In this notebook, You will do amazon review classification with BERT.[Download data from this link]

```
It contains 5 parts as below.  Detailed instrctions are given in the each cell. please read every comment we have written.
    1. Preprocessing
    2. Creating a BERT model from the Tensorflow HUB.
    3. Tokenization
    4. getting the pretrained embedding Vector for a given review from the BERT.
    5. Using the embedding data apply NN and classify the reviews.
    6. Creating a Data pipeline for BERT Model.
```

instructions:

```
    1. Don't change any Grader Functions. Don't manipulate any Grader functions.
    If you manipulate any, it will be considered as plagiarised.

    2. Please read the instructions on the code cells and markdown cells. We will explain what to write.

    3. please return outputs in the same format what we asked. Eg. Don't return List if we are asking for a numpy array.

    4. Please read the external links that we are given so that you will learn the concept behind the code that you are writing.

    5. We are giving instructions at each section if necessary, please follow them.
```

Every Grader function has to return True.

In [2]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [2]:
```python
#all imports
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.keras.models import Model
import os
```

In [4]:
```python
os.chdir('/content/drive/MyDrive/26_NLP_Transfer_Learning/')
```

In [5]:
```python
tf.test.gpu_device_name()
```

Out[5]:
```
'/device:GPU:0'
```

Grader function 1

In [6]:
```python
def grader_tf_version():
    assert((tf.__version__)>'2')
    return True
grader_tf_version()
```

Out[6]:
```
True
```

Part-1: Preprocessing

In [7]:
```python
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv('/content/drive/MyDrive/26_NLP_Transfer_Learning/Reviews.csv')
#check the info of the dataset
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Id                      568454 non-null  int64
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568438 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64
 5   HelpfulnessDenominator  568454 non-null  int64
 6   Score                   568454 non-null  int64
 7   Time                    568454 non-null  int64
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

In [8]:
```python
#get only 2 columns - Text, Score
reviews = reviews[[ 'Text' , 'Score' ]]

#drop the NAN values
reviews.dropna(axis = 0 , inplace = True)
```

In [9]:
```python
#if score == 3, remove the rows.
reviews = reviews.loc[ reviews['Score'] != 3 , : ]
#if score<=2, set score = 0
reviews.loc[ reviews['Score'] <= 2 , 'Score' ] = 0
#if score> 3, set score = 1
reviews.loc[ reviews['Score'] > 3 , 'Score' ] = 1
```

Grader function 2

In [10]:
```python
def grader_reviews():
    temp_shape = (reviews.shape == (525814, 2)) and (reviews.Score.value_counts()[1]==443777)
    assert(temp_shape == True)
    return True
grader_reviews()
```

Out[10]:
```
True
```

In [11]:
```python
def get_wordlen(x):
    return len(x.split())

reviews['len'] = reviews.Text.apply(get_wordlen)
reviews = reviews[reviews.len<50]
reviews = reviews.sample(n=100000, random_state=30)
```

In [12]:
```python
#remove HTML from the Text column and save in the Text column only
reviews['Text'] = reviews['Text'].str.replace(r"<[^<>]*>" , " ")
```

In [13]:
```python
#print head 5
reviews.head(5)
```

Out[13]:

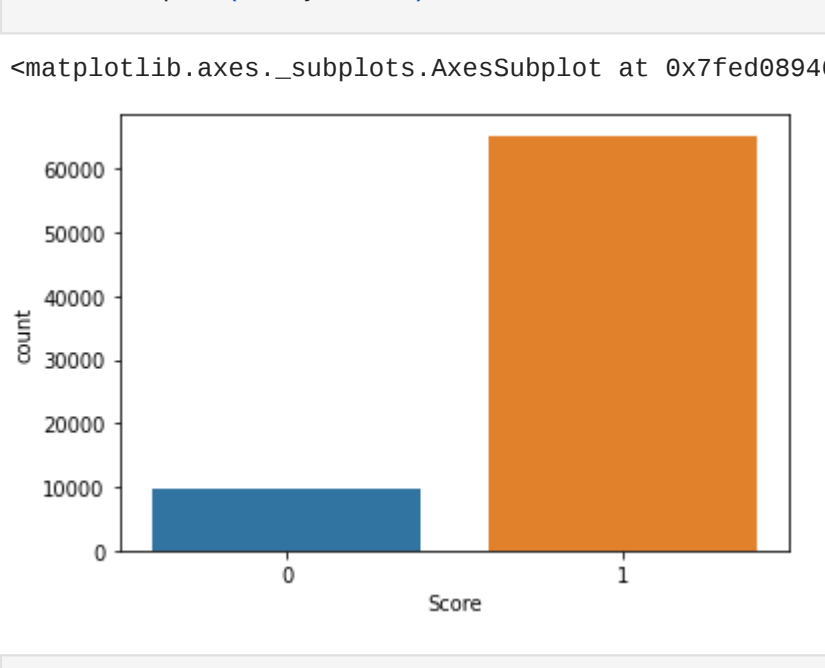
	Text	Score	len
64117	The tea was of great quality and it tasted lik...	1	30
418112	My cat loves this. The pellets are nice and s...	1	31
357829	Great product. Does not completely get rid of ...	1	41
175872	This gum is my favorite! I would advise every...	1	27
178716	I also found out about this product because of...	1	22

In [14]:
```python
#split the data into train and test data(20%) with Stratify sampling, random state 33

