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 [ ]:

```
# Downloading dataset zip file

!gdown --id 1ziSQS7bTrc0mV1IlYbzNpYrL452VVol-
```

```
Downloading...
From: https://drive.google.com/uc?id=1ziSQS7bTrc0mV1IlYbzNpYrL452VVol-
To: /content/Reviews.csv.zip
120MB [00:01, 108MB/s]
```

In [ ]:

```
# Unzipping file

!unzip 'Reviews.csv.zip'
```

```
Archive:  Reviews.csv.zip
  inflating: Reviews.csv
```

In [ ]:

```
#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
from tqdm import tqdm
import re
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

In [ ]:

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

Out[ ]:

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

## Grader function 1

In [ ]:

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

Out[ ]:

```
True
```

# Part-1: Preprocessing

In [ ]:

```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv("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 [ ]:

```
#get only 2 columns - Text, Score
#drop the NAN values
reviews = reviews[['Text', 'Score']]
reviews.dropna(inplace=True)
```

In [ ]:

```
#if score> 3, set score = 1
#if score<=2, set score = 0
#if score == 3, remove the rows.
reviews = reviews[reviews.Score != 3]
def partition(x):
    if x < 3:
        return 0
    return 1

scores = reviews['Score']
newScores = scores.map(partition)
reviews['Score'] = newScores
```

## Grader function 2

In [ ]:

```
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[ ]:

```
True
```

In [ ]:

```
# Adding column of word-length

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 [ ]:

```
#remove HTML from the Text column and save in the Text column only
def clean_text(x):
  return re.sub(r"http\S+", "", x)

reviews['Text'] = reviews.Text.apply(clean_text)
```

In [ ]:

```
#print head 5
reviews.head()
```

Out[ ]:

| | 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 |

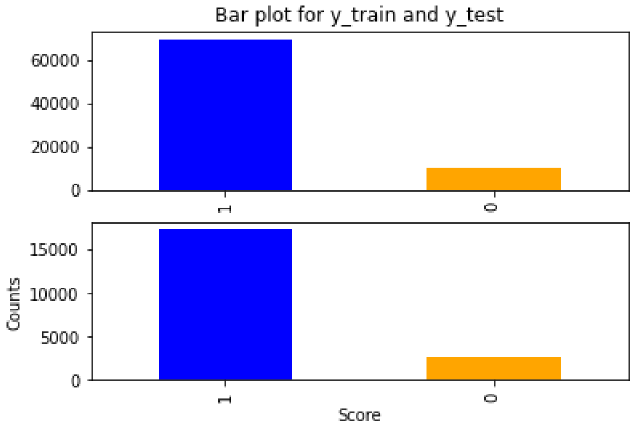
In [ ]:

```
#split the data into train and test data(20%) with Stratify sampling, random state 33,
X_train, X_test, y_train, y_test = train_test_split(reviews[['Text', 'len']], reviews['Score'], test_size

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(80000, 2)
(20000, 2)
(80000,)
(20000,)
```

In [ ]:

```
#plot bar graphs of y_train and y_test
plt.figure(1)

plt.subplot(211)
plt.title("Bar plot for y_train and y_test")
y_train.value_counts().plot(kind='bar', color=('blue','orange'))
plt.subplot(212)
y_test.value_counts().plot(kind='bar', color=('blue','orange'))

plt.xlabel("Score")
plt.ylabel("Counts")

plt.show()
```



We have more positive reviews than negative reviews.

