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.

```python
%tensorflow_version 2.x
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

```python
%tensorflow_version 2.x
import tensorflow as tf
import timeit

device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  print(
      '\n\nThis error most likely means that this notebook is not '
      'configured to use a GPU.  Change this in Notebook Settings via the '
      'command palette (cmd/ctrl-shift-P) or the Edit menu.\n\n')
  raise SystemError('GPU device not found')

def cpu():
  with tf.device('/cpu:0'):
    random_image_cpu = tf.random.normal((100, 100, 100, 3))
    net_cpu = tf.keras.layers.Conv2D(32, 7)(random_image_cpu)
    return tf.math.reduce_sum(net_cpu)

def gpu():
  with tf.device('/device:GPU:0'):
    random_image_gpu = tf.random.normal((100, 100, 100, 3))
    net_gpu = tf.keras.layers.Conv2D(32, 7)(random_image_gpu)
    return tf.math.reduce_sum(net_gpu)

# We run each op once to warm up; see: https://stackoverflow.com/a/45067900
cpu()
gpu()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=1663.2822>
```

```python
#in this assignment you need two files reviews.csv and tokenization file
#you can use gdown module to import both the files in colab from Google drive
#the syntax is for gdown is !gdown --id file_id
#please run the below cell to import the required files
```

```python
!gdown --id 1GsD8JlAc_0yJ-1151LNr6rLw83RRUPgt
!gdown --id 13exfXiyiByluh1PfYK1EyZyizqxeCVG9
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You don't need to pass it anymore to use a file ID.
  category=FutureWarning,
Access denied with the following error:

        Cannot retrieve the public link of the file. You may need to change
        the permission to 'Anyone with the link', or have had many accesses.

You may still be able to access the file from the browser:

         https://drive.google.com/uc?id=1GsD8JlAc_0yJ-1151LNr6rLw83RRUPgt

/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You don't need to pass it anymore to use a file ID.
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=13exfXiyiByluh1PfYK1EyZyizqxeCVG9
To: /content/tokenization.py
100% 17.3k/17.3k [00:00<00:00, 25.6MB/s]
```

```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
```

```python
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

Grader function 1

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

```
True
```

Part-1: Preprocessing

```python
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv(r"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
```

```python
#get only 2 columns - Text, Score
#drop the NAN values
```

```python
reviews = reviews[['Text','Score']].dropna()
```

```python
reviews.head()
```

	Text	Score
0	I have bought several of the Vitality canned d...	5
1	Product arrived labeled as Jumbo Salted Peanut...	1
2	This is a confection that has been around a fe...	4
3	If you are looking for the secret ingredient i...	2
4	Great taffy at a great price. There was a wid...	5

```python
#if score> 3, set score = 1
#if score<=2, set score = 0
#if score == 3, remove the rows.
```

```python
reviews.loc[reviews['Score'] <= 2, 'Score'] = 0
reviews.loc[reviews['Score'] > 3, 'Score'] = 1

reviews.drop(reviews[reviews['Score'] == 3].index, inplace = True)
```

```python
print (reviews)
```

```
                                                     Text  Score
0       I have bought several of the Vitality canned d...      1
1       Product arrived labeled as Jumbo Salted Peanut...      0
2       This is a confection that has been around a fe...      1
3       If you are looking for the secret ingredient i...      0
4       Great taffy at a great price.  There was a wid...      1
...                                                   ...    ...
568449  Great for sesame chicken..this is a good if no...      1
568450  I'm disappointed with the flavor. The chocolat...      0
568451  These stars are small, so you can give 10-15 o...      1
568452  These are the BEST treats for training and rew...      1
568453  I am very satisfied ,product is as advertised,...      1

[525814 rows x 2 columns]
```

```python
reviews['Score'].value_counts()
```

```
1    87004
0    12996
Name: Score, dtype: int64
```

The data is unbalanced

Grader function 2

```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()
```

```
True
```

```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)
```

```python
#remove HTML from the Text column and save in the Text column only
```

```python
import re
reviews['Text']=reviews['Text'].apply(lambda x:re.sub(r'<.*?>','',x))
```

```python
#print head 5
```

```python
reviews.head()
```

