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.

```
#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
```

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

```
''
```

Grader function 1

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

```
True
```

Part-1: Preprocessing

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv("/content/drive/My Drive/BERT/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
```

```
len(reviews)
```

```
568454
```

```
memory usage: 43.4+ MB
```

```
#get only 2 columns - Text, Score
data = reviews[['Text','Score']]
print(data.head())
print("Length of data:", len(data))
#drop the NAN values
data.dropna()
print("Length of data after removing Nan values:", len(data))
```

```
                                                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
Length of data: 568454
Length of data after removing Nan values: 568454
```

```
#if score> 3, set score = 1
#if score<=2, set score = 0
#if score == 3, remove the rows.
for i,j in data.iterrows():
  if j['Score'] > 3:
   data.at[i,'Score'] = 1
  elif j['Score'] <=2:
    data.at[i,'Score'] = 0
  elif j['Score'] ==3:
    data.drop(i,inplace=True)
```

```
print(data.head())
```

```
                                                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
```

```
import pickle
with open("/content/drive/My Drive/BERT/data.pkl", "wb") as f:
      pickle.dump(data, f)
```

```
import pickle
with open("/content/drive/My Drive/BERT/data.pkl", "rb") as f:
      data = pickle.load(f)
```

```
reviews = data
```

Grader function 2

```
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
```

```
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)
```

```
#remove HTML from the Text column and save in the Text column only
import re
for i,j in reviews.iterrows():
  reviews.at[i,'Text',] = re.sub('<[^<]+?>', '', j['Text'])
```

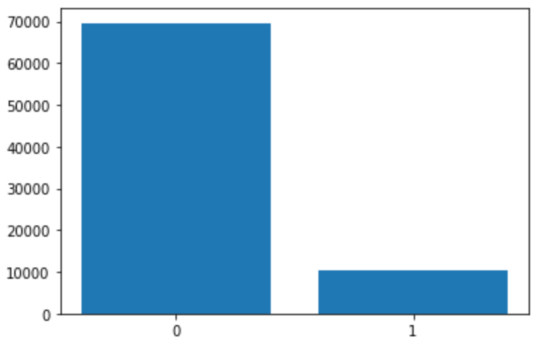
```
#print head 5
print(reviews.head(5))
```

```
                                                     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
```

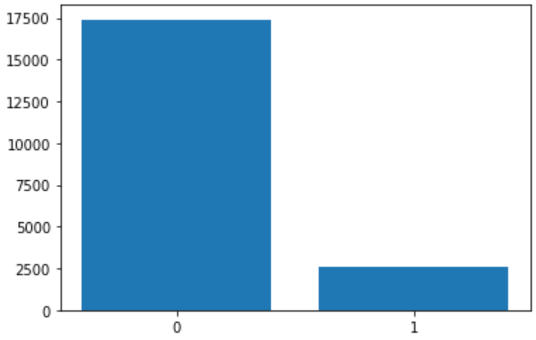
```
y = reviews['Score']
y.reset_index(drop = True,inplace=True)
X = reviews.drop(columns=['Score'])
X.reset_index(drop = True,inplace =True)
```

```
#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(X,y,test_size = 0.20, stratify = y,random_state = 33 )
```

```
#plot bar graphs of y_train and y_test
import matplotlib.pyplot as plt
fig = plt.figure()
vals = ['0','1']
plt.bar(vals,y_train.value_counts())
plt.show()
```



