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 r

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 explai

3. please return outputs in the same format what we asked. Eg. Don't return List if

4. Please read the external links that we are given so that you will learn the conce

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

Every Grader function has to return True.

```
!wget --header="Host: doc-0k-bk-docs.googleusercontent.com" --header="User-Agent: Mozilla/
```

```
--2020-07-24 05:32:26--  https://doc-0k-bk-docs.googleusercontent.com/docs/securesc/4
Resolving doc-0k-bk-docs.googleusercontent.com (doc-0k-bk-docs.googleusercontent.com)
Connecting to doc-0k-bk-docs.googleusercontent.com (doc-0k-bk-docs.googleusercontent
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/csv]
Saving to: ‘Reviews.csv’

Reviews.csv             [              <=>    ] 286.96M  70.1MB/s    in 4.1s

2020-07-24 05:32:30 (70.1 MB/s) - ‘Reviews.csv’ saved [300904694]
```

```
#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()
```

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

Grader function 1

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

```
True
```

Part-1: Preprocessing

```
#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
```

```
reviews= reviews[['Text', 'Score']]
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

```
reviews.dropna()
```

	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
...	...	...
568449	Great for sesame chicken..this is a good if no...	5
568450	I'm disappointed with the flavor. The chocolat...	2
568451	These stars are small, so you can give 10-15 o...	5
568452	These are the BEST treats for training and rew...	5
568453	I am very satisfied ,product is as advertised,...	5

568454 rows × 2 columns

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


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


##if score == 3, remove the rows.
reviews= reviews.drop(reviews[reviews.Score == 3].index)
#shape after removing score==3
reviews.shape
```

```
(525814, 2)
```

```
#if score> 3, set score = 1
#if score<=2, set score = 0
def set_score(value):
    if value > 3:
        return 1
    else:
        return 0
```

```
        return 0
reviews['Score'] = reviews['Score'].apply(set_score)
```

```
reviews.shape
```

```
(525814, 2)
```

```
reviews.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

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947
Enter your authorization code:
··········
Mounted at /content/drive
```

Grader function 2

```
def grader_reviews():
    temp_shape = (reviews.shape == (525814, 2)) and (reviews.Score.value_counts()[1]==4437
    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)
```

```
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

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase



#remove HTML from the Text column and save in the Text column only
from bs4 import BeautifulSoup
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(reviews['Text'].values):
    clean = re.compile('<.*?>')
    sentance=re.sub(clean, '',sentance )
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    #https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split())
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 100000/100000 [00:23<00:00, 4300.70it/s]
```

```
reviews['Text']=preprocessed_reviews


#print head 5


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

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

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

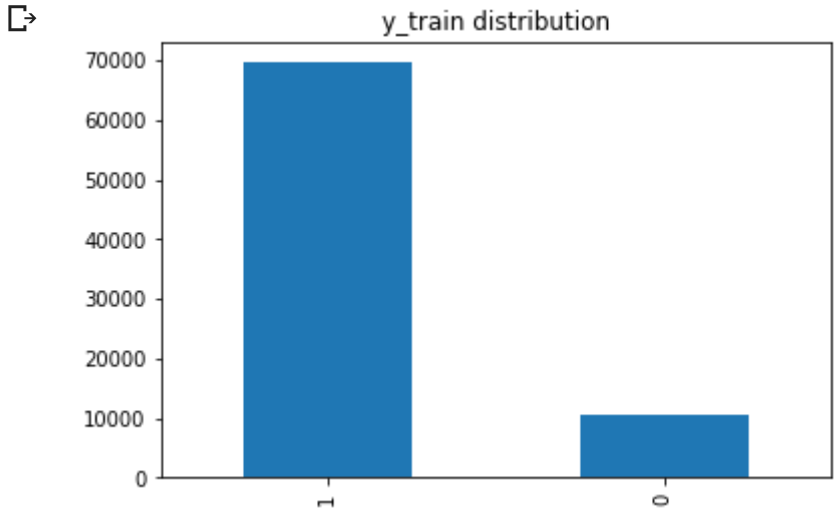
```
(80000,)
(80000,)
```

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

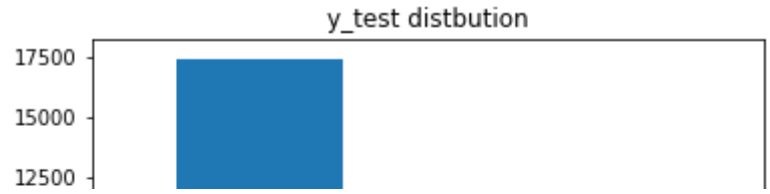
```
#plot bar graphs of y_train and y_test
```

