanding price. I have been purchasing this through
Amazon for less per box than I would be paying at my local grocery store for Lipton, etc.Overall,
this is a wonderful tea that is comparable, and even better than, other teas that are priced much
higher. It offers a well-balanced cup of green tea that I believe many will enjoy. In terms of t
aste, quality, and price, I would argue you won't find a better combination that that offered by R
evolution's Tropical Green Tea.

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 [19]:

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

Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig
h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo
r the lady gym rats, probably not "macho" enough for guys since it is soy based...
==

In [20]:

```
#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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt. It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.

In [21]:

```
#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 high pr
iced and high in calories this one is a great bargain and tastes great I highly recommend for the
lady gym rats probably not macho enough for guys since it is soy based

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', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [23]:

```
# 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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 46071/46071 [00:18<00:00, 2476.29it/s]
```

Applying LSTM using keras

In [24]:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection  import train_test_split
# fix random seed for reproducibility
np.random.seed(7)
```

```
Using TensorFlow backend.
```

In [25]:

```
#Splitting the data into train and test data
X=preprocessed_reviews
y=np.array(final['Score'])
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.4)
print("Number of words in train data:",len(X_tr))
print("Number of words in test data:",len(X_test))
```

```
Number of words in train data: 27642
Number of words in test data: 18429
```

In [26]:

```
#https://keras.io/preprocessing/text/
#turning each text into a sequence of integers (each integer being the index of a token in a dicti
onary)
#some help from AAIC members
tokenizer=Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_tr)
X_tr= tokenizer.texts_to_sequences(X_tr)
X_test=tokenizer.texts_to_sequences(X_test)
print(X_tr[0])
```

```
[386, 83, 4, 999, 2863, 1159]
```

Padding input sequences

In [27]:

```
max_review_length = 600
X_tr = sequence.pad_sequences(X_tr, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

print(X_tr.shape)
print(X_tr[1],y_tr[1])
```

```
(27642, 600)
[    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0   12    5  297    6 1646 3786    9 2711  292   27  190  582
  167 2462  190 1068  161  438 2711  272  332  143  326  745] 1
```

Creating 1 LSTM layer model

In [0]:

```
import matplotlib.pyplot as plt
import numpy as np

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()
```

In [29]:

```
# create the model
epoch=10
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(5001, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
```

```
WARNING: Logging before flag parsing goes to stderr.
W0828 15:04:36.117781 140209306556288 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:74: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0828 15:04:36.161686 140209306556288 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:517: The name
tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

W0828 15:04:36.169526 140209306556288 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4138: The name
tf.random_uniform is deprecated. Please use tf.random.uniform instead.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 600, 32)           160032
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,333
Trainable params: 213,333
Non-trainable params: 0
_________________________________________________________________
None
```

In [32]:

```
from keras.optimizers import Adam
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history=model.fit(X_tr, y_tr,batch_size=32,epochs=epoch,verbose=1,validation_data=(X_test, y_test))
```

```
Train on 27642 samples, validate on 18429 samples
Epoch 1/10
27642/27642 [==============================] - 600s 22ms/step - loss: 0.2675 - acc: 0.8918 - val_l
oss: 0.2200 - val_acc: 0.9121
Epoch 2/10
27642/27642 [==============================] - 604s 22ms/step - loss: 0.1768 - acc: 0.9335 - val_l
oss: 0.2302 - val_acc: 0.9120
Epoch 3/10
27642/27642 [==============================] - 599s 22ms/step - loss: 0.1511 - acc: 0.9444 - val_l
oss: 0.2526 - val_acc: 0.9044
Epoch 4/10
27642/27642 [==============================] - 597s 22ms/step - loss: 0.1309 - acc: 0.9511 - val_l
oss: 0.2387 - val_acc: 0.9073
Epoch 5/10
27642/27642 [==============================] - 593s 21ms/step - loss: 0.1096 - acc: 0.9613 - val_l
oss: 0.2767 - val_acc: 0.9073
Epoch 6/10
27642/27642 [==============================] - 586s 21ms/step - loss: 0.0866 - acc: 0.9697 - val_l
oss: 0.3399 - val_acc: 0.9055
Epoch 7/10
27642/27642 [==============================] - 587s 21ms/step - loss: 0.0675 - acc: 0.9772 - val_l
oss: 0.3316 - val_acc: 0.9066
Epoch 8/10
27642/27642 [==============================] - 586s 21ms/step - loss: 0.0542 - acc: 0.9823 - val_l
oss: 0.3592 - val_acc: 0.9029
Epoch 9/10
27642/27642 [==============================] - 585s 21ms/step - loss: 0.0527 - acc: 0.9826 - val_l
oss: 0.3971 - val_acc: 0.9025
Epoch 10/10
27642/27642 [==============================] - 584s 21ms/step - loss: 0.0466 - acc: 0.9853 - val_l
oss: 0.3256 - val_acc: 0.9005
```

In [33]:

```
score = model.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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


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

```
Test score: 0.3255654094095205
Test accuracy: 0.9005371968449537
```

Creating 2 LSTM layer model

In [0]:

```
# create the model
from keras.layers import Dropout
epoch=10
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(5001, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(200,return_sequences=True))
model.add(Dropout(0.25))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 600, 32)           160032
_________________________________________________________________
lstm_5 (LSTM)                (None, 600, 200)          186400
_________________________________________________________________
dropout_2 (Dropout)          (None, 600, 200)          0
_________________________________________________________________
lstm_6 (LSTM)                (None, 100)               120400
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101
=================================================================
Total params: 466,933
Trainable params: 466,933
Non-trainable params: 0
_________________________________________________________________
None
```

In [0]:

```
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history=model.fit(X_tr, y_tr,batch_size=64,epochs=epoch,verbose=1,validation_data=(X_test, y_test))
```

```
Train on 32249 samples, validate on 13822 samples
Epoch 1/10
32249/32249 [==============================] - 1830s 57ms/step - loss: 0.3389 - acc: 0.8697 - val_
loss: 0.4458 - val_acc: 0.8363
Epoch 2/10
32249/32249 [==============================] - 1866s 58ms/step - loss: 0.4485 - acc: 0.8347 - val_
loss: 0.4457 - val_acc: 0.8363
Epoch 3/10
32249/32249 [==============================] - 1874s 58ms/step - loss: 0.4486 - acc: 0.8347 - val_
loss: 0.4458 - val_acc: 0.8363
Epoch 4/10
32249/32249 [==============================] - 1874s 58ms/step - loss: 0.4485 - acc: 0.8347 - val_
loss: 0.4460 - val_acc: 0.8363
Epoch 5/10
32249/32249 [==============================] - 1874s 58ms/step - loss: 0.4485 - acc: 0.8347 - val_
loss: 0.4459 - val_acc: 0.8363
Epoch 6/10
32249/32249 [==============================] - 1875s 58ms/step - loss: 0.4484 - acc: 0.8347 - val_
loss: 0.4461 - val_acc: 0.8363
Epoch 7/10
32249/32249 [==============================] - 1887s 58ms/step - loss: 0.4485 - acc: 0.8347 - val_
loss: 0.4457 - val_acc: 0.8363
Epoch 8/10
32249/32249 [==============================] - 1858s 58ms/step - loss: 0.4485 - acc: 0.8347 - val_
loss: 0.4457 - val_acc: 0.8363
Epoch 9/10
32249/32249 [==============================] - 1811s 56ms/step - loss: 0.4485 - acc: 0.8347 - val_
loss: 0.4457 - val_acc: 0.8363
Epoch 10/10
32249/32249 [==============================] - 1806s 56ms/step - loss: 0.4485 - acc: 0.8347 - val_
loss: 0.4457 - val_acc: 0.8363
```

In [0]:

```
score = model.evaluate(X_test, y_test, verbose=0)
```

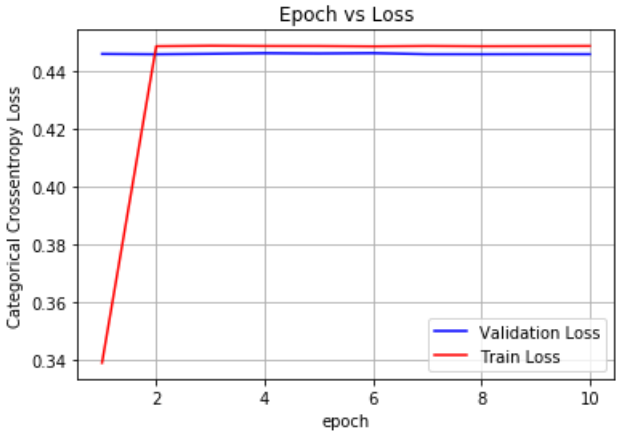
```
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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


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

```
Test score: 0.44570690078649794
Test accuracy: 0.8363478511998802
```



Observations and Conclusion

In [34]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Architecture", "Accuracy %"]

x.add_row(["1Lstm with batch size 32","90.05"])
x.add_row(["2Lstm with batch size 64","83.63"])

print(x)
```

```
+---------------------------+------------+
|       Architecture        | Accuracy % |
+---------------------------+------------+
| 1Lstm with batch size 32 |   90.05    |
| 2Lstm with batch size 64 |   83.63    |
+---------------------------+------------+
```

Our 1 LSTM layer model arranged:

Embedding-->LSTM-->Softmax

Our 2 LSTM layer model arranged in :

Embedding-->LSTM-->Dropout-->LSTM-->Softmax

First we extracted vocab for each words using tokenizer of keras then we splitted whole dataset into train and test and applied 1 layer LSTM and 2 layer LSTM on train data and then validated through test data to find our best accuracy.