```
fig = plt.figure()
vals = ['0','1']
plt.bar(vals,y_test.value_counts())
plt.show()
```

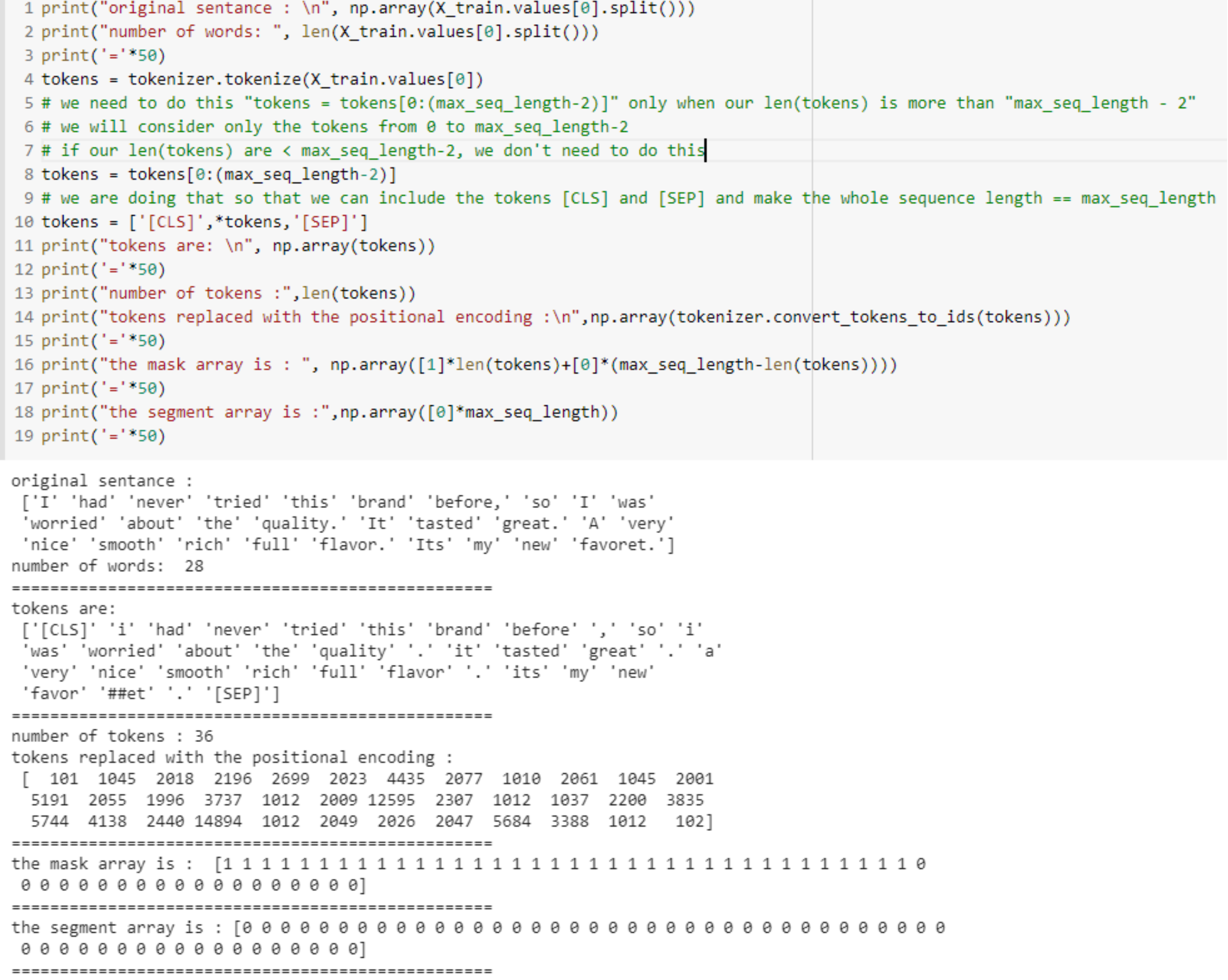


```
#saving to disk. if we need, we can load preprocessed data directly.
reviews.to_csv('/content/drive/My Drive/BERT/preprocessed.csv', index=False)
```

```
reviews = pd.read_csv('/content/drive/My Drive/BERT/preprocessed.csv')
```

```
    X_train_mask.append(arr)
    seg = [0]*55
    X_train_segment.append(seg)
#############################################################
for i, j in X_test.iterrows():
  temp = tokenizer.tokenize(j["Text"])
  if len(temp)>=53:
    temp = temp[0:53]
    temp = ['[CLS]'] + temp + ['[SEP]']
    temp = tokenizer.convert_tokens_to_ids(temp)
    X_test_tokens.append(temp)
    arr = [1]*55
    X_test_mask.append(arr)
    seg = [0]*55
    X_test_segment.append(seg)
  else:
    l = len(temp)
    temp = ['[CLS]'] + temp + ['[SEP]']
    temp = tokenizer.convert_tokens_to_ids(temp)
    arr = [1]*len(temp) + [0]*(55-len(temp))
    for k1 in range(55-len(temp)):
      temp.append(0)
    X_test_tokens.append(temp)

    X_test_mask.append(arr)
    seg = [0]*55
    X_test_segment.append(seg)


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)
```

▾ 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]
==================================================
```

```
import pickle


##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('/content/drive/My Drive/BERT/train_data.pkl','wb'))
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),open('/content/drive/My Drive/BERT/test_data.pkl','wb'))


#you can load from disk
import pickle
X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.load(open("/content/drive/My Drive/BERT/train_data.pkl", 'rb'))
X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(open("/content/drive/My Drive/BERT/test_data.pkl", 'rb'))
```

Grader function 4

```
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:
```

```
reviews = pd.read_csv('/content/drive/My Drive/BERT/preprocessed.csv')
```