```
import matplotlib.pyplot as plt
```

```
ax1 = y_train.value_counts().plot(kind='bar')
plt.title("y_train distribution")
plt.show()
```




```
ax2 = y_test.value_counts().plot(kind='bar')
plt.title('y_test distbution')
plt.show()
```




```
#saving to disk. if we need, we can load preprocessed data directly.
reviews.to_csv('drive/My Drive/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 BER 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

```
## 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 chang
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="inpu

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

#segment vectors. If you are giving only one sentence for the classification, total seg ve
#If you are giving two sentenced with [sep] token separated, first seq segment vectors are
#second seq segment vector are 1's
segment_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="segment

#bert layer
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/
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
bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_output
```

```
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
__________________________________________________________________________________________________
```

```
bert_model.output
```

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

Part-3: Tokenization

```
!pip install sentencepiece
```

```
Collecting sentencepiece
  Downloading https://files.pythonhosted.org/packages/d4/a4/d0a884c4300004a78cca907a6
     |████████████████████████████████| 1.1MB 7.1MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.91
```

```
from tokenization import FullTokenizer
```

```
#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()
```

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

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

```
# Create tokenizer " Instantiate FullTokenizer"
# name must be "tokenizer"
```

```
# 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
```

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) usin

# 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 (N

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

# 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_ma

# Create a segment input for train and test. We are using only one sentence so all zeros.

# 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.shape[0]
```

```
80000
```

```
max_seq_length=55
X_train_tokens=np.zeros((80000,55))
X_train_mask=np.zeros((80000,55))
X_train_segment=np.zeros((80000,55))
for i in tqdm(range(X_train.shape[0])):
  tokens=tokenizer.tokenize(X_train.values[i])
```

```
  tokens=['[CLS]',*tokens,'[SEP]']
  #print(max_seq_length-len(tokens)+len(tokens))
  #mask=np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens))
  if len(tokens) < max_seq_length:
        # Add the ['PAD'] token
        tokens = tokens + ['[PAD]' for item in range(max_seq_length-len(tokens))]
  else:
        # Truncate the tokens at maxLen - 1 and add a '[SEP]' tag.
        tokens = tokens[:max_seq_length-1] + ['[SEP]']
  #mask=[1 if x!=0 else 0 for x in tokens]
  tokens=np.array(tokenizer.convert_tokens_to_ids(tokens))
  mask=[1 if x!=0 else 0 for x in tokens]
  #mask=np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
  segment=np.array([0]*max_seq_length)
  X_train_tokens[i]=tokens
  X_train_mask[i]=np.array(mask)
  X_train_segment[i]=segment
```

```
100%|██████████| 80000/80000 [00:36<00:00, 2182.18it/s]
```

```
print(X_train_tokens.shape)
print(X_train_mask.shape)
print(X_train_segment.shape)
```