In [ ]:

```
#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.

In [ ]:

```
## 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=F
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 [ ]:

```
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 [ ]:

```
bert_model.output
```

Out[ ]:

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

# Part-3: Tokenization

In [ ]:

```
#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 [ ]:

```
#import tokenization - We have given tokenization.py file
!pip install sentencepiece
from tokenization import FullTokenizer
```

```
Collecting sentencepiece
  Downloading
https://files.pythonhosted.org/packages/14/67/e42bd1181472c95c8cda79305df848264f2a7f62740995a46945d9797b6
tencepiece-0.1.95-cp36-cp36m-manylinux2014_x86_64.whl (1.2MB)
     |████████████████████████████████| 1.2MB 5.4MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.95
```

In [ ]:

```
# 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 = FullTokenizer(vocab_file, do_lower_case)
```

## Grader function 3

In [ ]:

```
#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[ ]:

```
True
```

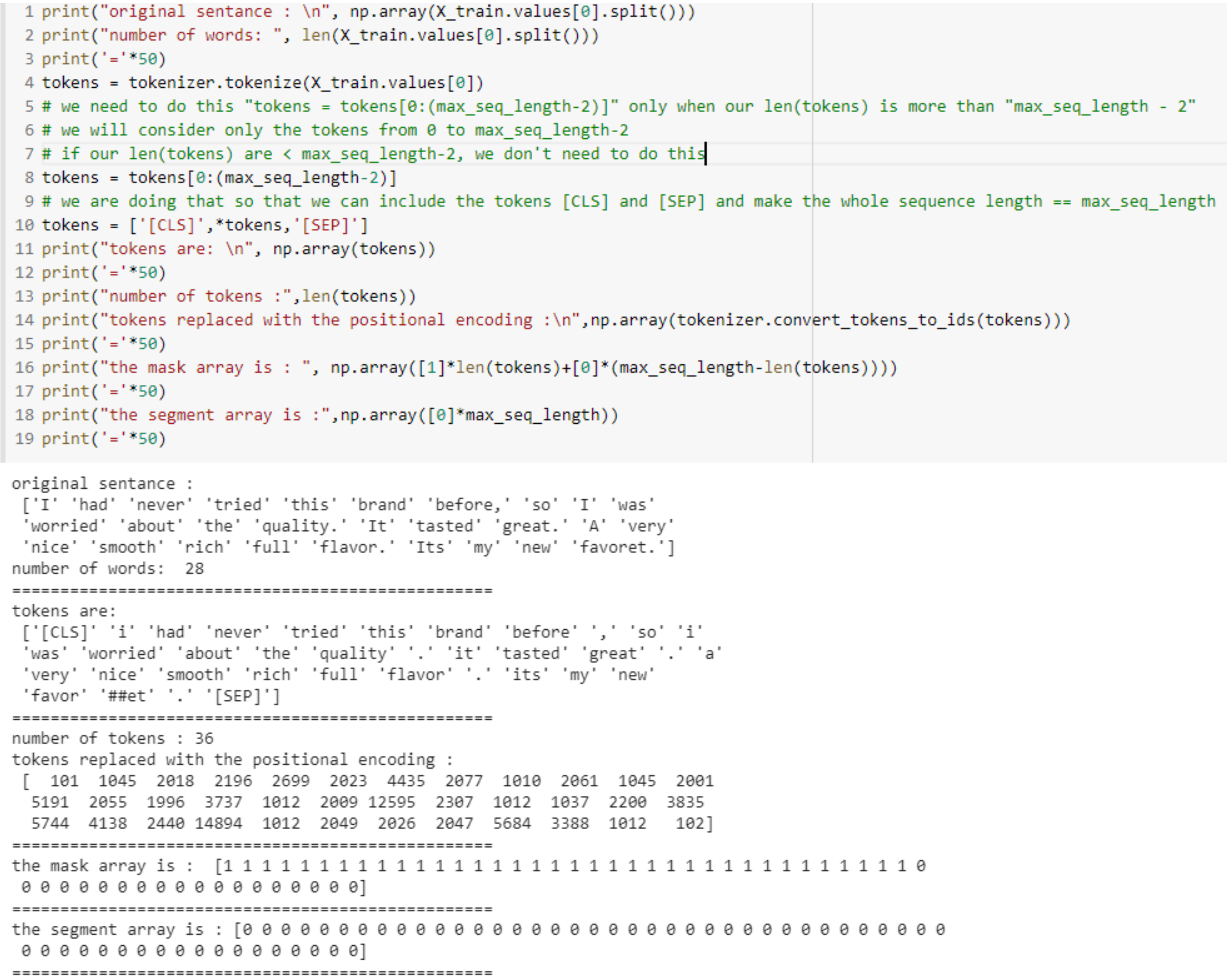
In [ ]:

```
# 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 wil

# 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 tokenize(texts):
  tokens = tokenizer.tokenize(text)
  if (len(tokens) < max_seq_length-2):
    tokens.extend(['[PAD]'] * (max_seq_length-2 - len(tokens)))
  elif (len(tokens) > max_seq_length-2):
    tokens = tokens[:(max_seq_length-2)]
  tokens = ['[CLS]', *tokens, '[SEP]']
  token_ids = np.array(tokenizer.convert_tokens_to_ids(tokens))
  mask = np.array([0 if i=='[PAD]' else 1 for i in tokens])
  segment = np.array([0]*max_seq_length)
  return token_ids, mask, segment

X_train_tokens = []
X_train_mask = []
X_train_segment = []
for text in tqdm(X_train['Text'].values):
  tokens, mask, segment = tokenize(text)
  X_train_tokens.append(tokens)
  X_train_mask.append(mask)
  X_train_segment.append(segment)

X_test_tokens = []
X_test_mask = []
X_test_segment = []
for text in tqdm(X_test['Text'].values):
  tokens, mask, segment = tokenize(text)
  X_test_tokens.append(tokens)
  X_test_mask.append(mask)
  X_test_segment.append(segment)

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)
```

```
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)
```

```
100%|██████████| 80000/80000 [00:35<00:00, 2246.73it/s]
100%|██████████| 20000/20000 [00:08<00:00, 2268.16it/s]
(80000, 55)
(80000, 55)
(80000, 55)
(20000, 55)
(20000, 55)
(20000, 55)
```

**Example**



```
print("original sentance : \n", np.array(X_train.values[0].split()))
print("number of words: ", len(X_train.values[0].split()))
print('='*50)
tokens = tokenizer.tokenize(X_train.values[0])
# we need to do this "tokens = tokens[0:(max_seq_length-2)]" only when our len(tokens) is more than "max_seq_length - 2"
# we will consider only the tokens from 0 to max_seq_length-2
# if our len(tokens) are < max_seq_length-2, we don't need to do this
tokens = tokens[0:(max_seq_length-2)]
# we are doing that so that we can include the tokens [CLS] and [SEP] and make the whole sequence length == max_seq_length
tokens = ['[CLS]',*tokens,'[SEP]']
print("tokens are: \n", np.array(tokens))
print('='*50)
print("number of tokens :",len(tokens))
print("tokens replaced with the positional encoding :\n",np.array(tokenizer.convert_tokens_to_ids(tokens)))
print('='*50)
print("the mask array is : ", np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens))))
print('='*50)
print("the segment array is :",np.array([0]*max_seq_length))
print('='*50)
```

```
original sentance :
 ['I' 'had' 'never' 'tried' 'this' 'brand' 'before,' 'so' 'I' 'was'
 'worried' 'about' 'the' 'quality.' 'It' 'tasted' 'great.' 'A' 'very'
 'nice' 'smooth' 'rich' 'full' 'flavor.' 'Its' 'my' 'new' 'favoret.']
number of words:  28
==================================================
tokens are:
 ['[CLS]' 'i' 'had' 'never' 'tried' 'this' 'brand' 'before' ',' 'so' 'i'
 'was' 'worried' 'about' 'the' 'quality' '.' 'it' 'tasted' 'great' '.' 'a'
 'very' 'nice' 'smooth' 'rich' 'full' 'flavor' '.' 'its' 'my' 'new'
 'favor' '##et' '.' '[SEP]']
==================================================
number of tokens : 36
tokens replaced with the positional encoding :
 [  101  1045  2018  2196  2699  2023  4435  2077  1010  2061  1045  2001
  5191  2055  1996  3737  1012  2009 12595  2307  1012  1037  2200  3835
  5744  4138  2440 14894  1012  2049  2026  2047  5684  3388  1012   102]
==================================================
the mask array is :  [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
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
==================================================
the segment array is : [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
==================================================
```

In [ ]:

```
import pickle
```

In [ ]:

```
##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 [ ]:

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

## Grader function 4

In [ ]:

```
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_lengt
        (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()
```

Out[ ]:

```
True
```

## Grader function 5

In [ ]:

```
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)
        (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[ ]:

```
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 [ ]:

```
bert_model.input
```

Out[ ]:

```
[<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 [ ]:

```
bert_model.output
```

Out[ ]:

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

In [ ]:

```
# 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 [ ]:

```
# 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 [ ]:

```
##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 [ ]:

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

## Grader function 6

In [ ]:

```
#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[ ]:

```
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 [ ]:

```
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import TensorBoard, Callback
import datetime
import os
```

In [ ]:

```
##create a NN on top of BERT model (It will take the BERT model output as its input)
# 7 dense layers are used with final output layer to create the NN model. Each layer uses regularizer to
# Dropout and Batch Normalization are used after few layers to avoid overfitting and normalize values re
# As its a deep NN, Batch Normalization helps to get better result
# As its a binary classification, 'sigmoid' activation function is used at output layer
# In other dense layers, 'relu' activation is used with 'he_normal' kernel initializer.

BERT_output = Input(shape=(768,), name="BERT_output")

dense1 = Dense(1024, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.0001), na
dense2 = Dense(512, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.0001), nam
dense3 = Dense(256, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.0001), nam
dropout1 = Dropout(0.5)(dense3)
bn1 = BatchNormalization()(dropout1)
dense4 = Dense(128, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.0001), nam
dense5 = Dense(64, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.0001), name
dense6 = Dense(32, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.0001), name
dense7 = Dense(16, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.0001), name
dropout2 = Dropout(0.5)(dense7)
bn2 = BatchNormalization()(dropout2)
output = Dense(1, activation='sigmoid', kernel_regularizer=l2(0.0001), name= 'output')(bn2)
```

```
model = Model(inputs=BERT_output, outputs=output)
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
BERT_output (InputLayer)     [(None, 768)]             0         
_________________________________________________________________
dense1 (Dense)               (None, 1024)              787456    
_________________________________________________________________
dense2 (Dense)               (None, 512)               524800    
_________________________________________________________________
dense3 (Dense)               (None, 256)               131328    
_________________________________________________________________
dropout (Dropout)            (None, 256)               0         
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024      
_________________________________________________________________
dense4 (Dense)               (None, 128)               32896     
_________________________________________________________________
dense5 (Dense)               (None, 64)                8256      
_________________________________________________________________
dense6 (Dense)               (None, 32)                2080      
_________________________________________________________________
dense7 (Dense)               (None, 16)                528       
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 16)                64        
_________________________________________________________________
output (Dense)               (None, 1)                 17        
=================================================================
Total params: 1,488,449
Trainable params: 1,487,905
Non-trainable params: 544
_________________________________________________________________
```

In [ ]:

```
# auc function to be passed as metric

def auc(y_true, y_pred):
  try:
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)
  except ValueError:
    return 0.5
```

