Amazon Fine Food reviews LSTM model

In [2]:

```
# Credits: https://machinelearningmastery.com/sequence-classification-l
stm-recurrent-neural-networks-python-keras/
# LSTM for sequence classification in the IMDB dataset

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
```

In [3]:

```
#to ignore warnings
import warnings
warnings.filterwarnings("ignore")
#to use sqlite3 database
import sqlite3
import numpy as np
import pandas as pd
import string
import nltk
import matplotlib.pyplot as plt

from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\cross_validation.py:
41: DeprecationWarning: This module was deprecated in version 0.18 in f
```

```
avor of the model_selection module into which all the refactored classe
s and functions are moved. Also note that the interface of the new CV i
terators are different from that of this module. This module will be re
moved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

data preprocessing, data cleaning, data deduplication

We are not removing stopwords here , so that our model will have good accuracy

In [5]:

```
con = sqlite3.connect('database.sqlite')

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 """, 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
```

In [6]:

```
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball s
temmer
def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation
```

```
 or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [7]:

```
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in filtered_data['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):

                s=(sno.stem(cleaned_words.lower())).encode('utf8')
                filtered_sentence.append(s)
                if (filtered_data['Score'].values)[i] == 'positive':
                    all_positive_words.append(s) #list of all words use
d to describe positive reviews
                if(filtered_data['Score'].values)[i] == 'negative':
                    all_negative_words.append(s) #list of all words use
d to describe negative reviews reviews

            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************
************")

    final_string.append(str1)
    i+=1
```

In [8]:

```
filtered_data['CleanedText']=final_string #adding a column of CleanedText which displays the data after pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("utf-8")
```

In [9]:

```
sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(50000)

S = sorted_data['Score']
Score = S.head(50000)
```

Splitting the data

In [10]:

```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(sampledata, Score, test_size=0.4, random_state=0)
```

converting negative to 0 and positive to 1

converting all the words in training dataset in to list of words

In [11]:

```
comment_words = []
for val in X_train['CleanedText'].values:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        comment_words.append(tokens[i])
```

particular word and frequency in different lists and zip them and sort and take top 5000 ranked words...

In [12]:

```
words = []
freq =[]

for word in comment_words:
    frequ =0
    if word not in words:
        words.append(word)
        for word1 in comment_words:
            if word1 == word:
                frequ+=1
        freq.append(frequ)
```

In [13]:

```
rank= sorted(zip(freq,words),reverse=True)[0:5000]
top_words=[]
for ran, word in rank:
    top_words.append(word)
```

In [14]:

```
subl = []
maindata = []
for val in X_train['CleanedText'].values:
    subl = []
    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        if tokens[i] in top_words:
            subl.append(top_words.index(tokens[i])+1)
    maindata.append(subl)
```

In [15]:

```
maindataxtr=np.asarray(maindata)
```

In [16]:

```
subl = []
maindata = []
for val in X_test['CleanedText'].values:
    subl = []
    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        if tokens[i] in top_words:
            subl.append(top_words.index(tokens[i])+1)
    maindata.append(subl)
```

In [17]:

```
maindataxtest = np.asarray(maindata)
```

padding them to 600

In [18]:

```
max_review_length = 600
maindataxtr = sequence.pad_sequences(maindataxtr, maxlen=max_review_len
gth)
maindataxtest = sequence.pad_sequences(maindataxtest, maxlen=max_review
_length)
```

In [19]:

```
print("training data",maindataxtr)
print("training data",maindataxtest)
```

```
training data [[   0    0    0 ...    6  596  312]
```

```
 [   0    0    0 ...  103 1330  269]
 [   0    0    0 ...  449 1194 2195]
 ...
 [   0    0    0 ...  793   37   87]
 [   0    0    0 ...  355 1181  280]
 [   0    0    0 ...  153    2 1012]]
training data [[   0    0    0 ...   18  190  488]
 [   0    0    0 ...  153  325  120]
 [   0    0    0 ... 1021    3  206]
 ...
 [   0    0    0 ...  102   22 1391]
 [   0    0    0 ...    6  124  478]
 [   0    0    0 ...  253   86  212]]
```

Single LSTM

In [30]:

```
embedding_vecor_length = 32
model1 = Sequential()
model1.add(Embedding(5001, embedding_vecor_length, input_length=max_review_length))
model1.add(LSTM(100))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model1.summary())
#Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in-an-lstm-model
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_2 (Embedding)      (None, 600, 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
```