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.

```
## 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)
```

```
bert_model.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
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
__________________________________________________________________________________________________
```

```
bert_model.output
```

```
<tf.Tensor 'keras_layer/StatefulPartitionedCall:0' shape=(None, 768) dtype=float32>
```

Part-3: Tokenization

```
#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()
```

```
pip install sentencepiece
```

```
Collecting sentencepiece
  Downloading https://files.pythonhosted.org/packages/d4/a4/d0a884c4300004a78cca907a6ff9a5e9fe4f090f5d95ab341c53d28cbc58/sentencepiece-0.1.91-c
     |████████████████████████████████| 1.1MB 4.3MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.91
```

```
#import tokenization - We have given tokenization.py file
%run '/content/drive/My Drive/BERT/tokenization.py'
```

```
# 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

```
#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
```

```
# 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
```

```
X_train_tokens =[] ; X_test_tokens = []; X_train_mask = [] ; X_test_mask=[];
X_train_segment =[] ;X_test_segment=[];
for i, j in X_train.iterrows():
  temp = tokenizer.tokenize(j["Text"])
  if len(temp)>=53:
    temp = temp[0:53]
    temp = ['[CLS]'] + temp + ['[SEP]']
    temp = tokenizer.convert_tokens_to_ids(temp)
    X_train_tokens.append(temp)
    arr = [1]*55
    X_train_mask.append(arr)
    seg = [0]*55
    X_train_segment.append(seg)
  else:
    l = len(temp)
    temp = ['[CLS]'] + temp + ['[SEP]']
    temp = tokenizer.convert_tokens_to_ids(temp)
    arr = [1]*len(temp) + [0]*(55-len(temp))
    for k1 in range(55-len(temp)):
      temp.append(0)
    X_train_tokens.append(temp)
```

```
        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

```
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_leng

        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()
```

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.

```
X_train_mask = np.asarray(X_train_mask).astype(np.float32)
X_train_segment = np.asarray(X_train_segment).astype(np.float32)
X_train_tokens = np.asarray(X_train_tokens).astype(np.float32)
```

```
# 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])
```

```
X_test_mask = np.asarray(X_test_mask).astype(np.float32)
X_test_segment = np.asarray(X_test_segment).astype(np.float32)
X_test_tokens = np.asarray(X_test_tokens).astype(np.float32)
```

```
# 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])
```

```
##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('/content/drive/My Drive/BERT/final_output.pkl','wb'))
```

```
X_train_pooled_output, X_test_pooled_output= pickle.load(open('/content/drive/My Drive/BERT/final_output.pkl', 'rb'))
```

Grader function 6

```
#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(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.**
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.

```
#https://datascience.stackexchange.com/questions/13490/how-to-set-class-weights-for-imbalanced-classes-in-keras
from sklearn.utils import class_weight
class_weights = {0:3.84726363,1:0.57468787}
#class_weights=class_weight.compute_class_weight('balanced',np.unique(y_train),y_train)
```

```
class_weight.compute_class_weight('balanced',np.unique(y_train),y_train)
```

```
array([3.84726363, 0.57468787])
```

```
from sklearn.metrics import roc_auc_score
def auc(y_true,y_pred):
  if len(np.unique(y_true))==1:
    return 0.5
  elif len(np.unique(y_test))==1:
    return 0.5
  else:
    return roc_auc_score(y_true,y_pred)

def auroc(y_true, y_pred): # https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-keras
    return tf.py_function(auc, (y_true, y_pred), tf.double)
```