```
(80000, 55)
(80000, 55)
(80000, 55)
```

```
max_seq_length=55
X_test_tokens=np.zeros((X_test.shape[0],55))
X_test_mask=np.zeros((X_test.shape[0],55))
X_test_segment=np.zeros((X_test.shape[0],55))
for i in tqdm(range(X_test.shape[0])):
  tokens=tokenizer.tokenize(X_test.values[i])
  tokens=['[CLS]',*tokens,'[SEP]']
  #print(max_seq_length-len(tokens)+len(tokens))
  #mask=np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens))
  if len(tokens) < max_seq_length:
        # Add the ['PAD'] token
        tokens = tokens + ['[PAD]' for item in range(max_seq_length-len(tokens))]
  else:
        # Truncate the tokens at maxLen - 1 and add a '[SEP]' tag.
        tokens = tokens[:max_seq_length-1] + ['[SEP]']
  #mask=[1 if x!=0 else 0 for x in tokens]
  tokens=np.array(tokenizer.convert_tokens_to_ids(tokens))
  mask=[1 if x!=0 else 0 for x in tokens]
  #mask=np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
  segment=np.array([0]*max_seq_length)
  X_test_tokens[i]=tokens
  X_test_mask[i]=np.array(mask)
  X_test_segment[i]=segment
```

```
100%|██████████| 20000/20000 [00:09<00:00, 2189.32it/s]
```

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

```
(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 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]
==================================================
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 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('/conte
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),open('/content/dr
```

```
#you can load from disk
#X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.load(open("train
#X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(open("test_data.
```

```
with open('/content/drive/My Drive/train_data.pkl', 'rb') as f:
    X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.load(f)


with open('/content/drive/My Drive/test_data.pkl', 'rb') as f:
    X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(f)
```

Grader function 4

```
max_seq_length=55


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]
        (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

```
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]==
        (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.

```
bert_model.input
```

```
[<tf.Tensor 'input_word_ids:0' shape=(None, 55) dtype=int32>,
 <tf.Tensor 'input_mask:0' shape=(None, 55) dtype=int32>,
 <tf.Tensor 'segment_ids:0' shape=(None, 55) dtype=int32>]
```

```
bert_model.output
```

```
<tf.Tensor 'keras_layer/Identity:0' shape=(None, 768) dtype=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])
```

```
# 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])
```

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

```
#X_train_pooled_output, X_test_pooled_output= pickle.load(open('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(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 thos
4. Print the loss and metric at every epoch.
5. You have to submit without overfitting and underfitting.

```
X_test_pooled_output.shape
```

```
(20000, 768)
```

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y_train)
y_train_encoded = encoder.transform(y_train)
y_test_encoded = encoder.transform(y_test)
y_train_ohe = tf.keras.utils.to_categorical(y_train_encoded)
y_test_ohe = tf.keras.utils.to_categorical(y_test_encoded)
```

