In [0]:
```
#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.metrics import accuracy_score
from sklearn.model_selection import cross_validate
```

In [2]:
```
# Load the Drive helper and mount
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=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=emai
l%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2
Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fpeopleapi.readonly&response_type=code

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

In [3]:
```
%cd '/content/drive/My Drive'
con = sqlite3.connect('database.sqlite')

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

```
/content/drive/My Drive
```

In [0]:
```
# 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 [0]:
```
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 [0]:
```
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 [0]:
```
filtered_data['CleanedText']=final_string #adding a column of CleanedTe
xt which displays the data after pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("u
tf-8")
```

In [0]:
```
sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(50000)

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

In [0]:
```
from sklearn.model_selection import train_test_split

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

In [0]:
```
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])
```

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

In [0]:
```
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 [0]:
```
maindataxtr=np.asarray(maindata)
```

In [0]:
```
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 [0]:
```
maindataxtest = np.asarray(maindata)
```

In [17]:
```
from keras.preprocessing import sequence
max_review_length = 600
maindataxtr = sequence.pad_sequences(maindataxtr, maxlen=max_review_len
gth)
```

```
maindataxtest = sequence.pad_sequences(maindataxtest, maxlen=max_review
_length)
```

```
Using TensorFlow backend.
```

In [18]:
```
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]]
```

In [0]:
```
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 [0]:
```
embedding_vecor_length = 32
model1 = Sequential()
model1.add(Embedding(5001, embedding_vecor_length, input_length=max_rev
iew_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_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 [0]:
```
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))
```

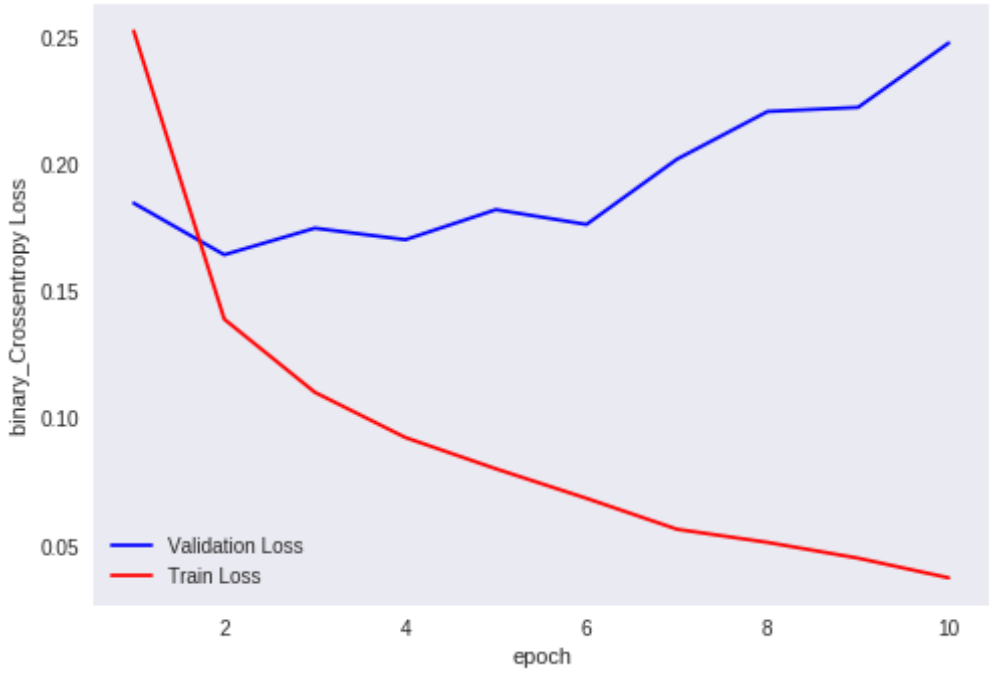
```
Train on 30000 samples, validate on 20000 samples
Epoch 1/10
30000/30000 [==============================] - 407s 14ms/step - loss:
0.2528 - acc: 0.9093 - val_loss: 0.1849 - val_acc: 0.9325
Epoch 2/10
30000/30000 [==============================] - 409s 14ms/step - loss:
0.1392 - acc: 0.9502 - val_loss: 0.1646 - val_acc: 0.9396
Epoch 3/10
30000/30000 [==============================] - 411s 14ms/step - loss:
0.1104 - acc: 0.9615 - val_loss: 0.1751 - val_acc: 0.9456
Epoch 4/10
30000/30000 [==============================] - 407s 14ms/step - loss:
0.0926 - acc: 0.9700 - val_loss: 0.1705 - val_acc: 0.9450
Epoch 5/10
30000/30000 [==============================] - 411s 14ms/step - loss:
0.0802 - acc: 0.9740 - val_loss: 0.1824 - val_acc: 0.9464
```