```
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout
from tensorflow.keras.callbacks import TensorBoard,ReduceLROnPlateau
from tensorflow.keras.models import Model
import os,datetime
os.environ['PYTHONHASHSEED'] = '0'
tf.keras.backend.clear_session()
import warnings
```

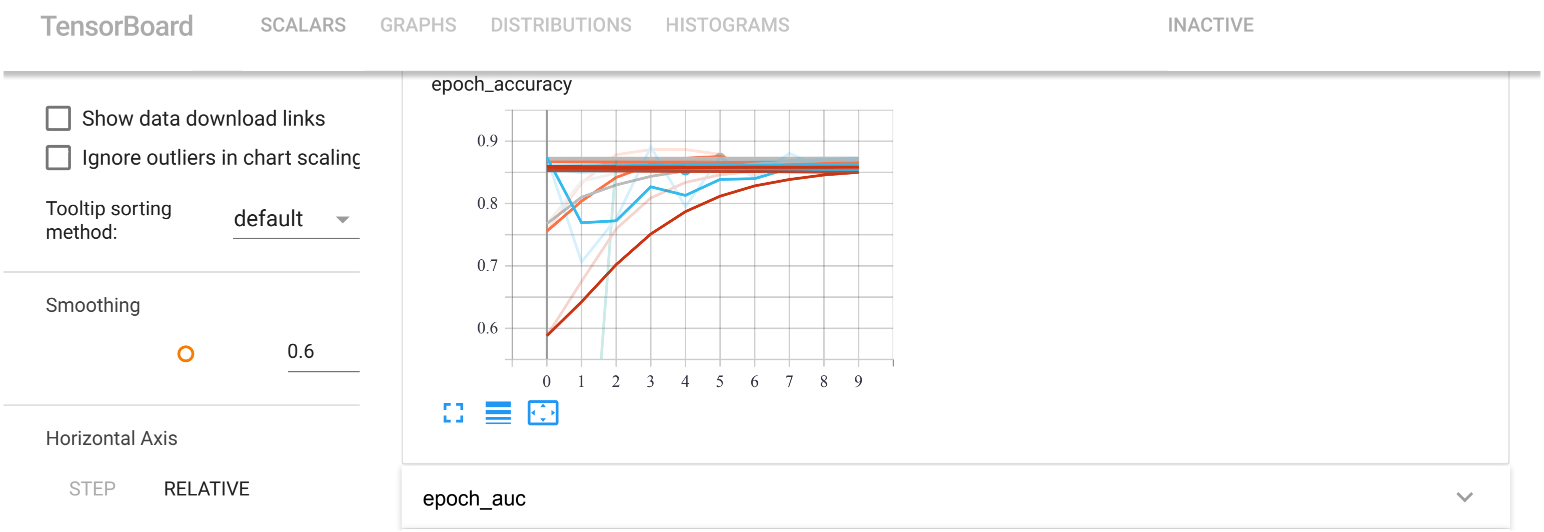
```
#https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
#optimizer = ['SGD', 'RMSprop',  'Adam'] ( already tried grid seaarch on this and found SGD works best)
batch_size = [10, 20, 40,60, 80, 100]
epochs = [10, 50, 80]
#learn_rate = [0.001, 0.000001, 0.0000001]
init_mode = ['uniform', 'lecun_uniform', 'normal', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']
neurons = [1,12,20,32]

from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from sklearn.model_selection import GridSearchCV
# Function to create model, required for KerasClassifier
def create_model(init_mode='uniform',batch_size=10,epochs=10,learn_rate=0.001, neurons=10 ):
  # create model
  model = Sequential()
  model.add(Dense(neurons, input_dim=768, activation='relu',kernel_initializer=init_mode, ))
  model.add(Dense(1, activation='sigmoid', kernel_initializer=init_mode))
  # Compile model
  model.compile(loss='binary_crossentropy',optimizer='SGD', metrics=['accuracy'])
  return model

model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
param_grid = dict(batch_size=batch_size, epochs =epochs,init_mode=init_mode,neurons=neurons )
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train_pooled_output, y_train, class_weight =class_weights)


##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout
from tensorflow.keras.callbacks import TensorBoard,ReduceLROnPlateau
from tensorflow.keras.models import Model
import os,datetime
os.environ['PYTHONHASHSEED'] = '0'
tf.keras.backend.clear_session()
import warnings
reduce_lr = ReduceLROnPlateau(monitor='auroc',factor=0.6,patience=1)
warnings.filterwarnings("ignore")
Input = Input(shape=(768,), name='Input')
model = Activation('relu')(Input)
model = Dense(64,activation='relu', kernel_initializer='he_uniform' )(model)
model = Dense(64,activation='relu',kernel_initializer='he_uniform')(model)
model = Dense(64,activation='relu',kernel_initializer='he_uniform')(model)
model = Dense(32,activation='relu',kernel_initializer='he_uniform')(model)

Output = Dense(1,activation='sigmoid',kernel_initializer='he_uniform' )(model)
model = Model(inputs=Input,outputs=Output)

checkpoint_path = "/content/drive/My Drive/BERT/my_model_weights.hdf5"
checkpoint_dir = os.path.dirname(checkpoint_path)
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath='/content/drive/My Drive/BERT/my_model_weights{epoch}.hdf5',  verbose=1, save_weights_only=
log_dir="/content/drive/My Drive/BERT/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True,write_grads=True)



sgd = tf.keras.optimizers.Adamax(learning_rate=0.01)
model.compile(loss='binary_crossentropy',optimizer =sgd ,metrics=[auroc,'accuracy'])
model.fit(X_train_pooled_output,y_train,batch_size = 64,validation_data=(X_test_pooled_output,y_test),epochs=10,callbacks=[reduce_lr,checkpoint, tens
     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.
     Epoch 1/10
        2/1250 [..............................] - ETA: 1:14 - loss: 1.3180 - auroc: 0.4936 - accuracy: 0.1797WARNING:tensorflow:Callbacks method `on
     WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0093s vs `on_train_batch_end` time:
     1248/1250 [============================>.] - ETA: 0s - loss: 0.4596 - auroc: 0.8830 - accuracy: 0.7675
     Epoch 00001: saving model to /content/drive/My Drive/BERT/my_model_weights1.hdf5
```