In [ ]:

```
# Custom callback to be used for manipulating the trainig life-cycle

class My_callback(Callback):
    '''Custom callback to save best model, stop training if accuracy is not improving and compute F1-sco
    def __init__(self, epochs):
        self.epochs = epochs

    def save_best(self):
        # Saving the best model based on high val_accuracy
        best_val_auc = max(self.history['model'].keys())
        print("Saving best weights before terminating having val_auc {}".format(best_val_auc))
        best_model = self.history['model'].get(best_val_auc)
        filepath="model.hdf5"
        best_model.save(filepath)

    def on_train_begin(self, logs={}):
        # On begin of training, history dict is created with keys [model, loss, accuracy, val_loss, val_
        self.history={'model': dict(), 'val_auc': []}

    def on_epoch_end(self, epoch, logs={}):
        # In each epoch end, update loss, accuracy and F1-score
        loss = logs.get('loss')
        if (loss is not None):
            if (np.isnan(loss) or np.isinf(loss)):
                print('Invalid loss. Terminating at epoch {}'.format(epoch))
            else:
                # Record model at each epcoh with val_auc
                if logs.get('val_auc', -1) != -1:
                  self.history['val_auc'].append(logs.get('val_auc'))
                  self.history['model'][logs.get('val_auc')] = self.model
                # Stop training if val_auc doesnt improve for last 4 epochs
                if len(self.history['val_auc']) >= 4:
                        if ((self.history['val_auc'][-1] < self.history['val_auc'][-4]) and
                            (self.history['val_auc'][-1] < self.history['val_auc'][-3]) and
                          (self.history['val_auc'][-1] < self.history['val_auc'][-2])):
                                print("Validation auc not improving and terminated at epoch {}".format(ep
                                self.save_best()
                                self.model.stop_training = True

                # Finally save the best model at the end of training
                if epoch == self.epochs-1:
                    self.save_best()
```

In [ ]:

```
%load_ext tensorboard
```

In [ ]:

```
#Train NN with the BERT output values and Y-values (scores).
# Use 50 epcohs, Adam optimizer with 0.001 learning rate, accuracy and auc as evaluation metrics.
# Use binary_crossentropy as loss function
# tensorboard and callback object needs to be passed as callback

epochs = 50
logdir = os.path.join("logs")
tensorboard_callback = TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)
myCallback = My_callback(epochs)

opti = tf.keras.optimizers.Adam(learning_rate = 0.001)
model.compile(optimizer=opti,loss='binary_crossentropy',metrics=['accuracy', auc])
model.fit(X_train_pooled_output, y_train,epochs=epochs,
           validation_data=(X_test_pooled_output, y_test),
           batch_size=128, callbacks=[tensorboard_callback, myCallback])
```

```
Epoch 1/50
  3/625 [..............................] - ETA: 28s - loss: 1.2871 - accuracy: 0.4727 - auc: 0.4867
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time:
0.0053s vs `on_train_batch_end` time: 0.0132s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time:
0.0053s vs `on_train_batch_end` time: 0.0132s). Check your callbacks.
625/625 [==============================] - 5s 6ms/step - loss: 0.8903 - accuracy: 0.7044 - auc: 0.5012 -
val_loss: 0.5506 - val_accuracy: 0.8701 - val_auc: 0.5516
Epoch 2/50
625/625 [==============================] - 3s 5ms/step - loss: 0.5309 - accuracy: 0.8703 - auc: 0.5101 -
val_loss: 0.4777 - val_accuracy: 0.8701 - val_auc: 0.9143
Epoch 3/50
625/625 [==============================] - 3s 5ms/step - loss: 0.3715 - accuracy: 0.8806 - auc: 0.8720 -
val_loss: 0.2789 - val_accuracy: 0.8920 - val_auc: 0.9409
Epoch 4/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2970 - accuracy: 0.8991 - auc: 0.9100 -
val_loss: 0.3096 - val_accuracy: 0.8982 - val_auc: 0.9464
Epoch 5/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2611 - accuracy: 0.9077 - auc: 0.9260 -
val_loss: 0.8546 - val_accuracy: 0.5907 - val_auc: 0.9386
Epoch 6/50
625/625 [==============================] - 4s 6ms/step - loss: 0.2408 - accuracy: 0.9111 - auc: 0.9342 -
val_loss: 0.2211 - val_accuracy: 0.9259 - val_auc: 0.9421
Epoch 7/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2307 - accuracy: 0.9132 - auc: 0.9344 -
val_loss: 0.3465 - val_accuracy: 0.8749 - val_auc: 0.9499
Epoch 8/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2223 - accuracy: 0.9146 - auc: 0.9381 -
val_loss: 0.2428 - val_accuracy: 0.8974 - val_auc: 0.9476
Epoch 9/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2175 - accuracy: 0.9166 - auc: 0.9397 -
val_loss: 0.1993 - val_accuracy: 0.9176 - val_auc: 0.9521
Epoch 10/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2174 - accuracy: 0.9151 - auc: 0.9398 -
val_loss: 0.5699 - val_accuracy: 0.8704 - val_auc: 0.9516
Epoch 11/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2127 - accuracy: 0.9161 - auc: 0.9381 -
val_loss: 0.3734 - val_accuracy: 0.8741 - val_auc: 0.9514
Epoch 12/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2067 - accuracy: 0.9196 - auc: 0.9431 -
val_loss: 0.3455 - val_accuracy: 0.8753 - val_auc: 0.9530
Epoch 13/50
625/625 [==============================] - 3s 5ms/step - loss: 0.2072 - accuracy: 0.9193 - auc: 0.9418 -
val_loss: 0.1891 - val_accuracy: 0.9317 - val_auc: 0.9508
Validation auc not improving and terminated at epoch 13
Saving best weights before terminating having val_auc 0.9529764652252197
```