```
Epoch 6/10
30000/30000 [==============================] - 417s 14ms/step - loss:
0.0686 - acc: 0.9791 - val_loss: 0.1765 - val_acc: 0.9438
Epoch 7/10
30000/30000 [==============================] - 417s 14ms/step - loss:
0.0564 - acc: 0.9832 - val_loss: 0.2023 - val_acc: 0.9469
Epoch 8/10
30000/30000 [==============================] - 415s 14ms/step - loss:
0.0512 - acc: 0.9841 - val_loss: 0.2211 - val_acc: 0.9433
Epoch 9/10
30000/30000 [==============================] - 407s 14ms/step - loss:
0.0450 - acc: 0.9865 - val_loss: 0.2227 - val_acc: 0.9463
Epoch 10/10
30000/30000 [==============================] - 404s 13ms/step - loss:
0.0373 - acc: 0.9896 - val_loss: 0.2481 - val_acc: 0.9471
Accuracy: 94.71%
```

In [0]:

```
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: 0.24807016671383753
Test accuracy: 0.9471
```



In [0]:
```
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
```

In [24]:
```
from keras.layers import Dropout
from keras.layers import BatchNormalization
from keras import optimizers
from keras.layers import Activation
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(LSTM(50))
model2.add(Dropout(0.25))


RMSprop = optimizers.RMSprop()

model2.add(Dense(1, activation='softmax'))
model2.compile(loss='binary_crossentropy', optimizer=RMSprop, metrics=[
'accuracy'])
print(model2.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 600, 32)           160032
_________________________________________________________________
lstm_7 (LSTM)                (None, 600, 100)          53200
_________________________________________________________________
lstm_8 (LSTM)                (None, 50)                30200
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 243,483
Trainable params: 243,483
Non-trainable params: 0
_________________________________________________________________
None
```

In [28]:

```
history=model2.fit(maindataxtr, y_train, nb_epoch=20, batch_size=64,val
idation_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 1/20
30000/30000 [==============================] - 793s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
```

```
Epoch 2/20
30000/30000 [==============================] - 793s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 3/20
30000/30000 [==============================] - 794s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 4/20
30000/30000 [==============================] - 796s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 5/20
30000/30000 [==============================] - 795s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 6/20
30000/30000 [==============================] - 797s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 7/20
30000/30000 [==============================] - 797s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 8/20
30000/30000 [==============================] - 795s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 9/20
30000/30000 [==============================] - 792s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 10/20
30000/30000 [==============================] - 796s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 11/20
30000/30000 [==============================] - 796s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 12/20
30000/30000 [==============================] - 796s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 13/20
30000/30000 [==============================] - 796s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 14/20
30000/30000 [==============================] - 796s 27ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
```

```
Epoch 15/20
30000/30000 [==============================] - 792s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 16/20
30000/30000 [==============================] - 793s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 17/20
30000/30000 [==============================] - 791s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 18/20
30000/30000 [==============================] - 791s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 19/20
30000/30000 [==============================] - 789s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Epoch 20/20
30000/30000 [==============================] - 789s 26ms/step - loss:
1.7489 - acc: 0.8903 - val_loss: 1.7441 - val_acc: 0.8906
Accuracy: 89.06%
```

In [31]:

```
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()

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(1,21))


vy = history.history['val_loss']
```

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

```
Test score: 1.7440970114707954
Test accuracy: 0.8906
```