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split( reviews['Text'] , reviews['Score'], stratify=reviews['Score'], random_state=33)
```

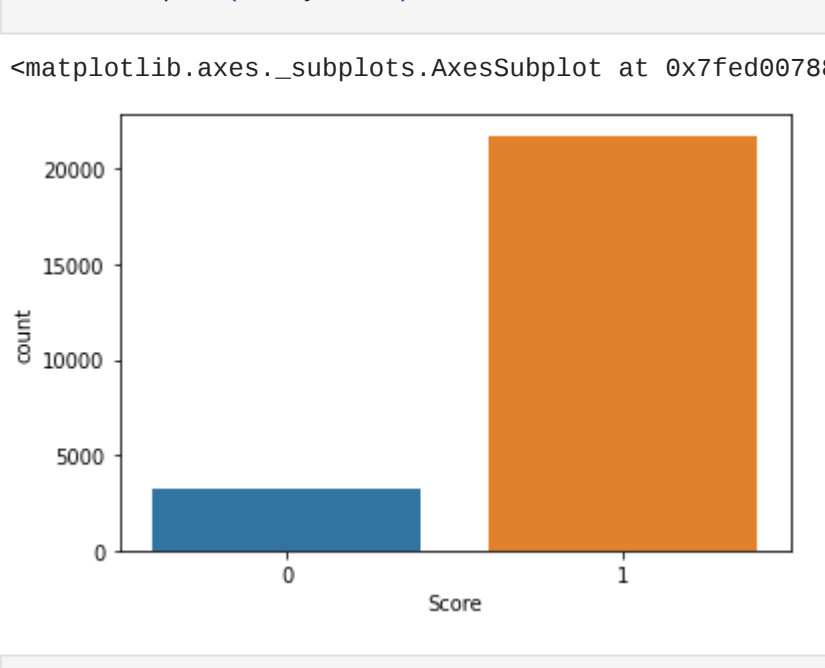
In [15]:
```python
import seaborn as sns
sns.countplot(x = y_train)
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed089467d0>
```



In [16]:
```python
sns.countplot(x = y_test)
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed00788210>
```



In [17]:
```python
#saving to disk. if we need, we can load preprocessed data directly.
reviews.to_csv('/content/drive/MyDrive/26_NLP_Transfer_Learning/preprocessed.csv', index=False)
```

Part-2: Creating BERT Model

```
If you want to know more about BERT, You can watch live sessions on Transformers and BERt.
we will strongly recommend you to read Transformers, BERT Paper and, This blog.


For this assignment, we are using BERT uncased Base model.
It uses L=12 hidden layers (i.e., Transformer blocks), a hidden size of H=768, and A=12 attention heads.
```

In [18]:
```python
## Loading the Pretrained Model from tensorflow HUB
tf.keras.backend.clear_session()

max_seq_length = 55  # maximum length of a seq in the data we have

#BERT takes 3 inputs

#input words
#Sequence of words represented as integers
input_word_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input_word_ids")

#mask vector
#if you are padding anything
input_mask = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input_mask")

#segment vectors
#If you are giving only one sentence for the classification, total seg vector is 0. If you are giving two sentenced with [sep] token separated, first seq segm
segment_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="segment_ids")

#bert layer
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1", trainable=False)
pooled_output, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])