	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

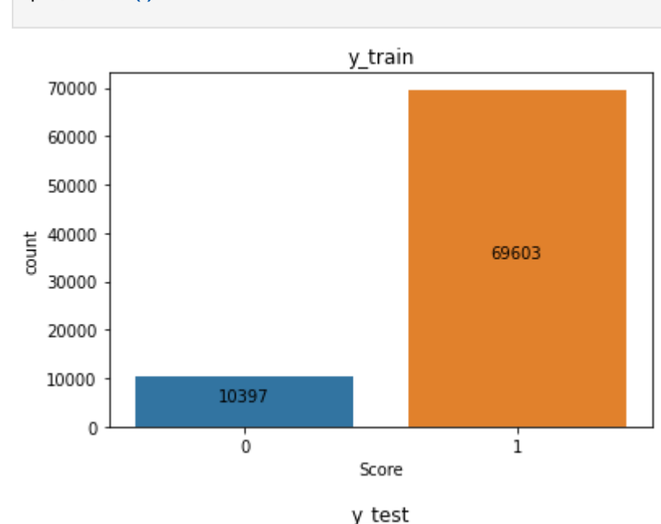
```python
#split the data into train and test data(20%) with Stratify sampling, random state 33,
```

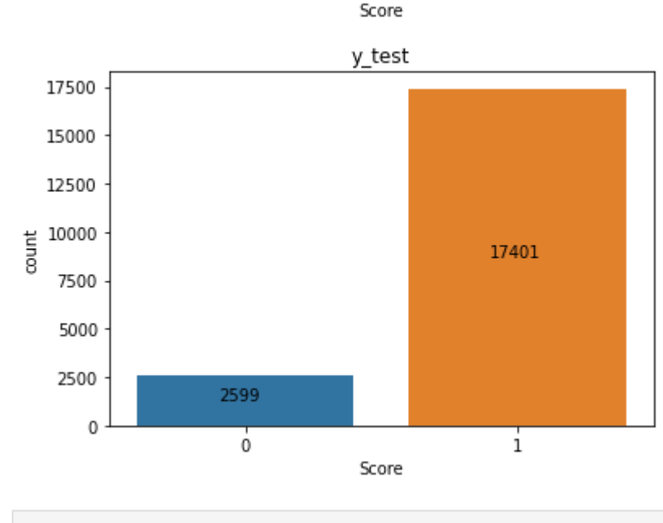
```python
from sklearn.model_selection  import train_test_split
x = reviews['Text']

y = reviews['Score']
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 33, test_size = 0.2, stratify = y)
```

```python
#plot bar graphs of y_train and y_test
```

```python
import seaborn as sean
import matplotlib.pyplot as plt
```

```python
count=sean.countplot(x=y_train)
for i in np.unique(y_train):
  count.annotate((y_train==i).sum(),(i-0.1,(y_train==i).sum()/2))
plt.title('y_train')
plt.show()
count=sean.countplot(x=y_test)
for i in np.unique(y_test):
  count.annotate((y_test==i).sum(),(i-0.1,(y_test==i).sum()/2))
plt.title('y_test')
plt.show()
```






```python
#saving to disk. if we need, we can load preprocessed data directly.
reviews.to_csv('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.
```

```python
## Loading the Pretrained Model from tensorflow HUB
tf.keras.backend.clear_session()

# maximum length of a seq in the data we have, for now i am making it as 55. You can change this
max_seq_length = 55

#BERT takes 3 inputs

#this is 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 segment vectors are zeros and
#second seq segment vector are 1's
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)
```

```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),        109482241   ['input_word_ids[0][0]',         
                                 (None, 55, 768)]                 'input_mask[0][0]',             
                                                                  'segment_ids[0][0]']            
                                                                                                  
==================================================================================================
Total params: 109,482,241
Trainable params: 0
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
```

```python
bert_model.output
```

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

Part-3: Tokenization

```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()
```

```python
!pip install sentencepiece
```

```
Collecting sentencepiece
  Downloading sentencepiece-0.1.96-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.2 MB)
     |████████████████████████████████| 1.2 MB 30.0 MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.96
```

```python
import tokenization  #We have given tokenization.py file
```

```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
```

```python
# if you are getting error for sentencepiece module you can install it using below command while running this cell for the first time
#!pip install sentencepiece
tokenizer=tokenization.FullTokenizer(vocab_file,do_lower_case )
```

