Amazon Fine food reviews dataset with LSTM modelling

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
```

[1]. Reading Data

In [0]:
```
#https://towardsdatascience.com/3-ways-to-load-csv-files-into-colab-7c14fcbdcb92
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

In [0]:
```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
downloaded = drive.CreateFile({'id':'1rdxqPjDNY1GaEprB7oN-b-HJ6_qKSlVE'}) # replace the id with id o
f file you want to access
downloaded.GetContentFile('database.sqlite') #https://drive.google.com/open?id=1rdxqPjDNY1GaEprB7oN-
b-HJ6_qKSlVE
```

In [7]:
```
# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data 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 LIMIT 100000""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
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[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	S
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	I
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	A
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	

In [14]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head(2)
```

Out[14]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600

In [0]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quic
ksort', na_position='last')
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpla
ce=False)
final.shape
import datetime
final['Time']=pd.to_datetime(final['Time'], unit='s')
final=final.sort_values(by='Time')
```

In [19]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[19]:
```
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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

#display.head()
```

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

[3]. Text Preprocessing.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [0]:
```
# 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 [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', "yo
u're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himse
lf', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'thes
e', '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', 'w
hile', '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', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "might
n't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "was
n't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [33]:
```
# 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%|██████████| 87773/87773 [00:33<00:00, 2589.48it/s]
```

In [34]:
```
preprocessed_reviews[:2]
```

Out[34]:
```
['bought apartment infested fruit flies hours trap attracted many flies within days practically
gone may not long term solution flies driving crazy consider buying one caution surface sticky t
ry avoid touching',
 'really good idea final product outstanding use decals car window everybody asks bought decals
made two thumbs']
```

In [0]:
```
#X=preprocessed_reviews
#Y=final['Score']
train=preprocessed_reviews[:70000]
test=preprocessed_reviews[70000:]
y_train=final['Score'][:70000]
y_test=final['Score'][70000:]
```

In [ ]:
```
'''def wordtofreq(wordlst):
    bow = CountVectorizer(lowercase= False, max_features=5000)
    bow_words = bow.fit_transform(wordlst)
    #print(bow_words.shape)
    freqs = bow_words.sum(axis=0).A1
    index = freqs.argsort()
    words = bow.get_feature_names()
    return dict(list(zip(words,freqs)))

def rankingWords(sentence,dictionary):
    ranked_sent=list()
    for sent in sentence:
        d=list()
        for i in range(len(sent.split())):
            w =sent.split()[i]
            if w in dictionary.keys():
                r =dictionary[w]
                d.append(r)
        ranked_sent.append(d)
    return ranked_sent

def converting_text_to_rankednumericals(text):
  review_count=list()
  for i in text.values:
      review_count.append(i.split())

  words_list=[item for review_count in review_count for item in review_count]

  word_dict=wordtofreq(words_list)

  #https://stackoverflow.com/questions/40488532/python-sorting-the-values-of-a-dict-and-extracting-t
he-keys-corresponding-to-th
  ranked_word_dict={key: rank for rank, key in enumerate(sorted(word_dict, key=word_dict.get,reverse
=True), 1)}

  #maping each and every review into ranked listed review
  ranked_datareviews=np.array(rankingWords(text.values,ranked_word_dict))
  return ranked_datareviews'''
```

Observation:

- since the above code cannot give the top words based on the train data we shall discard above algorithm.
- On tokenizing the each word of text, we can use the predefined keras library for this(like below).

In [37]:
```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(train)

X_train = tokenizer.texts_to_sequences(train)
X_test =tokenizer.texts_to_sequences(test)
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

In [38]:
```
print(X_train[1])
print(type(X_train[1]))
print(len(X_train[1]))
```

```
[16, 3, 523, 2483, 8, 1458, 22, 1181, 4336, 2750, 3643, 48, 46, 50, 2263]
<class 'list'>
15
```

In [39]:
```
r=list()
for i in preprocessed_reviews:
    r.append(len(i.split()))

max(r)  # this is the maximum length of words of all the preprocessed reviews.
```

Out[39]:
```
1596
```

In [0]:
```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
```

In [41]:
```
# truncate and/or pad input sequences
max_review_length = 1596
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])
```

```
(70000, 1596)
[    0     0     0 ...    46    50 2263]
```

With 1-LSTM layer model

In [0]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time

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()
```