#Bert model
#We are using only pooled output not sequence out. If you want to know about those, please read https://www.kaggle.com/questions-and-answers/86510
bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_output)
```

In [19]:
```python
bert_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_word_ids (InputLayer)     [(None, 55)]         0
__________________________________________________________________________________________________
input_mask (InputLayer)         [(None, 55)]         0
__________________________________________________________________________________________________
segment_ids (InputLayer)        [(None, 55)]         0
__________________________________________________________________________________________________
keras_layer (KerasLayer)        [(None, 768), (None, 109482241   input_word_ids[0][0]
                                                                 input_mask[0][0]
                                                                 segment_ids[0][0]
==================================================================================================
Total params: 109,482,241
Trainable params: 0
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
```

In [20]:
```python
bert_model.output
```

Out[20]:
```
<KerasTensor: shape=(None, 768) dtype=float32 (created by layer 'keras_layer')>
```

Part-3: Tokenization

In [21]:
```python
#getting Vocab file
vocab_file    = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
```

In [22]:
```python
!pip install sentencepiece
import tokenization as tk
```

```
Collecting sentencepiece
  Downloading sentencepiece-0.1.96-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.2 MB)
     |████████████████████████████████| 1.2 MB 15.9 MB/s eta 0:00:01
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.96
```

In [23]:
```python
# Create tokenizer " Instantiate FullTokenizer"
# name must be "tokenizer"
# the FullTokenizer takes two parameters 1. vocab_file and 2. do_lower_case
# we have created these in the above cell ex: FullTokenizer(vocab_file, do_lower_case )
# please check the "tokenization.py" file the complete implementation

tokenizer = tk.FullTokenizer(vocab_file, do_lower_case )
```

Grader function 3

In [24]:
```python
#it has to give no error
def grader_tokenize(tokenizer):
    out = False
    try:
        out=('[CLS]' in tokenizer.vocab) and ('[SEP]' in tokenizer.vocab)
    except:
        out = False
    assert(out==True)
    return out
grader_tokenize(tokenizer)
```

Out[24]:
```
True
```

In [108…
```python
from tensorflow.keras.preprocessing.sequence import pad_sequences

def tokenization_function(tokens):
  global max_seq_length
  tokens = tokenizer.tokenize(tokens)   # train and test tokens using Tokenizer

  # maximum number of tokens i.e. max_seq_length = 55,
  # if it is > 55 then truncate the tokens length (similar to padding)
  if len(tokens) > max_seq_length - 2:
    tokens = tokens[ : max_seq_length - 2 ]

  # add '[CLS]' at start of the Tokens and '[SEP]' at the end of the tokens
  tokens = ['[CLS]' , *tokens , '[SEP]']
  # if it is less than 55 then add '[PAD]' token. Based on padding, create the mask for Train and Test ( 1 for real token, 0 for '[PAD]'), it will also same s
  mask = np.array( [1]*len(tokens) + [0]*(max_seq_length - len(tokens)))

  # Create a segment input for train and test. We are using only one sentence so all zeros. This shape will also (None, 55)
  segment = np.array( [0]*max_seq_length )

  tokens = np.array(tokenizer.convert_tokens_to_ids(tokens) + [0]*(max_seq_length - len(tokens)) )
  return tokens , mask , segment
```

In [26]:
```python
X_train_tokens = []
X_train_mask = []
X_train_segment = []

for i in X_train:
  tokens , mask , segment = tokenization_function(i)
  X_train_tokens.append(tokens)
  X_train_mask.append(mask)
  X_train_segment.append(segment)
```

In [27]:
```python
X_test_tokens = []
X_test_mask = []
X_test_segment = []

for i in X_test:
  tokens , mask , segment = tokenization_function(i)
  X_test_tokens.append(tokens)
  X_test_mask.append(mask)
  X_test_segment.append(segment)
```

Example

In [28]:
```python
import pickle
```

In [29]:
```python
##save all your results to disk so that, no need to run all again.
pickle.dump((X_train, X_train_tokens, X_train_mask, X_train_segment, y_train),open('train_data.pkl','wb'))
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),open('test_data.pkl','wb'))
```

In [30]:
```python
#you can load from disk
X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.load(open("train_data.pkl", 'rb'))
X_test, X_test_tokens, X_test_mask, X_test_segment, y_test      = pickle.load(open("test_data.pkl", 'rb'))
```

Grader function 4

In [31]:
```python
X_train_tokens  = np.array(X_train_tokens)
X_train_mask    = np.array(X_train_mask)
X_train_segment = np.array(X_train_segment)