In [31]:

```
history=model1.fit(maindataxtr, y_train, nb_epoch=10, batch_size=64,val
idation_data=(maindataxtest, y_test))
# Final evaluation of the model
scores = model1.evaluate(maindataxtest, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

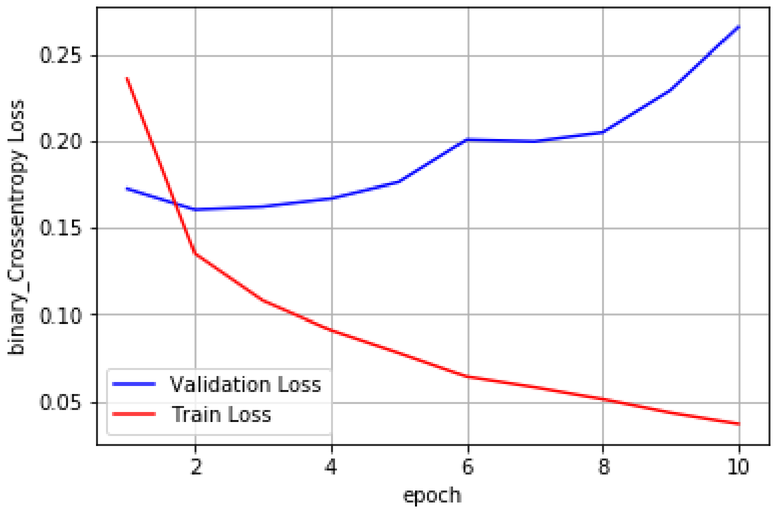
```
Train on 30000 samples, validate on 20000 samples
Epoch 1/10
30000/30000 [==============================] - 337s 11ms/step - loss:
0.2358 - acc: 0.9162 - val_loss: 0.1725 - val_acc: 0.9386
Epoch 2/10
30000/30000 [==============================] - 329s 11ms/step - loss:
0.1349 - acc: 0.9533 - val_loss: 0.1605 - val_acc: 0.9446
Epoch 3/10
30000/30000 [==============================] - 331s 11ms/step - loss:
0.1079 - acc: 0.9639 - val_loss: 0.1622 - val_acc: 0.9442
Epoch 4/10
30000/30000 [==============================] - 332s 11ms/step - loss:
0.0909 - acc: 0.9705 - val_loss: 0.1669 - val_acc: 0.9454
Epoch 5/10
30000/30000 [==============================] - 332s 11ms/step - loss:
0.0778 - acc: 0.9750 - val_loss: 0.1762 - val_acc: 0.9459
Epoch 6/10
30000/30000 [==============================] - 331s 11ms/step - loss:
0.0641 - acc: 0.9801 - val_loss: 0.2007 - val_acc: 0.9413
Epoch 7/10
30000/30000 [==============================] - 345s 12ms/step - loss:
0.0580 - acc: 0.9829 - val_loss: 0.1996 - val_acc: 0.9384
Epoch 8/10
30000/30000 [==============================] - 356s 12ms/step - loss:
0.0511 - acc: 0.9851 - val_loss: 0.2049 - val_acc: 0.9474
Epoch 9/10
30000/30000 [==============================] - 343s 11ms/step - loss:
```

```
0.0433 - acc: 0.9873 - val_loss: 0.2292 - val_acc: 0.9469
Epoch 10/10
30000/30000 [==============================] - 340s 11ms/step - loss:
0.0368 - acc: 0.9892 - val_loss: 0.2656 - val_acc: 0.9445
Accuracy: 94.45%
```

In [33]:

```
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()
score = model1.evaluate(maindataxtest, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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


x = list(range(1,11))


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

```
Test score: 0.26559420577352866
Test accuracy: 0.9445
```



Multiple LSTM

In [42]:

```
from keras.layers import Dropout
from keras.layers import BatchNormalization
embedding_vecor_length = 32
model2 = Sequential()
model2.add(Embedding(5001, embedding_vecor_length, input_length=max_rev
iew_length))
model2.add(LSTM(100,return_sequences=True))
model2.add(BatchNormalization())
model2.add(Dropout(0.25))

model2.add(LSTM(50))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=[
'accuracy'])
print(model2.summary())
#Refer: https://datascience.stackexchange.com/questions/10615/number-of
-parameters-in-an-lstm-model
```

```
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_7 (Embedding)      (None, 600, 32)           160032    
_________________________________________________________________
lstm_12 (LSTM)               (None, 600, 100)          53200     
_________________________________________________________________
batch_normalization_5 (Batch (None, 600, 100)          400       
_________________________________________________________________
dropout_5 (Dropout)          (None, 600, 100)          0         
_________________________________________________________________
lstm_13 (LSTM)               (None, 50)                30200     
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 51        
=================================================================
Total params: 243,883
Trainable params: 243,683
Non-trainable params: 200
_________________________________________________________________
None
```