```
model.fit(X_train_pooled_output,y_train,batch_size = 64,validation_data=(X_test_pooled_output,y_test),epochs=10,callbacks=[reduce_lr,checkpoint, tens
```

```
Epoch 1/10
   2/1250 [..............................] - ETA: 1:26 - loss: 0.2992 - auroc: 0.9339 - accuracy: 0.9141WARNING:tensorflow:Callbacks method `on
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0096s vs `on_train_batch_end` time:
1242/1250 [============================>.] - ETA: 0s - loss: 0.2848 - auroc: 0.9501 - accuracy: 0.8729
Epoch 00001: saving model to /content/drive/My Drive/BERT/my_model_weights1.hdf5
1250/1250 [==============================] - 10s 8ms/step - loss: 0.2846 - auroc: 0.9502 - accuracy: 0.8729 - val_loss: 0.2981 - val_auroc: 0.9
Epoch 2/10
1241/1250 [============================>.] - ETA: 0s - loss: 0.2841 - auroc: 0.9502 - accuracy: 0.8723
Epoch 00002: saving model to /content/drive/My Drive/BERT/my_model_weights2.hdf5
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2844 - auroc: 0.9502 - accuracy: 0.8725 - val_loss: 0.3018 - val_auroc: 0.94
Epoch 3/10
1245/1250 [============================>.] - ETA: 0s - loss: 0.2842 - auroc: 0.9503 - accuracy: 0.8721
Epoch 00003: saving model to /content/drive/My Drive/BERT/my_model_weights3.hdf5
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2839 - auroc: 0.9504 - accuracy: 0.8722 - val_loss: 0.3133 - val_auroc: 0.94
Epoch 4/10
1244/1250 [============================>.] - ETA: 0s - loss: 0.2837 - auroc: 0.9505 - accuracy: 0.8719
Epoch 00004: saving model to /content/drive/My Drive/BERT/my_model_weights4.hdf5
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2836 - auroc: 0.9506 - accuracy: 0.8719 - val_loss: 0.3061 - val_auroc: 0.94
Epoch 5/10
1241/1250 [============================>.] - ETA: 0s - loss: 0.2832 - auroc: 0.9495 - accuracy: 0.8729
Epoch 00005: saving model to /content/drive/My Drive/BERT/my_model_weights5.hdf5
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2834 - auroc: 0.9495 - accuracy: 0.8730 - val_loss: 0.3073 - val_auroc: 0.94
Epoch 6/10
1245/1250 [============================>.] - ETA: 0s - loss: 0.2833 - auroc: 0.9506 - accuracy: 0.8726
Epoch 00006: saving model to /content/drive/My Drive/BERT/my_model_weights6.hdf5
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2834 - auroc: 0.9506 - accuracy: 0.8726 - val_loss: 0.3083 - val_auroc: 0.94
Epoch 7/10
1240/1250 [============================>.] - ETA: 0s - loss: 0.2835 - auroc: 0.9502 - accuracy: 0.8720
Epoch 00007: saving model to /content/drive/My Drive/BERT/my_model_weights7.hdf5
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2832 - auroc: 0.9504 - accuracy: 0.8720 - val_loss: 0.2961 - val_auroc: 0.94
Epoch 8/10
1249/1250 [============================>.] - ETA: 0s - loss: 0.2833 - auroc: 0.9501 - accuracy: 0.8731
Epoch 00008: saving model to /content/drive/My Drive/BERT/my_model_weights8.hdf5
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2832 - auroc: 0.9501 - accuracy: 0.8731 - val_loss: 0.3029 - val_auroc: 0.94
Epoch 9/10
1240/1250 [============================>.] - ETA: 0s - loss: 0.2828 - auroc: 0.9495 - accuracy: 0.8721
Epoch 00009: saving model to /content/drive/My Drive/BERT/my_model_weights9.hdf5
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2831 - auroc: 0.9494 - accuracy: 0.8719 - val_loss: 0.2981 - val_auroc: 0.94
Epoch 10/10
1241/1250 [============================>.] - ETA: 0s - loss: 0.2835 - auroc: 0.9500 - accuracy: 0.8734
Epoch 00010: saving model to /content/drive/My Drive/BERT/my_model_weights10.hdf5
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2831 - auroc: 0.9501 - accuracy: 0.8735 - val_loss: 0.3039 - val_auroc: 0.94
<tensorflow.python.keras.callbacks.History at 0x7fa5245b9860>
```