X_test_tokens  = np.array(X_test_tokens)
X_test_mask    = np.array(X_test_mask)
X_test_segment = np.array(X_test_segment)
```

In [32]:
```python
print (X_train_tokens.shape)
print (X_train_mask.shape)
print (X_train_segment.shape)

print (X_test_tokens.shape)
print (X_test_mask.shape)
print (X_test_segment.shape)
```

```
(75000, 55)
(75000, 55)
(75000, 55)
(25000, 55)
(25000, 55)
(25000, 55)
```

In [33]:
```python
def grader_alltokens_train():
  out = False
  if type(X_train_tokens) == np.ndarray:
    temp_shapes = (X_train_tokens.shape[1]==max_seq_length) and (X_train_mask.shape[1]==max_seq_length) and (X_train_segment.shape[1]==max_seq_length)
    segment_temp = not np.any(X_train_segment)
    mask_temp = np.sum(X_train_mask==0) == np.sum(X_train_tokens==0)
    no_cls = np.sum(X_train_tokens==tokenizer.vocab['[CLS]'])==X_train_tokens.shape[0]
    no_sep = np.sum(X_train_tokens==tokenizer.vocab['[SEP]'])==X_train_tokens.shape[0]
    out = temp_shapes and segment_temp and mask_temp and no_cls and no_sep
  else:
    print('Type of all above token arrays should be numpy array not list')
    out = False
  assert(out==True)
  return out
```

In [34]:
```python
grader_alltokens_train()
```

Out[34]:
```
True
```

Grader function 5

In [35]:
```python
def grader_alltokens_test():
  out = False
  if type(X_test_tokens) == np.ndarray:
    temp_shapes = (X_test_tokens.shape[1]==max_seq_length) and (X_test_mask.shape[1]==max_seq_length) and (X_test_segment.shape[1]==max_seq_length)
    segment_temp = not np.any(X_test_segment)
    mask_temp = np.sum(X_test_mask==0) == np.sum(X_test_tokens==0)
    no_cls = np.sum(X_test_tokens==tokenizer.vocab['[CLS]'])==X_test_tokens.shape[0]
    no_sep = np.sum(X_test_tokens==tokenizer.vocab['[SEP]'])==X_test_tokens.shape[0]
    out = temp_shapes and segment_temp and mask_temp and no_cls and no_sep
  else:
    print('Type of all above token arrays should be numpy array not list')
    out = False
  assert(out==True)
  return out
grader_alltokens_test()
```

Out[35]:
```
True
```

Part-4: Getting Embeddings from BERT Model

```
We already created the BERT model in the part-2 and input data in the part-3.
We will utlize those two and will get the embeddings for each sentence in the
Train and test data.
```

In [36]:
```python
bert_model.input
```

Out[36]:
```
[<KerasTensor: shape=(None, 55) dtype=int32 (created by layer 'input_word_ids')>,
 <KerasTensor: shape=(None, 55) dtype=int32 (created by layer 'input_mask')>,
 <KerasTensor: shape=(None, 55) dtype=int32 (created by layer 'segment_ids')>]
```

In [37]:
```python
bert_model.output
```

Out[37]:
```
<KerasTensor: shape=(None, 768) dtype=float32 (created by layer 'keras_layer')>
```

In [38]:
```python
# get the train output, BERT model will give one output so save in
# X_train_pooled_output
X_train_pooled_output=bert_model.predict([X_train_tokens,X_train_mask,X_train_segment])
```

In [39]:
```python
# get the test output, BERT model will give one output so save in
# X_test_pooled_output
X_test_pooled_output=bert_model.predict([X_test_tokens,X_test_mask,X_test_segment])
```

In [40]:
```python
##save all your results to disk so that, no need to run all again.
pickle.dump((X_train_pooled_output, X_test_pooled_output),open('final_output.pkl','wb'))
```

In [41]:
```python
X_train_pooled_output, X_test_pooled_output= pickle.load(open('final_output.pkl', 'rb'))
```

Grader function 6

In [42]:
```python
#now we have X_train_pooled_output, y_train
#X_test_pooled_ouput, y_test

