51.11.Amazon_food_review_LSTM_50k

August 26, 2018

1 Amazon food review dataset apply LSTM

Data set from https://www.kaggle.com/snap/amazon-fine-food-reviews

2 Objective

1. Create vocabulary, get frequancy then index data convert data into imdb dataset format
2. Run LSTM and report accuracy for 10 epoch, try 2 layer of lstm(add one more layer with imdb)

3 Import data and libraries

```
In [1]: import warnings
        warnings.filterwarnings('ignore')
        from sklearn.manifold import TSNE
        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
        from sklearn.cross_validation import train_test_split,KFold
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.cross_validation import cross_val_score
        from collections import Counter
        from sklearn.metrics import accuracy_score
        from sklearn import cross_validation
```

```
from sklearn.grid_search import GridSearchCV
from sklearn.linear_model import LogisticRegression
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
np.random.seed(7)
con = sqlite3.connect('database.sqlite')
#get only +ve and -ve review
raw_data = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3""", con)
```

```
C:\Users\suman\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning:
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\suman\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: DeprecationWarning: This
  DeprecationWarning)
Using TensorFlow backend.
```

4 Data preprocessing

In [2]:
```
filtered_data=raw_data
# Score>3 a positive rating, and score<3 a negative rating.
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

#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

filtered_data.sample(5)
filtered_data['Score'].value_counts()

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

#Deduplication of entries for same profilename,userid, time, text and take first eleme
sorted_data=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},
```

In [3]:
```
#take only 20000 data
print('total data \n',sorted_data['Score'].value_counts())
#take stratified sampling i.e. positive and negative reviews are proportionate to raw
_ , clean_data = train_test_split(sorted_data, test_size = 50000, random_state=0,strati
```

```
clean_data['Score'].value_counts()
topitem=5000
nb_epoch=6
```

```
total data
 positive    307063
negative     57110
Name: Score, dtype: int64
```

In [4]:
```
# Clean html tag and punctuation
import warnings
warnings.filterwarnings('ignore')
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

#substitute html tag and punctuation
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 cha
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
#print(sno.stem('tasty'))

i=0
str1=' '
mystop={'of','four','one','would'}
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
#Create new catagory as Cleanedtext after removing htmltag and punctuation and upperca
for sent in clean_data['Text'].values:
    #change later
    #sent=sent[:20]
    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)):
                        if((cleaned_words.lower() not in stop) & (cleaned_words.lower() not in
                            s=(sno.stem(cleaned_words.lower())).encode('utf8')
                            filtered_sentence.append(s)
                            if (clean_data['Score'].values)[i] == 'positive':
                                all_positive_words.append(s) #list of all words used to descri
                            if(clean_data['Score'].values)[i] == 'negative':
                                all_negative_words.append(s) #list of all words used to descri
                        else:
                            continue
                    else:
                        continue
            str1 = b" ".join(filtered_sentence) #final string of cleaned words

            final_string.append(str1)
            i+=1

        clean_data['CleanedText']=final_string
        print(clean_data.shape)
        #Sort data on timestamp
        clean_data=clean_data.sort_values(by=['Time'],ascending=False)
        #clean_data
        clean_data['CleanedText'].sample(2)
```

```
(50000, 11)
```

```
Out[4]: 20841    b'amazon indic normal price lbs dog food sell ...
        98150    b'wonder get amaz beverag like get local donut...
        Name: CleanedText, dtype: object
```