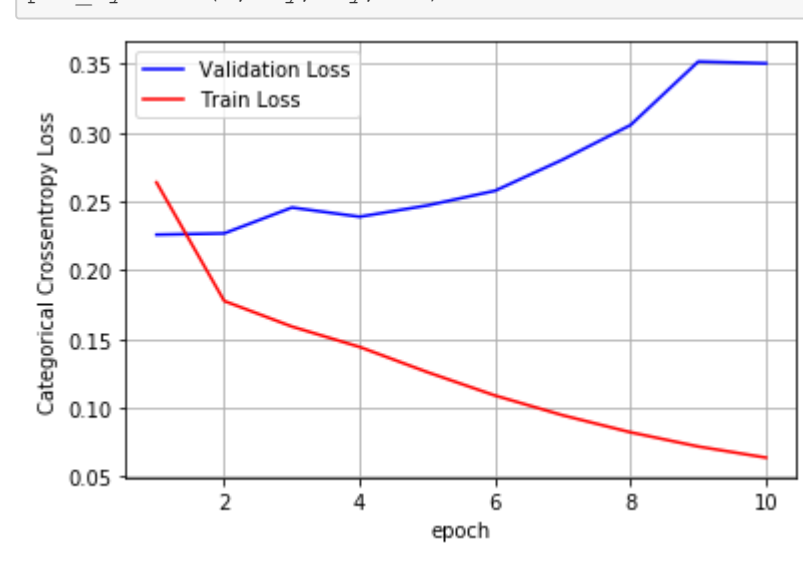
In [45]:
```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(5000+1, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())

history = model.fit(X_train, y_train, nb_epoch=10, batch_size=64,validation_split=0.33)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1596, 32)          160032
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,333
Trainable params: 213,333
Non-trainable params: 0
_________________________________________________________________
None
Train on 46899 samples, validate on 23101 samples
Epoch 1/10
46899/46899 [==============================] - 2115s 45ms/step - loss: 0.2638 - acc: 0.8949 - va
l_loss: 0.2259 - val_acc: 0.9083
Epoch 2/10
46899/46899 [==============================] - 2084s 44ms/step - loss: 0.1776 - acc: 0.9306 - va
l_loss: 0.2268 - val_acc: 0.9100
Epoch 3/10
46899/46899 [==============================] - 2055s 44ms/step - loss: 0.1590 - acc: 0.9391 - va
l_loss: 0.2456 - val_acc: 0.9074
Epoch 4/10
46899/46899 [==============================] - 2169s 46ms/step - loss: 0.1442 - acc: 0.9445 - va
l_loss: 0.2389 - val_acc: 0.9050
Epoch 5/10
46899/46899 [==============================] - 2215s 47ms/step - loss: 0.1259 - acc: 0.9526 - va
l_loss: 0.2471 - val_acc: 0.9063
Epoch 6/10
46899/46899 [==============================] - 2134s 46ms/step - loss: 0.1089 - acc: 0.9595 - va
l_loss: 0.2578 - val_acc: 0.9052
Epoch 7/10
46899/46899 [==============================] - 2053s 44ms/step - loss: 0.0945 - acc: 0.9661 - va
l_loss: 0.2805 - val_acc: 0.9039
Epoch 8/10
46899/46899 [==============================] - 2035s 43ms/step - loss: 0.0822 - acc: 0.9709 - va
l_loss: 0.3055 - val_acc: 0.9033
Epoch 9/10
46899/46899 [==============================] - 2083s 44ms/step - loss: 0.0718 - acc: 0.9756 - va
l_loss: 0.3517 - val_acc: 0.8965
Epoch 10/10
46899/46899 [==============================] - 2072s 44ms/step - loss: 0.0637 - acc: 0.9780 - va
l_loss: 0.3504 - val_acc: 0.8996
Accuracy: 90.45%
```

In [46]:
```
nb_epoch=10
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```



With 2-LSTM layers model

In [43]:
```
embedding_vecor_length = 32
model1 = Sequential()
model1.add(Embedding(5000+1, embedding_vecor_length, input_length=max_review_length))
model1.add(LSTM(100, return_sequences=True))
model1.add(LSTM(100, return_sequences=False))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model1.summary())

history = model1.fit(X_train, y_train, nb_epoch=5, batch_size=64, validation_split=0.33)
# Final evaluation of the model
scores1 = model1.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores1[1]*100))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:66: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph in
stead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:541: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:4432: The name tf.random_uniform is deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name
tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:3657: The name tf.log is deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/nn_imp
l.py:183: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1596, 32)          160032
_________________________________________________________________
lstm_1 (LSTM)                (None, 1596, 100)         53200
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 293,733
Trainable params: 293,733
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:1033: The name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:1020: The name tf.assign is deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:3005: The name tf.Session is deprecated. Please use tf.compat.v1.Session instead.

Train on 46899 samples, validate on 23101 samples
Epoch 1/5
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:190: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_sessi
on instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:197: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:207: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables ins
tead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:216: The name tf.is_variable_initialized is deprecated. Please use tf.compat.v1.is_variable_i
nitialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.
py:223: The name tf.variables_initializer is deprecated. Please use tf.compat.v1.variables_initi
alizer instead.

46899/46899 [==============================] - 4051s 86ms/step - loss: 0.2546 - acc: 0.9015 - va
l_loss: 0.2357 - val_acc: 0.9011
Epoch 2/5
46899/46899 [==============================] - 4010s 85ms/step - loss: 0.1784 - acc: 0.9314 - va
l_loss: 0.2205 - val_acc: 0.9100
Epoch 3/5
46899/46899 [==============================] - 3989s 85ms/step - loss: 0.1533 - acc: 0.9415 - va
l_loss: 0.2287 - val_acc: 0.9099
Epoch 4/5
46899/46899 [==============================] - 3992s 85ms/step - loss: 0.1320 - acc: 0.9507 - va
l_loss: 0.2612 - val_acc: 0.9055
Epoch 5/5
46899/46899 [==============================] - 3998s 85ms/step - loss: 0.1113 - acc: 0.9596 - va
l_loss: 0.2700 - val_acc: 0.9071
Accuracy: 91.32%
```

In [47]:
```
nb_epoch=5
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```