#please use this grader to evaluate
def greader_output():
    assert(X_train_pooled_output.shape[1]==768)
    assert(len(y_train)==len(X_train_pooled_output))
    assert(X_test_pooled_output.shape[1]==768)
    assert(len(y_test)==len(X_test_pooled_output))
    assert(len(y_train.shape)==1)
    assert(len(X_train_pooled_output.shape)==2)
    assert(len(y_test.shape)==1)
    assert(len(X_test_pooled_output.shape)==2)
    return True
greader_output()
```

Out[42]:
```
True
```

Part-5: Training a NN with 768 features

```
Create a NN and train the NN.
1. You have to use AUC as metric.
2. You can use any architecture you want.
3. You have to use tensorboard to log all your metrics and Losses. You have to send those logs.
4. Print the loss and metric at every epoch.
5. You have to submit without overfitting and underfitting.
```

In [70]:
```python
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout
from tensorflow.keras.models import Model
```

In [122…
```python
inp = tf.keras.Input( shape = 768 )
x = Dense(124 , activation = 'relu')(inp)
x = Dropout(0.1)(x)
x = Dense(64 , activation = 'relu')(x)
x = Dropout(0.1)(x)
x = Dense(32 , activation = 'relu')(x)
x = Dropout(0.1)(x)
x = Dense(16 , activation = 'relu')(x)
x = Dropout(0.1)(x)
out = Dense(1,activation = 'sigmoid')(x)
```

In [123…
```python
model = tf.keras.Model( inputs = inp , outputs = out )
model.compile(optimizer='adam', loss='binary_crossentropy')
model.summary()
```

```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 768)]             0
_________________________________________________________________
dense_28 (Dense)             (None, 124)               95356
_________________________________________________________________
dropout_20 (Dropout)         (None, 124)               0
_________________________________________________________________
dense_29 (Dense)             (None, 64)                8000
_________________________________________________________________
dropout_21 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_30 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_22 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_31 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_23 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_32 (Dense)             (None, 1)                 17
=================================================================
Total params: 105,981
Trainable params: 105,981
Non-trainable params: 0
_________________________________________________________________
```

In [124…
```python
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import (Callback, EarlyStopping,TensorBoard, ReduceLROnPlateau)
import datetime

class ROCCallback(Callback):
  def __init__(self, validation_data=()):
    super(Callback,self).__init__()
    self.validation_data = validation_data
  def on_epoch_end(self, epoch, logs={}):
    y_pred = self.model.predict(self.validation_data[0],verbose = 0)
    y_true = self.validation_data[1]
    score = roc_auc_score(y_true, y_pred)
    logs['auc'] = score
    print ('val_auroc :',score)
```

In [125…
```python
class AchieveTarget(Callback):
  def __init__(self, target):
    super(AchieveTarget, self).__init__()
    self.target = target
  def on_epoch_end(self, epoch, logs={}):
    acc = logs['auc']
    if acc >= self.target:
      self.model.stop_training = True
```

In [128…
```python
log_dir ="/content/drive/MyDrive/26_NLP_Transfer_Learning/logs/fit/"+"bert_" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard = TensorBoard(log_dir=log_dir, histogram_freq=1, write_graph=True,write_grads=True)
roc_callback = ROCCallback((X_test_pooled_output, y_test))
early_stopping = EarlyStopping(patience=5)
reduce_lr = ReduceLROnPlateau(patience=3)
target = AchieveTarget(0.952)

callbacks = [roc_callback,tensorboard ,reduce_lr,target,early_stopping]
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