5 Split train and test

```
In [5]: x=clean_data['CleanedText'].values
        y = clean_data['Score']
        n=x.shape[0]
        n1=int(n*.3)
        X_test_raw = x[0:n1]
        X_train_raw= x[n1:n+1]
        y_test=y[0:n1]
        y_train=y[n1:n+1]

        print('size of X_train, X_test, y_train , y_test ',X_train_raw.shape, X_test_raw.shape
        print("positive and negative review in train and test\n",y_train.value_counts(),"\n",y

        from sklearn.preprocessing import label_binarize
        encoded_column_vector = label_binarize(y_train, classes=['negative','positive']) # neg
```

```
        encoded_labels = np.ravel(encoded_column_vector) # Reshape array
        y_train=encoded_labels

        encoded_column_vector = label_binarize(y_test, classes=['negative','positive']) # nega
        encoded_labels = np.ravel(encoded_column_vector) # Reshape array
        y_test=encoded_labels
```

```
size of X_train, X_test, y_train , y_test  (35000,) (15000,) (35000,) (15000,)
positive and negative review in train and test
 positive    29732
negative     5268
Name: Score, dtype: int64
 positive    12427
negative     2573
Name: Score, dtype: int64
```

5.1 Create dictionary of words

First create dict with word frequency then sort descending

```
In [6]: # Form dictionary from train as word and freq
        from collections import defaultdict
        fq= defaultdict( int )
        for sent in X_train_raw:
          for w in sent.split():
            fq[w] += 1

        #Sort dictionary on descending of freq
        from collections import OrderedDict
        from operator import itemgetter
        sorteddict=OrderedDict(sorted(fq.items(), key = itemgetter(1), reverse = True))

        import collections
        #Take top items and sort again
        sorteddictnew=dict(collections.Counter(sorteddict).most_common(topitem))
        sorteddictnew=OrderedDict(sorted(sorteddictnew.items(), key = itemgetter(1), reverse =

        #change values of freq top with 1 then 2 ,3,4 like that
        for i, k in enumerate(sorteddictnew):
            sorteddictnew[k]=i+1
```

5.2 Replace train and test words with the rank from dictionary

```
In [7]: #replace each word with its index form dict
        final_string=[]
        for sent in X_train_raw:
            filtered_sentence=[]
```

```
            for w in sent.split():
               try:
                 s=list(sorteddictnew.keys()).index(w)
                 filtered_sentence.append(s)
               except:
                 continue

            final_string.append(filtered_sentence)
        X_train_new=final_string


        #replace each word with its index form dict
        final_string=[]
        for sent in X_test_raw:
            filtered_sentence=[]
            for w in sent.split():
               try:
                 s=list(sorteddictnew.keys()).index(w)
                 filtered_sentence.append(s)
               except:
                 continue
            final_string.append(filtered_sentence)
        X_test_new=final_string
```

5.3 Create padding in the input

```
In [8]: X_train=X_train_new
        X_test=X_test_new

        # truncate and/or pad input sequences
        max_review_length = 600
        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)
```

```
(35000, 600)
```

5.4 Create model

1 Layer LSTM

```
In [9]: # create the model
        embedding_vecor_length = 32
        model = Sequential()
        model.add(Embedding(topitem, 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())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 600, 32)           160000
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
```