```
model.save('/content/drive/My Drive/BERT/')
%load_ext tensorboard
!kill 465
%tensorboard --logdir "/content/drive/My Drive/BERT/"
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
/bin/bash: line 0: kill: (465) - No such process
Reusing TensorBoard on port 6006 (pid 293), started 0:00:56 ago. (Use '!kill 293' to kill it.)
```



☐ Show data download links

☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis

STEP RELATIVE

```
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Input (InputLayer)           [(None, 768)]             0         
_________________________________________________________________
activation (Activation)      (None, 768)               0         
_________________________________________________________________
dense (Dense)                (None, 64)                49216     
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33        
=================================================================
Total params: 59,649
Trainable params: 59,649
Non-trainable params: 0
_________________________________________________________________
```

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
```

```
test_final = pd.read_csv('/content/drive/My Drive/BERT/test_final_part6.csv')
```

```
test_final[0:5]
```

	Text
0	Just opened Greenies Joint Care (individually ...
1	This product rocks :) My mom was very happy w/

```
import re
#https://stackoverflow.com/questions/9662346/python-code-to-remove-html-tags-from-a-string
def cleanhtml(raw_html):
  cleanr = re.compile('<.*?>')
  cleantext = re.sub(cleanr, '', raw_html)
  return cleantext
test_fin = []
for i in test_final['Text']:
  test_fin.append(cleanhtml(i))
```

```
test_fin[0:5]
```

```
['Just opened Greenies Joint Care (individually sealed) in December 2011 and found small worm crawling all over it.  Next one looked fine, but
 'This product rocks :) My mom was very happy w/the product it was excatly as described we loved seeing all the candy and eating it all :)',
 "The product was fine, but the cost of shipping was more than the costof the tea.  Won't make that mistake again.",
 "I love this soup. It's great as part of a meal or as a nutritious and satisifying low-in-calorie snack. For a light lunch, I stir in some shr
 "Getting ready to order again. These are great because they compost very quickly. BUT, some of the bags have tears in the bottom, so remember
```

```
X_test_fin_mask=[];
X_test_fin_segment=[];
X_test_fin_tokens=[];
for j in test_fin:
  temp = tokenizer.tokenize(j)
  if len(temp)>=53:
    temp = temp[0:53]
    temp = ['[CLS]'] + temp + ['[SEP]']
    temp = tokenizer.convert_tokens_to_ids(temp)
    X_test_fin_tokens.append(temp)
    arr = [1]*55
    X_test_fin_mask.append(arr)
    seg = [0]*55
    X_test_fin_segment.append(seg)
  else:
    l = len(temp)
    temp = ['[CLS]'] + temp + ['[SEP]']
    temp = tokenizer.convert_tokens_to_ids(temp)
    arr = [1]*len(temp) + [0]*(55-len(temp))
    for k1 in range(55-len(temp)):
      temp.append(0)
    X_test_fin_tokens.append(temp)

    X_test_fin_mask.append(arr)
    seg = [0]*55
    X_test_fin_segment.append(seg)
```

```
X_test_fin_mask = np.asarray(X_test_fin_mask).astype(np.float32)
X_test_fin_segment = np.asarray(X_test_fin_segment).astype(np.float32)
X_test_fin_tokens = np.asarray(X_test_fin_tokens).astype(np.float32)
```

```
X_test_fin_pooled_output=bert_model.predict([X_test_fin_tokens,X_test_fin_mask,X_test_fin_segment])
```

```
test_final.shape
```

```
(352, 1)
```

```
X_test_fin_pooled_output.shape
```

```
(352, 768)
```

```
#model.compile(loss='binary_crossentropy',optimizer =sgd ,metrics=[auroc,'accuracy'])
model.predict(X_test_fin_pooled_output)#(X_train_pooled_output,y_train,validation_data=(X_test_fin_pooled_output,y_test),epochs=20,callbacks=[reduce_
     array([[0.09261394],
            [0.98358333],
            [0.3295642 ],
            [0.36641663],
            [0.90176976],
            [0.1418013 ],
            [0.01603916],
            [0.9936055 ],
            [0.9910226 ],
            [0.98616207],
            [0.24247175],
            [0.90441906],
            [0.03613943],
            [0.6826369 ],
            [0.9308605 ],
            [0.02109239],
            [0.998559  ],
            [0.99325955],
            [0.91145754],
            [0.84758174],
            [0.49692056],
            [0.64441764],
            [0.5503431 ],
            [0.51497877],
            [0.90309507],
            [0.9719659 ],
            [0.80409425],
            [0.47489572],
            [0.34562814],
            [0.97433484],
            [0.05364725],
            [0.5527432 ],
            [0.94420755],
            [0.99466974],
            [0.73705107],
