Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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 [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
```

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

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 da
ta 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 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 L
IMIT 100000""", con)

# 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 (100000, 10)
```

Out[2]:

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

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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 [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

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

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

Out[6]:
```
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 [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inp
lace=False, kind='quicksort', na_position='last')
```

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

Out[9]:
```
(87775, 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]:
```
87.775
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulr
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

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

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)
#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(87773, 10)
```

Out[13]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

In [14]:
```
y=final['Score']
```

[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_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So
if you are afraid of the fishy smell, don't get it. But I think my dog likes
it because of the smell. These treats are really small in size. They are grea
t for training. You can give your dog several of these without worrying about
him over eating. Amazon's price was much more reasonable than any other retai
ler. You can buy a 1 pound bag on Amazon for almost the same price as a 6 oun
ce bag at other retailers. It's definitely worth it to buy a big bag if your
dog eats them a lot.
==================================================
```

In [16]:
```
# 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 [17]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [18]:
```
# 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', "you'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', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "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', 'becau
se', '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', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

In [20]:
```
# Combining all the above stundents
from tqdm import tqdm
from bs4 import BeautifulSoup
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 i
n stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:27<00:00, 3242.09it/s]
```

In [21]:
```
preprocessed_reviews[1500]
```

Out[21]:
```
'way hot blood took bite jig lol'
```

Applying LSTM

In [22]:
```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

In [23]:
```
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
# fix random seed for reproducibility
np.random.seed(1)
```

```
Using TensorFlow backend.
```

In [24]:
```
X=preprocessed_reviews
y=np.array(final['Score'])
```

Splitting dataset into train and test

In [25]:
```
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
print(len(X_train))
print(len(X_test))
```

```
52663
35110
```

In [26]:
```
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
print(len(X_train))
print(len(X_test))
```

```
52663
35110
```

In [27]:
```
#https://stackoverflow.com/questions/51956000/what-does-keras-tokenizer-method
-exactly-do
from keras.preprocessing.text import Tokenizer
token=Tokenizer(num_words=5000)
token.fit_on_texts(X_train)
X_train= token.texts_to_sequences(X_train)
X_test=token.texts_to_sequences(X_test)
print(X_train[0])
```

```
[16, 197, 11, 2282, 898, 163, 1344, 5, 6, 2867, 1, 31, 732, 264, 15, 17, 163,
49, 139, 2541, 214, 1432, 752]
```

Padding input sequences

In [28]:
```
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(X_train[1],y_train[1])
```

```
(52663, 200)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0  553  117 4834 3455  112 3087
   18   33  120  307  590 2573 1666  874   29 4550  111 4940  272    7
 4208  295  129   45] 0
```

Model 1 : 1 LSTM laver model

In [29]:
```
import matplotlib.pyplot as plt
import numpy as np
warnings.filterwarnings("ignore")
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 [31]:
```
# create the model
warnings.filterwarnings("ignore")

epoch=10
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(5000, embedding_vecor_length, input_length=max_review_leng
th))
model.add(LSTM(70))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
y'])
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 200, 32)           160000
_________________________________________________________________
lstm_2 (LSTM)                (None, 70)                28840
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 71
=================================================================
Total params: 188,911
Trainable params: 188,911
Non-trainable params: 0
_________________________________________________________________
None
```

In [32]:
```
from keras.optimizers import Adam
batch_size=32
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accurac
y'])
history=model.fit(X_train, y_train,batch_size=batch_size,epochs=epoch,verbose=
1,validation_data=(X_test, y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/
tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please use
tf.compat.v1.assign_add instead.

Train on 52663 samples, validate on 35110 samples
Epoch 1/10
52663/52663 [==============================] - 500s 9ms/step - loss: 0.2530 -
acc: 0.8994 - val_loss: 0.2072 - val_acc: 0.9184
Epoch 2/10
52663/52663 [==============================] - 498s 9ms/step - loss: 0.1837 -
acc: 0.9287 - val_loss: 0.2156 - val_acc: 0.9151
Epoch 3/10
52663/52663 [==============================] - 496s 9ms/step - loss: 0.1633 -
acc: 0.9368 - val_loss: 0.2030 - val_acc: 0.9189
Epoch 4/10
52663/52663 [==============================] - 496s 9ms/step - loss: 0.1392 -
acc: 0.9474 - val_loss: 0.2237 - val_acc: 0.9163
Epoch 5/10
52663/52663 [==============================] - 495s 9ms/step - loss: 0.1190 -
acc: 0.9565 - val_loss: 0.2189 - val_acc: 0.9162
Epoch 6/10
52663/52663 [==============================] - 495s 9ms/step - loss: 0.1019 -
acc: 0.9636 - val_loss: 0.2490 - val_acc: 0.9156
Epoch 7/10
52663/52663 [==============================] - 494s 9ms/step - loss: 0.0861 -
acc: 0.9696 - val_loss: 0.2726 - val_acc: 0.9135
Epoch 8/10
52663/52663 [==============================] - 493s 9ms/step - loss: 0.0748 -
acc: 0.9742 - val_loss: 0.2709 - val_acc: 0.9077
Epoch 9/10
52663/52663 [==============================] - 494s 9ms/step - loss: 0.0619 -
acc: 0.9790 - val_loss: 0.3219 - val_acc: 0.9081
Epoch 10/10
52663/52663 [==============================] - 493s 9ms/step - loss: 0.0504 -
acc: 0.9836 - val_loss: 0.3649 - val_acc: 0.9029
```