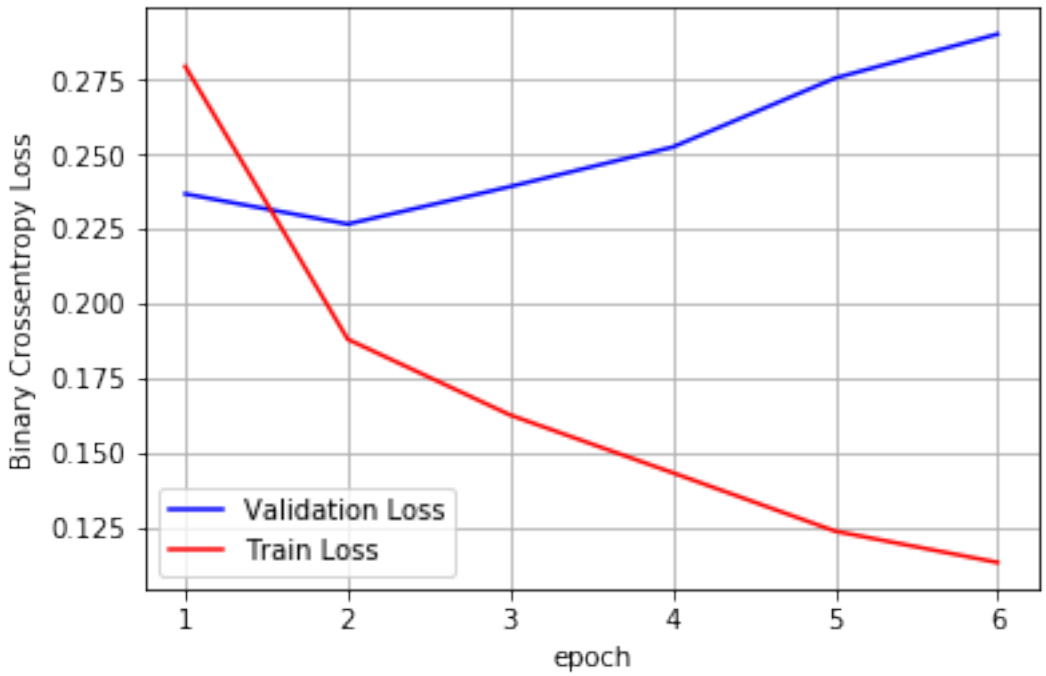
```
In [10]: import warnings
         warnings.filterwarnings('ignore')
         history=model.fit(X_train, y_train, nb_epoch=nb_epoch, batch_size=64,validation_data=
         # Final evaluation of the model
         scores = model.evaluate(X_test, y_test, verbose=0)
         print("Accuracy: %.2f%%"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

         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()

         fig,ax = plt.subplots(1,1)
         ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
         x = list(range(1,nb_epoch+1))
         vy = history.history['val_loss']
         ty = history.history['loss']
         plt_dynamic(x, vy, ty, ax)
```

```
Train on 35000 samples, validate on 15000 samples
Epoch 1/6
35000/35000 [==============================] - 887s 25ms/step - loss: 0.2794 - acc: 0.8902 - va
Epoch 2/6
35000/35000 [==============================] - 637s 18ms/step - loss: 0.1880 - acc: 0.9262 - va
Epoch 3/6
35000/35000 [==============================] - 626s 18ms/step - loss: 0.1626 - acc: 0.9382 - va
```

```
Epoch 4/6
35000/35000 [==============================] - 627s 18ms/step - loss: 0.1432 - acc: 0.9456 - va
Epoch 5/6
35000/35000 [==============================] - 625s 18ms/step - loss: 0.1237 - acc: 0.9536 - va
Epoch 6/6
35000/35000 [==============================] - 625s 18ms/step - loss: 0.1132 - acc: 0.9582 - va
Accuracy: 90.16%
```




```
In [11]: aa=pd.DataFrame({'type':['LSTM 1 layer 100'],'test_accuracy':[scores[1]*100],'test_sc
```

2 Layer LSTM

```
In [12]: # create the model
         embedding_vecor_length = 32
         model = Sequential()
         model.add(Embedding(topitem, embedding_vecor_length, input_length=max_review_length))
         model.add(LSTM(100,return_sequences=True))
         model.add(LSTM(100))
         model.add(Dense(250, activation='relu'))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         print(model.summary())
         #param in 2nd LSTM=4*100(100+100+1) m and n are both 100.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 600, 32)           160000
_________________________________________________________________
lstm_2 (LSTM)                (None, 600, 100)          53200
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_2 (Dense)              (None, 250)               25250
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 251
=================================================================
Total params: 319,101
Trainable params: 319,101
Non-trainable params: 0
_________________________________________________________________
None
```