In [43]:

```
history=model2.fit(maindataxtr, y_train, nb_epoch=10, batch_size=64,validation_data=(maindataxtest, y_test))
# Final evaluation of the model
scores = model2.evaluate(maindataxtest, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

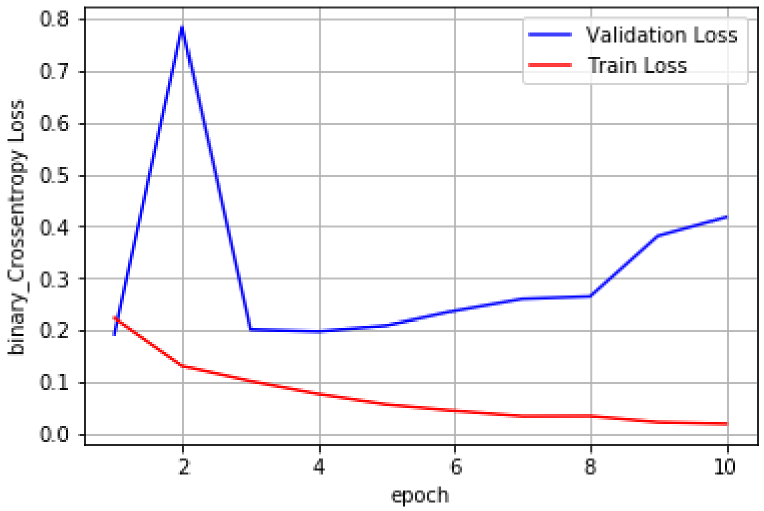
```
Train on 30000 samples, validate on 20000 samples
Epoch 1/10
30000/30000 [==============================] - 819s 27ms/step - loss: 0.2231 - acc: 0.9180 - val_loss: 0.1923 - val_acc: 0.9194
Epoch 2/10
30000/30000 [==============================] - 754s 25ms/step - loss: 0.1309 - acc: 0.9534 - val_loss: 0.7833 - val_acc: 0.5566
Epoch 3/10
30000/30000 [==============================] - 756s 25ms/step - loss: 0.1016 - acc: 0.9647 - val_loss: 0.2012 - val_acc: 0.9273
Epoch 4/10
30000/30000 [==============================] - 757s 25ms/step - loss: 0.0768 - acc: 0.9751 - val_loss: 0.1975 - val_acc: 0.9339
```

```
Epoch 5/10
30000/30000 [==============================] - 849s 28ms/step - loss:
0.0565 - acc: 0.9822 - val_loss: 0.2085 - val_acc: 0.9294
Epoch 6/10
30000/30000 [==============================] - 794s 26ms/step - loss:
0.0442 - acc: 0.9864 - val_loss: 0.2366 - val_acc: 0.9320
Epoch 7/10
30000/30000 [==============================] - 770s 26ms/step - loss:
0.0338 - acc: 0.9896 - val_loss: 0.2597 - val_acc: 0.9463
Epoch 8/10
30000/30000 [==============================] - 862s 29ms/step - loss:
0.0341 - acc: 0.9887 - val_loss: 0.2644 - val_acc: 0.9227
Epoch 9/10
30000/30000 [==============================] - 850s 28ms/step - loss:
0.0222 - acc: 0.9930 - val_loss: 0.3813 - val_acc: 0.8998
Epoch 10/10
30000/30000 [==============================] - 875s 29ms/step - loss:
0.0190 - acc: 0.9937 - val_loss: 0.4177 - val_acc: 0.9026
Accuracy: 90.25%
```

In [44]:

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

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


x = list(range(1,11))


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

```
Test score: 0.41765666490010916
Test accuracy: 0.90255
```



In [45]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["lstm layers","Test accuracy","no_of_epoch"]

x.add_row(["1","94.45","10"])
x.add_row(["2","90.2","10"])



print(x)
```

```
+-------------+---------------+-------------+
| lstm layers | Test accuracy | no_of_epoch |
+-------------+---------------+-------------+
|      1      |     94.45     |      10     |
|      2      |      90.2     |      10     |
+-------------+---------------+-------------+
```