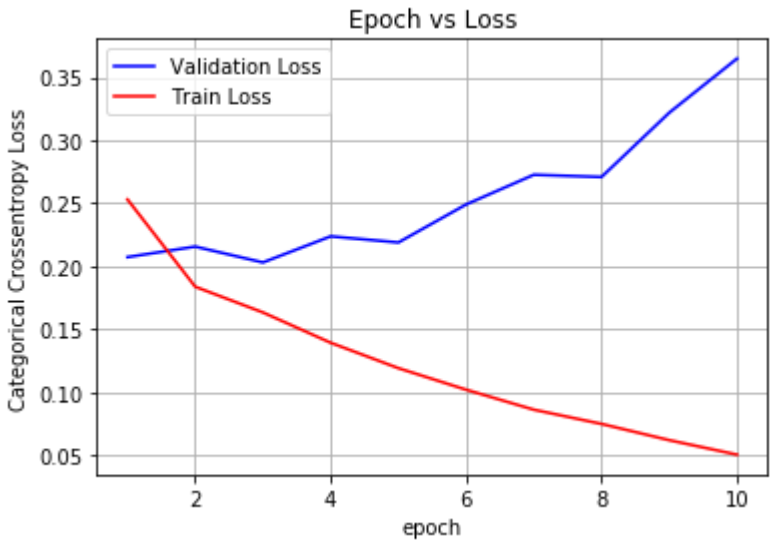
In [33]:
```
score1 = model.evaluate(X_test, y_test, verbose=0)
test1_s=score1[0]
test1_a=score1[1]
print('Test score:', score1[0])
print('Test accuracy:', score1[1])

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))


vy1 = history.history['val_loss']
ty1 = history.history['loss']
plt_dynamic(x, vy1, ty1, ax)
```

```
Test score: 0.3648570926431616
Test accuracy: 0.9029336371404159
```



Model2: 2 LSTM layer model (LSTM+Dropout+LSTM)

In [38]:
```
# create the model
from keras.layers import Dropout
epoch=10
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(5000, embedding_vecor_length, input_length=max_review_leng
th))
model.add(LSTM(70,return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(70))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
y'])
print(model.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 200, 32)           160000
_________________________________________________________________
lstm_7 (LSTM)                (None, 200, 70)           28840
_________________________________________________________________
dropout_3 (Dropout)          (None, 200, 70)           0
_________________________________________________________________
lstm_8 (LSTM)                (None, 70)                39480
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 71
=================================================================
Total params: 228,391
Trainable params: 228,391
Non-trainable params: 0
_________________________________________________________________
None
```