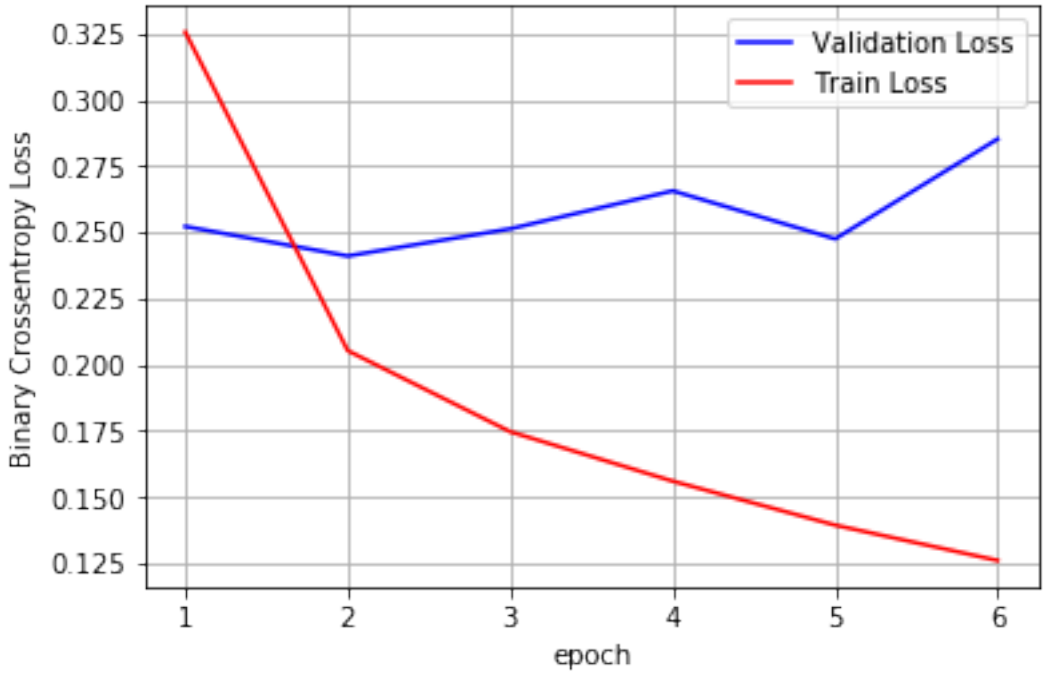
```
In [13]: import warnings
         warnings.filterwarnings('ignore')
         history=model.fit(X_train, y_train, nb_epoch=nb_epoch, batch_size=64,validation_data=
         # Final evaluation of the model
         scores = model.evaluate(X_test, y_test, verbose=0)
         print("Accuracy: %.2f%%" % (scores[1]*100))

         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()

         fig,ax = plt.subplots(1,1)
         ax.set_xlabel('epoch') ; ax.set_ylabel('Binary Crossentropy Loss')
         x = list(range(1,nb_epoch+1))
         vy = history.history['val_loss']
         ty = history.history['loss']
         plt_dynamic(x, vy, ty, ax)

Train on 35000 samples, validate on 15000 samples
Epoch 1/6
35000/35000 [==============================] - 1460s 42ms/step - loss: 0.3255 - acc: 0.8739 - v
Epoch 2/6
35000/35000 [==============================] - 1516s 43ms/step - loss: 0.2053 - acc: 0.9201 - v
Epoch 3/6
35000/35000 [==============================] - 1512s 43ms/step - loss: 0.1747 - acc: 0.9330 - v
```

```
Epoch 4/6
35000/35000 [==============================] - 1557s 44ms/step - loss: 0.1560 - acc: 0.9405 - v
Epoch 5/6
35000/35000 [==============================] - 1660s 47ms/step - loss: 0.1394 - acc: 0.9477 - v
Epoch 6/6
35000/35000 [==============================] - 1689s 48ms/step - loss: 0.1261 - acc: 0.9527 - v
Accuracy: 90.00%
```




```
In [14]: bb=pd.DataFrame({'type':['LSTM 2 layer 100'],'test_accuracy':[scores[1]*100],'test_sc
         aa=aa.append(bb)
```

6 Observation

Model quickly overfit after 2 epoch The 2 different model accuracy is below

```
In [15]: aa
```

```
Out[15]:    test_accuracy  test_score              type
         0          90.16   29.024494  LSTM 1 layer 100
         0          90.00   28.517860  LSTM 2 layer 100
```

```
In [ ]:
```