            [0.99833524],
            [0.29511094],
            [0.9961871 ],
            [0.9926808 ],
            [0.9959121 ],
            [0.01490715],
            [0.9939571 ],
            [0.97391045],
            [0.82650375],
            [0.0401386 ],
            [0.39573503],
            [0.91829324],
            [0.3559314 ],
            [0.96159565],
            [0.8630614 ],
            [0.9922832 ],
            [0.6731637 ],
            [0.9957177 ],
            [0.0324055 ],
            [0.08034998],
            [0.9789829 ],
            [0.8549325 ],
            [0.9403297 ],
            [0.9901848 ],
            [0.99765235],
            [0.9344524 ],
            [0.99969167],
            [0.8305807 ],
            [0.79009014],
            [0.8570354 ],
            [0.85591936],
            [0.99663717],
            [0.9926151 ],
            [0.98516786],
            [0.9714698 ],
            [0.9394655 ],
            [0.99794126],
            [0.493119  ],
            [0.99672586],
            [0.891539  ],
            [0.9184561 ],
            [0.9559604 ],
            [0.59937966],
            [0.9882234 ],
            [0.9911972 ],
            [0.9644444 ],
            [0.01427445],
            [0.71185595],
            [0.1338216 ],
            [0.08282512],
            [0.01699492],
            [0.26730105],
            [0.9988174 ],
            [0.9447145 ],
            [0.46052378],
            [0.81814706],
            [0.535938  ],
            [0.955146  ],
            [0.985412  ],
            [0.9690273 ],
            [0.98061526],
            [0.01816344],
            [0.3096192 ],
            [0.9336511 ],
            [0.99252367],
            [0.9819353 ],
            [0.9956696 ],
            [0.9903294 ],
            [0.14018837],
            [0.9448386 ],
            [0.96980965],
            [0.9911111 ],
            [0.98770857],
            [0.8119792 ],
            [0.8146183 ],
            [0.9944707 ],
            [0.99200785],
            [0.99835587],
            [0.98967034],
            [0.9864118 ],
            [0.94829416],
            [0.73371595],
            [0.8270116 ],
            [0.958928  ],
            [0.9643993 ],
            [0.99841124],
            [0.95353   ],
            [0.02696958],
            [0.9683702 ],
            [0.97988844],
            [0.9955385 ],
            [0.9757234 ],
            [0.9917741 ],
            [0.3817287 ],
            [0.34889787],
            [0.16350895],
            [0.14288625],
            [0.995229  ],
            [0.94037366],
            [0.94307727],
            [0.91057336],
            [0.5091499 ],
            [0.26437265],
            [0.9817636 ],
            [0.7645675 ],
            [0.94880813],
            [0.05465132],
            [0.6488952 ],
            [0.8323649 ],
            [0.9708655 ],
            [0.9856577 ],
            [0.9542023 ],
            [0.37220085],
            [0.94983983],
            [0.74165606],
            [0.99544966],
            [0.2246868 ],
            [0.9978613 ],
            [0.81493473],
            [0.82600856],
            [0.23843145],
            [0.9988983 ],
            [0.9249346 ],
            [0.12124363],
            [0.38579023],
            [0.99618614],
            [0.93453467],
            [0.947662  ],
            [0.9399698 ],
            [0.9810164 ],
            [0.992295  ],
            [0.89187527],
            [0.24864444],
            [0.9941852 ],
            [0.9959966 ],
            [0.78507614],
            [0.97320867],
            [0.26722437],
            [0.8933843 ],
            [0.02531016],
            [0.99335074],
            [0.79990864],
            [0.21258616],
            [0.9811654 ],
            [0.8637824 ],
            [0.0298081 ],
            [0.48816377],
            [0.9949062 ],
            [0.9396006 ],
            [0.9714895 ],
            [0.40808475],
            [0.09337029],
            [0.09478331],
            [0.97926486],
            [0.1346193 ],
            [0.9913163 ],
            [0.7159512 ],
            [0.90244234],
            [0.9986042 ],
            [0.24808946],
            [0.2964556 ],
            [0.66405594],
            [0.9895443 ],
            [0.7738496 ],
            [0.9919021 ],
            [0.9022745 ],
            [0.9967282 ],
            [0.7708167 ],
            [0.90106654],
            [0.9921218 ],
            [0.98720014],
            [0.99817   ],
            [0.6864863 ],
            [0.96848285],
            [0.7943082 ],
            [0.9830791 ],
            [0.9573217 ],
            [0.47908145],
            [0.9965394 ],
            [0.6678133 ],
            [0.8250985 ],