In [39]:
```
from keras.optimizers import Adam
batch_size=32
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accurac
y'])
history1=model.fit(X_train, y_train,batch_size=batch_size,epochs=epoch,verbose
=1,validation_data=(X_test, y_test))
```

```
Train on 52663 samples, validate on 35110 samples
Epoch 1/10
52663/52663 [==============================] - 978s 19ms/step - loss: 0.2515
- acc: 0.8993 - val_loss: 0.2083 - val_acc: 0.9173
Epoch 2/10
52663/52663 [==============================] - 976s 19ms/step - loss: 0.1827
- acc: 0.9291 - val_loss: 0.2065 - val_acc: 0.9201
Epoch 3/10
52663/52663 [==============================] - 975s 19ms/step - loss: 0.1582
- acc: 0.9390 - val_loss: 0.2088 - val_acc: 0.9212
Epoch 4/10
52663/52663 [==============================] - 973s 18ms/step - loss: 0.1346
- acc: 0.9481 - val_loss: 0.2248 - val_acc: 0.9210
Epoch 5/10
52663/52663 [==============================] - 975s 19ms/step - loss: 0.1148
- acc: 0.9578 - val_loss: 0.2228 - val_acc: 0.9173
Epoch 6/10
52663/52663 [==============================] - 972s 18ms/step - loss: 0.0935
- acc: 0.9672 - val_loss: 0.2490 - val_acc: 0.9111
Epoch 7/10
52663/52663 [==============================] - 972s 18ms/step - loss: 0.0784
- acc: 0.9734 - val_loss: 0.2777 - val_acc: 0.9176
Epoch 8/10
52663/52663 [==============================] - 970s 18ms/step - loss: 0.0684
- acc: 0.9773 - val_loss: 0.2938 - val_acc: 0.9091
Epoch 9/10
52663/52663 [==============================] - 972s 18ms/step - loss: 0.0563
- acc: 0.9811 - val_loss: 0.3217 - val_acc: 0.9095
Epoch 10/10
52663/52663 [==============================] - 971s 18ms/step - loss: 0.0462
- acc: 0.9850 - val_loss: 0.3144 - val_acc: 0.9146
```

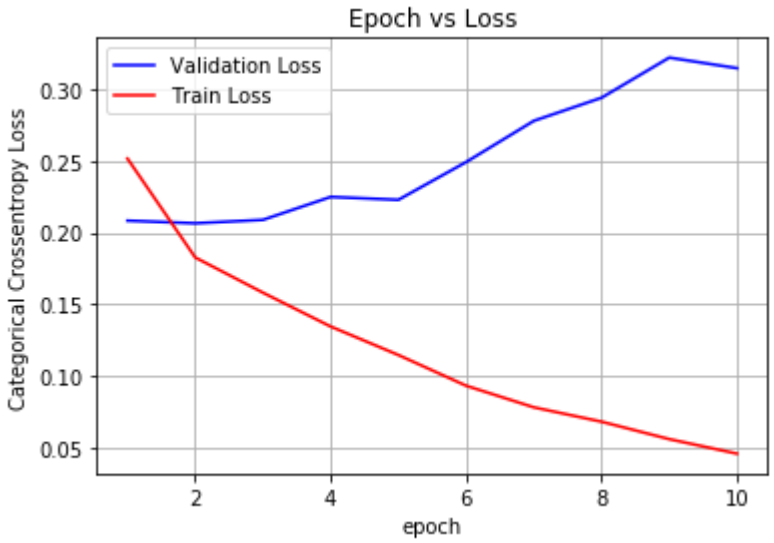
In [40]:
```
score2 = model.evaluate(X_test, y_test, verbose=0)
test2_s=score2[0]
test2_a=score2[1]
print('Test score:', score2[0])
print('Test accuracy:', score2[1])

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))


vy2 = history1.history['val_loss']
ty2 = history1.history['loss']
plt_dynamic(x, vy2, ty2, ax)
```

```
Test score: 0.3143605502811453
Test accuracy: 0.9145542580461407
```



Observations and Conclusion

In [3]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["No of LSTM layers", "Accuracy %","Test_Loss"]

x.add_row(["1","0.9029336371404159","0.3648570926431616"])
x.add_row(["2","0.9145542580461407","0.3143605502811453"])

#x.add_row(["1",test1_a,test1_s])
#x.add_row(["1",test2_a,test2_s])

print(x)
```

```
+------------------+--------------------+--------------------+
| No of LSTM layers |     Accuracy %     |     Test_Loss      |
+------------------+--------------------+--------------------+
|         1         | 0.9029336371404159 | 0.3648570926431616 |
|         2         | 0.9145542580461407 | 0.3143605502811453 |
+------------------+--------------------+--------------------+
```

Performance of 2 LSTM layered (with Dropout) is better in terms of both Accuracy and Loss.

Steps followed in this assignment:

Extraction of vocab for each word using tokenizer.

Splitting dataset into train and test in Ratio 60-40.

Application of 1 layer LSTM and 2 layer LSTM on train data and validation using test data to analyze model performance using Accuracy and Loss.

1 LSTM layer model: Embedding-->LSTM-->Softmax

2 LSTM layer model: Embedding-->LSTM-->Dropout-->LSTM-->Softmax