In [129…
```python
history = model.fit(X_train_pooled_output, y_train, batch_size=32, epochs=100,callbacks=callbacks, validation_data=(X_test_pooled_output, y_test))
```

```
Epoch 1/100
   3/2344 [..............................] - ETA: 6:19 - loss: 0.1496WARNING:tensorflow:Callback method `on_train_batch_begin` is slow compared to the batch t
ime (batch time: 0.0030s vs `on_train_batch_begin` time: 0.0332s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_begin` is slow compared to the batch time (batch time: 0.0030s vs `on_train_batch_begin` time: 0.0332s). Ch
eck your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0030s vs `on_train_batch_end` time: 0.0200s). Check
your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0030s vs `on_train_batch_end` time: 0.0200s). Check
your callbacks.
2344/2344 [==============================] - 7s 3ms/step - loss: 0.2091 - val_loss: 0.1862
val_auroc : 0.9463946772473742
Epoch 2/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.2059 - val_loss: 0.1901
val_auroc : 0.9475729944894224
Epoch 3/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.2044 - val_loss: 0.1940
val_auroc : 0.9470934348454546
Epoch 4/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.2114 - val_loss: 0.1957
val_auroc : 0.947921952311791
Epoch 5/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1936 - val_loss: 0.1851
val_auroc : 0.9487405715199108
Epoch 6/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1884 - val_loss: 0.1823
val_auroc : 0.9492742284350172
Epoch 7/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1810 - val_loss: 0.1817
val_auroc : 0.9499616232005776
Epoch 8/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1797 - val_loss: 0.1809
val_auroc : 0.950003260977278
Epoch 9/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1790 - val_loss: 0.1799
val_auroc : 0.9503878072477013
Epoch 10/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1793 - val_loss: 0.1841
val_auroc : 0.9500746642810096
Epoch 11/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1785 - val_loss: 0.1797
val_auroc : 0.9505244654731844
Epoch 12/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1780 - val_loss: 0.1801
val_auroc : 0.9508472732152327
Epoch 13/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1767 - val_loss: 0.1859
val_auroc : 0.9508103404719231
Epoch 14/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1762 - val_loss: 0.1785
val_auroc : 0.9511289455224915
Epoch 15/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1769 - val_loss: 0.1787
val_auroc : 0.9514073773706629
Epoch 16/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1757 - val_loss: 0.1806
val_auroc : 0.9514156270983829
Epoch 17/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1762 - val_loss: 0.1780
val_auroc : 0.9515577969344097
Epoch 18/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1760 - val_loss: 0.1839
val_auroc : 0.9511399262921497
Epoch 19/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1755 - val_loss: 0.1778
val_auroc : 0.9515660749630827
Epoch 20/100
2344/2344 [==============================] - 7s 3ms/step - loss: 0.1752 - val_loss: 0.1762
val_auroc : 0.9520123173104519
```

In [130…
```python
%reload_ext tensorboard
%tensorboard --logdir='/content/drive/MyDrive/26_NLP_Transfer_Learning/logs/fit/'
```

```
Output hidden; open in https://colab.research.google.com to view.
```

This is an image

Part-6: Creating a Data pipeline for BERT Model

```
1. Download data from here

2. Read the csv file

3. Remove all the html tags

4. Now do tokenization [Part 3 as mentioned above]

    • Create tokens,mask array and segment array


5. Get Embeddings from BERT Model [Part 4 as mentioned above] , let it be X_test

    • Print the shape of output(X_test.shape).You should get (352,768)


6. Predit the output of X_test with the Neural network model which we trained earlier.

7. Print the occurences of class labels in the predicted output


</pre>
```

In [131…
```python
class BertVectorizer():

  def __init__(self, max_sequence_len, tokenizer, bert_model):
    super(BertVectorizer, self).__init__()
    self.max_sequence_len = max_seq_length
    self.tokenizer = tokenizer
    self.bert_model = bert_model

  def bert_vectors(self,X):

    #remove all the html tags
    data = X.str.replace(r"<[^<>]*>" , " ")

    #tokenization
    data_tokens = []
    data_mask = []
    data_segment = []
    for i in data:
      tokens,mask,segment = tokenization_function(i)
      data_tokens.append(tokens)
      data_mask.append(mask)
      data_segment.append(segment)

    data_tokens  = np.array(data_tokens)
    data_mask    = np.array(data_mask)
    data_segment = np.array(data_segment)

    bert_vectors = bert_model.predict([data_tokens,data_mask,data_segment])

    return bert_vectors
```

In [132…
```python
def pipeline(X, max_sequence_len, tokenizer, bert_model, classifier):
  vectorizer = BertVectorizer(max_sequence_len, tokenizer, bert_model)
  bertvectors = vectorizer.bert_vectors(X)
  print(bertvectors.shape)
  return classifier.predict(bertvectors)
```

In [133…
```python
def review_classifier(review):
  global max_seq_length
  global tokenizer
  global bert_model
  global model
  return pipeline(review, max_seq_length, tokenizer, bert_model, model)
```

In [137…
```python
X_test = pd.read_csv('./test.csv')
predicts = review_classifier(X_test['Text'])
```

```
(352, 768)
```

In [138…
```python
predicted_labels = [1 if x > 0.5 else 0 for x in predicts]
print(predicted_labels)
```

```
[0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1]
```

In [139…
```python
pd.DataFrame(predicted_labels).value_counts()
```

Out[139…
```
1    301
0     51
dtype: int64
```