Grader function 3

```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)
```

```
True
```

```python
# Create train and test tokens (X_train_tokens, X_test_tokens) from (X_train, X_test) using Tokenizer and

# add '[CLS]' at start of the Tokens and '[SEP]' at the end of the tokens.

# maximum number of tokens is 55(We already given this to BERT layer above) so shape is (None, 55)

# if it is less than 55, add '[PAD]' token else truncate the tokens length.(similar to padding)

# Based on padding, create the mask for Train and Test ( 1 for real token, 0 for '[PAD]'),
# it will also same shape as input tokens (None, 55) save those in X_train_mask, X_test_mask

# Create a segment input for train and test. We are using only one sentence so all zeros. This shape will also (None, 55)

# type of all the above arrays should be numpy arrays

# after execution of this cell, you have to get
# X_train_tokens, X_train_mask, X_train_segment
# X_test_tokens, X_test_mask, X_test_segment
def tokenizing_data(data,max_seq_length):
  data_tokens=np.zeros((data.shape[0],max_seq_length))

  data_mask=np.zeros((data.shape[0],max_seq_length))
  data_segment=np.zeros((data.shape[0],max_seq_length))

  for i in range(data.shape[0]):
    tokens=tokenizer.tokenize(data.values[i])

    if(len(tokens)>=max_seq_length-2):
      tokens=tokens[0:(max_seq_length-2)]
    tokens=['[CLS]',*tokens,'[SEP]']

    data_tokens[i]=np.array(tokenizer.convert_tokens_to_ids(tokens)+[0]*(max_seq_length-len(tokens)))
    data_mask[i]=np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
  return data_tokens,data_mask,data_segment

X_train_tokens,X_train_mask,X_train_segment=tokenizing_data(x_train,max_seq_length)
X_test_tokens,X_test_mask,X_test_segment=tokenizing_data(x_test,max_seq_length)
```

Example

```python
import pickle
```

```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'))
```

```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

```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

grader_alltokens_train()
```

```
True
```

Grader function 5

```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()
```

```
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.
```

```python
bert_model.input
```

```
[<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')>]
```

```python
bert_model.output
```

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

```python
# get the train output, BERT model will give one output so save in
# X_train_pooled_output
#this cell will take some time to execute, make sure thay you have stable internet connection
X_train_pooled_output=bert_model.predict([X_train_tokens,X_train_mask,X_train_segment])
```

```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])
```

```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'))
```

```python
#X_train_pooled_output, X_test_pooled_output= pickle.load(open('final_output.pkl', 'rb'))
```

Grader function 6

```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()
```

```
True
```

Part-5: Training a NN with 768 features

Create a NN and train the NN.
1. **You have to use AUC as metric. Do not use tf.keras.metrics.AUC**
 You have to write custom code for AUC and print it at the end of each epoch
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.

```python
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout, LSTM
from tensorflow.keras.models import Model
```

```python
##create an Neural Network and train your model on X_train_pooled_output and y_train
# you can start as follows
#input_layer=Input(shape=(X_train_pooled_output.shape[1],))
```

```python
from tensorflow.keras.layers import Input,Dense,Activation,Dropout,BatchNormalization
```

```python
class ReviewClassifier(Model):
  def __init__(self):
    super(ReviewClassifier, self).__init__()
    self.dense_1 = Dense(64, activation='relu')
    self.dense_2 = Dense(32, activation='relu')
    self.dense_3 = Dense(16, activation='relu')
    self.classify = Dense(1, activation='sigmoid')
    self.dropout_1 = Dropout(0.2)
    self.dropout_2 = Dropout(0.2)
    self.dropout_3 = Dropout(0.2)
  def call(self, inputs):
    x = self.dense_1(inputs)
    x = self.dropout_1(x)
    x = self.dense_2(x)
    x = self.dropout_2(x)
    x = self.dense_3(x)
    x = self.dropout_3(x)
    x = self.classify(x)
    return x
```

```python
review_classifier = ReviewClassifier()
review_classifier.compile(loss='binary_crossentropy', optimizer='adam')
review_classifier.build((None, 768))
review_classifier.summary()
```

```
Model: "review_classifier"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             multiple                  49216     
                                                                 
 dense_5 (Dense)             multiple                  2080      
                                                                 
 dense_6 (Dense)             multiple                  528       
                                                                 
 dense_7 (Dense)             multiple                  17        
                                                                 
 dropout_3 (Dropout)         multiple                  0         
                                                                 
 dropout_4 (Dropout)         multiple                  0         
                                                                 
 dropout_5 (Dropout)         multiple                  0         
                                                                 
=================================================================
Total params: 51,841
Trainable params: 51,841
Non-trainable params: 0
_________________________________________________________________
```

```python
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import (Callback, EarlyStopping,TensorBoard, ReduceLROnPlateau)
class ROCCallback(Callback):
  def __init__(self, validation_data):
    super(ROCCallback, self).__init__()
    self.validation_data = validation_data
  def on_epoch_end(self, epoch, logs={}):
    probs = self.model.predict(self.validation_data[0])
    y_true = self.validation_data[1]
    score = roc_auc_score(y_true, probs)
    logs['auc'] = score
```

```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
```

```python
roc_callback = ROCCallback((X_test_pooled_output, y_test))
early_stopping = EarlyStopping(patience=5)

tensorboard = TensorBoard()
reduce_lr = ReduceLROnPlateau(patience=3)
target = AchieveTarget(0.95)
callbacks = [roc_callback,early_stopping,tensorboard,reduce_lr,target,]
```

```python
!rm -rf ./logs/*


log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

history = review_classifier.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
   1/2500 [..............................] - ETA: 1:16 - loss: 0.1720WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0028s vs `on_train_batch_end` time: 0.0035s). Check your callbacks.

WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0028s vs `on_train_batch_end` time: 0.0035s). Check your callbacks.
2500/2500 [==============================] - 9s 4ms/step - loss: 0.2125 - val_loss: 0.1910 - auc: 0.9500 - lr: 1.0000e-04
```

```python
from keras.callbacks import Callback
from keras.callbacks import TensorBoard

import datetime
```

```python
logdir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=logdir, histogram_freq=1)
```

```python
%load_ext tensorboard
```

Part-6: Creating a Data pipeline for BERT Model

1. Pipeline is a way to codify and automate the workflow.
2. Download the test.csv file from here here </pre>

```python
#there is an alterante way to load files from Google drive directly to your Colab session
# you can use gdown module to import the files as follows
#for example for test.csv you can write your code as !gdown --id file_id (remove the # from next line and run it)
```

1. You have to write a function that takes the test_df,trained model and the required parameters as input.

1. Perform all the preproceesing steps inside the function.

- Remove all the html tags
- Now do tokenization [Part 3 as mentioned above]
- Create tokens,mask array and segment array
- 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)

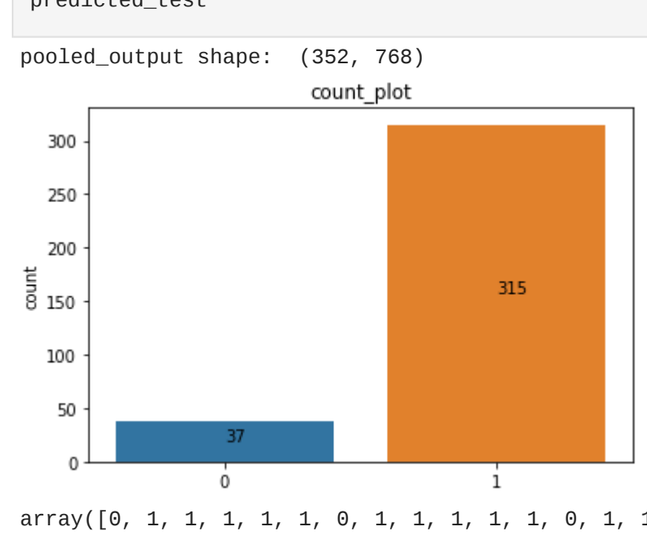
1. Predit the output of X_test with the neural network model which we trained earlier.

2. Return the occurences of class labels from the function.
   The output should be the count of datapoints classified as 1 or 0.

</pre>

```python
def pipeline():
  test=pd.read_csv('/content/test.csv')
  test['Text']=test['Text'].apply(lambda x :re.sub(r'<.*?>','',x))
  test_tokens,test_masks,test_segments=tokenizing_data(test['Text'],max_seq_length)
  test_pooled_output=bert_model.predict([test_tokens,test_masks,test_segments])
  print('pooled_output shape: ',test_pooled_output.shape)
  test_predict=review_classifier.predict(test_pooled_output)
  return np.where(test_predict<0.5,0,1)
predicted_test=np.array(tf.squeeze(pipeline()))
count=sean.countplot(x=predicted_test)
for i in np.unique(predicted_test):
  count.annotate((predicted_test==i).sum(),(i,(predicted_test==i).sum()/2))
plt.title('count_plot')
plt.show()
predicted_test
```

```
pooled_output shape:  (352, 768)
```




```
array([0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 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, 0, 1, 0, 1, 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, 1, 1, 0, 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, 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, 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, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 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])
```

```python
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["Test-Loss","Test - AUC"]
x.add_row([ "0.1910", "0.9500"])
print(x)
```

```
+-----------+------------+
| Test-Loss | Test - AUC |
+-----------+------------+
|   0.1910  |   0.9500   |
+-----------+------------+
```

Please write your observations at the end of notebook and explain each and every step you followed in solving this assignment.

1)As seen from the data analysis the data is highly unbalanced. Thus such a high accuracy and high AUC from start. Most of the data is labeled in class 1

2) During the classification Model training part I observe that the AUC of the model is high as >0.9 due to the BERT embedded vectors