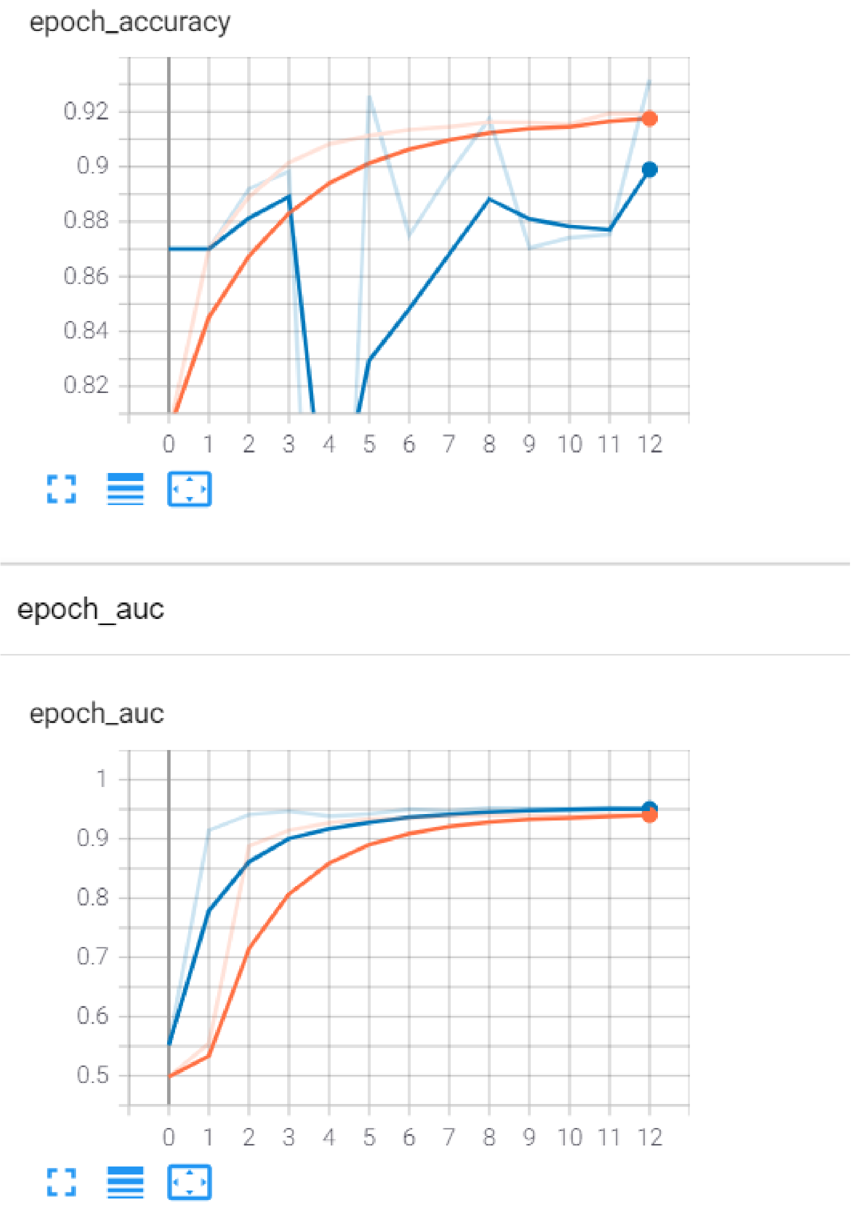
Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f4d329ef3c8>
```

The model got trained and the best weight values are saved that gave 0.9529 val_auc. Now when we use this model after BERT, it can give better AUC score.

In [ ]:

```
%tensorboard --logdir logs
```



The training and validaion metrics got improved with each epoch and finally AUC score reached 0.95.

# 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 [ ]:

```
# Building a pipeline where we can give the test data, new NN model and pre_trained BERT model as input.
# Threshold value will be passed to convert probability score to final output with high confidence.
# it will clean the input data and predict response with BERT model.
# The output will be fed into new NN model to get finally predicted output (Y_pred)

def predictBERT(data, model, bertModel, threshold):
  data['Text'] = data.Text.apply(clean_text)
  X = data['Text'].values
  X_tokens = []
  X_mask = []
  X_segment = []
  for text in tqdm(X):
    tokens, mask, segment = tokenize(text)
    X_tokens.append(tokens)
    X_mask.append(mask)
    X_segment.append(segment)
  X_tokens = np.array(X_tokens)
  X_mask = np.array(X_mask)
  X_segment = np.array(X_segment)
  bert_output = bertModel.predict([X_tokens, X_mask, X_segment])
  y_pred_prob = model.predict(bert_output)
  y_pred = []
  for y in y_pred_prob:
    if (y > threshold):
      y_pred.append(1)
    else:
      y_pred.append(0)
  return np.array(y_pred)
```

In [ ]:

```
# Load test data from file
# Load the best weights we got during training to our NN
# Put a threshold to determine the final predicted review score

test_data = pd.read_csv("test.csv")
model.load_weights('model.hdf5')
threshold = 0.9
predictBERT(test_data, model, bert_model, threshold)
```

```
100%|██████████| 352/352 [00:00<00:00, 1655.33it/s]
```

Out[ ]:

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

Here all the text reviews got classified into positive score. We can accept this predicted value as our model gives 0.95 validation AUC score.

In [ ]:

```
!zip -r logs.zip logs/
```

```
  adding: logs/ (stored 0%)
  adding: logs/validation/ (stored 0%)
  adding: logs/validation/events.out.tfevents.1613972224.47d3b28fe05c.76.33831.v2 (deflated 59%)
  adding: logs/train/ (stored 0%)
  adding: logs/train/events.out.tfevents.1613972220.47d3b28fe05c.76.27825.v2 (deflated 86%)
  adding: logs/train/events.out.tfevents.1613972221.47d3b28fe05c.profile-empty (deflated 5%)
  adding: logs/train/plugins/ (stored 0%)
  adding: logs/train/plugins/profile/ (stored 0%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/ (stored 0%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/47d3b28fe05c.overview_page.pb (deflated 57%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/47d3b28fe05c.tensorflow_stats.pb (deflated 74%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/47d3b28fe05c.trace.json.gz (deflated 1%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/47d3b28fe05c.memory_profile.json.gz (stored 0%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/47d3b28fe05c.input_pipeline.pb (deflated 55%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/47d3b28fe05c.xplane.pb (deflated 82%)
  adding: logs/train/plugins/profile/2021_02_22_05_37_01/47d3b28fe05c.kernel_stats.pb (deflated 96%)
```