            [0.94584966],
            [0.98841536],
            [0.98619175],
            [0.98131895],
            [0.9889616 ],
            [0.10288456],
            [0.9749257 ],
            [0.99931264],
            [0.9950278 ],
            [0.1429677 ],
            [0.09266484],
            [0.37764424],
            [0.8758687 ],
            [0.99563867],
            [0.08845821],
            [0.9978293 ],
            [0.02449611],
            [0.9497987 ],
            [0.999473  ],
            [0.0162656 ],
            [0.96068966],
            [0.90925926],
            [0.77475846],
            [0.09346083],
            [0.44738507],
            [0.97995555],
            [0.9912846 ],
            [0.9891367 ],
            [0.5161625 ],
            [0.03507844],
            [0.81508374],
            [0.70825887],
            [0.86638504],
            [0.9670111 ],
            [0.01790833],
            [0.9901515 ],
            [0.98767275],
            [0.98094755],
            [0.96514046],
            [0.05978197],
            [0.9922044 ],
            [0.01913142],
            [0.0287559 ],
            [0.93931824],
            [0.9925282 ],
            [0.01791006],
            [0.20475781],
            [0.9876512 ],
            [0.72570527],
            [0.9923657 ],
            [0.9923657 ],
            [0.7758096 ],
            [0.9966024 ],
            [0.99097466],
            [0.9957997 ],
            [0.98727894],
            [0.12313509],
            [0.998997  ],
            [0.840925  ],
            [0.99022406],
            [0.90710354],
            [0.9766169 ],
            [0.9916594 ],
            [0.8484369 ],
            [0.64213645],
            [0.9964262 ],
            [0.93430513],
            [0.9495941 ],
            [0.9133235 ],
            [0.99787414],
            [0.99516296],
            [0.99332273],
            [0.86631477],
            [0.94693446],
            [0.9974062 ],
            [0.7310173 ],
            [0.05571589],
            [0.9277846 ],
            [0.976707  ],
            [0.61220217],
            [0.2679181 ],
            [0.98999816],
            [0.6227717 ],
            [0.9906833 ],
            [0.9612173 ],
            [0.9376713 ],
            [0.19750917],
            [0.2033244 ],
            [0.8718586 ],
            [0.3954281 ],
            [0.9493785 ],
            [0.6806934 ],
            [0.01122317],
            [0.7199875 ],
            [0.20097235],
            [0.62048775],
            [0.9791797 ],
            [0.99691707],
            [0.18384728],
            [0.9900043 ],
            [0.9765943 ],
            [0.86504054],
            [0.69494367],
            [0.9905957 ],
            [0.9932203 ],
            [0.9942424 ],
            [0.809901  ],
            [0.9620651 ],
            [0.10927233],
            [0.26619467],
            [0.44718766],
            [0.4059997 ],
            [0.0206897 ],
            [0.99619335],
            [0.28441733],
            [0.99465513],
            [0.2350344 ],
            [0.6886294 ],
            [0.9507673 ],
            [0.86847275],
            [0.9908328 ],
            [0.96128   ],
            [0.9652637 ],
            [0.99283993],
            [0.88719153],
            [0.19331747],
            [0.985057  ],
            [0.9957258 ],
            [0.9938971 ],
            [0.9721954 ],
            [0.8142438 ],
            [0.09951347],
            [0.96259934],
            [0.9696255 ],
            [0.85494196],
            [0.99082386],
            [0.9958991 ]], dtype=float32)
```

```
y_test_pr = model.predict(X_test_fin_pooled_output)
y_fin=[]
for i in y_test_pr:
  if i>0.5:
    y_fin.append(1)
  else:
    y_fin.append(0)
```

```
np.unique(y_fin)
```

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

```
pd.Series(y_fin).value_counts()
```

```
1    264
0     88
dtype: int64
```

Conclusion:

After preprocessing the data, I split it into train and test data. EDA on both train and test suggested high class imbalance.
BERT model was created and then using train and test data, padded sequence and masked vectors were obtained which passed as input to the BERT model, embeddings were obtained as output.
Then, A Neural Network was trained with above output and train AUC and validation AUC were seen to be 95.01,94.88 respectively.
Tensorboard plots were also obtained.

The neural network had the below parameters:

1. Adam Optimizer
2. Learning rate = 0.001
3. Initializer = He_Uniform
4. Activation = Relu
5. Metric = AUC, Accuracy

Finally, X_test was obtained from a data link and predicted with the above model.

Observations:

Train AUC: 95.01
Test AUC: 94.88
Train Accuracy: 87.35
Test Accuracy: 86.79