```
for i in range(1000):
  if y_test.iloc[i]==0:
    print(i)
    break
```

```
14
```

```
y_test_ohe.shape
```

```
(20000, 2)
```

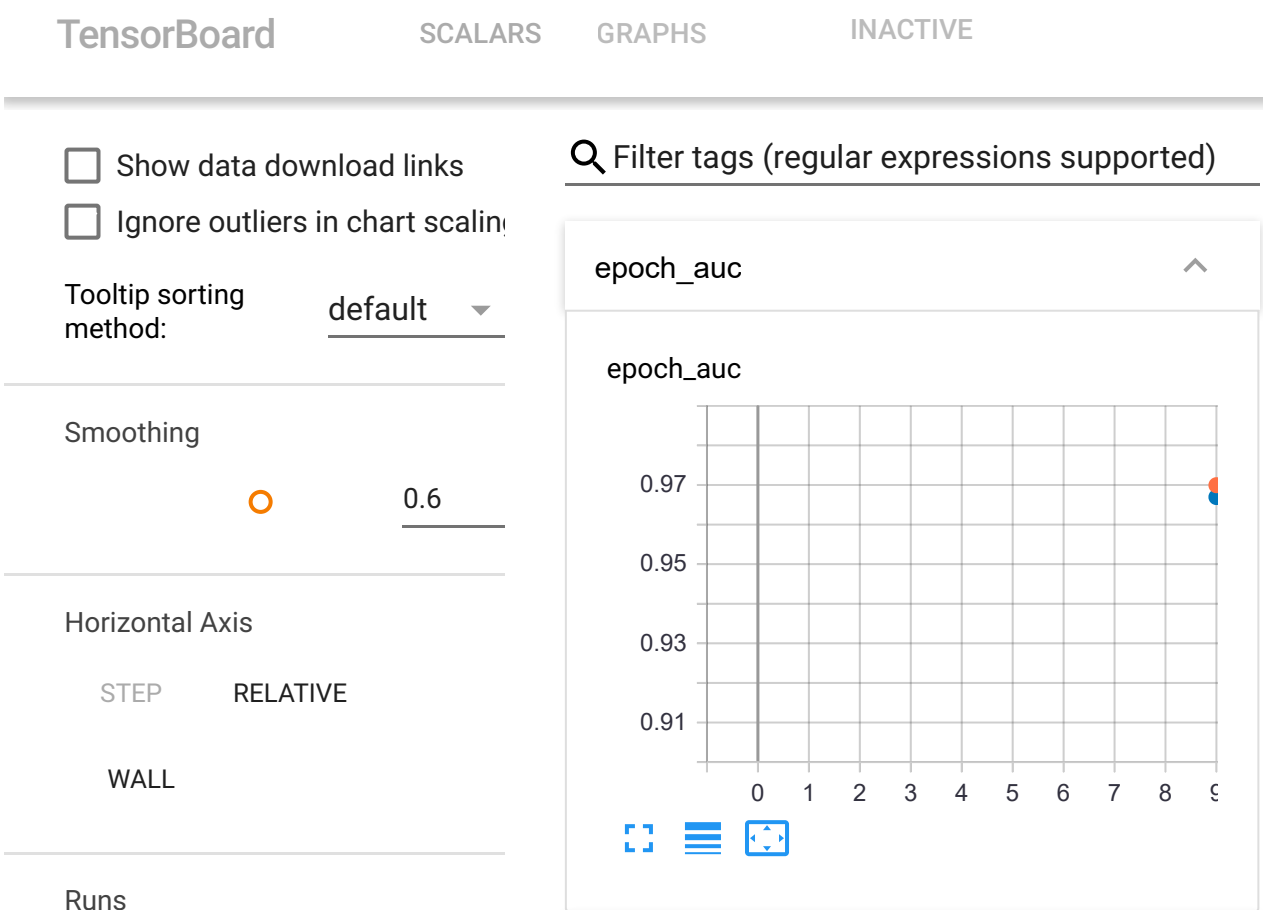
```
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout,BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import TensorBoard



#https://stackoverflow.com/questions/43263111/defining-an-auc-metric-for-keras-to-support-
def auc( y_true, y_pred ) :
    score = tf.py_function( lambda y_true, y_pred : roc_auc_score( y_true, y_pred, average
    return score


##create an NN and

#input layer
input1= Input(shape=(768,))
#danse layer1

dense1 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(se
drop1 =Dropout(0.2)(dense1)

#dense layer 2
dense2 = Dense(64,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop2=Dropout(0.2)(dense2)

# dense layer3
batch_norm = BatchNormalization()(drop2)
dense3 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop3= Dropout(0.2)(dense3)

# dense layer4

dense4 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop4= Dropout(0.2)(dense4)

#output layer
Out = Dense(units=2,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_n
model= Model(inputs=input1,outputs=Out)



model.summary()
```

```
Model: "model_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        [(None, 768)]             0
_________________________________________________________________
dense_52 (Dense)             (None, 128)               98432
_________________________________________________________________
dropout_52 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_53 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_53 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_13 (Batc (None, 64)                256
_________________________________________________________________
dense_54 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_54 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_55 (Dense)             (None, 32)                1056
_________________________________________________________________
dropout_55 (Dropout)         (None, 32)                0
_________________________________________________________________
Output (Dense)               (None, 2)                 66
=================================================================
Total params: 110,146
Trainable params: 110,018
```

```
#tf.keras.metrics.AUC(name='auc')
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001),loss='categorical_crossentropy
```

```
import os
import datetime
%load_ext tensorboard
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```



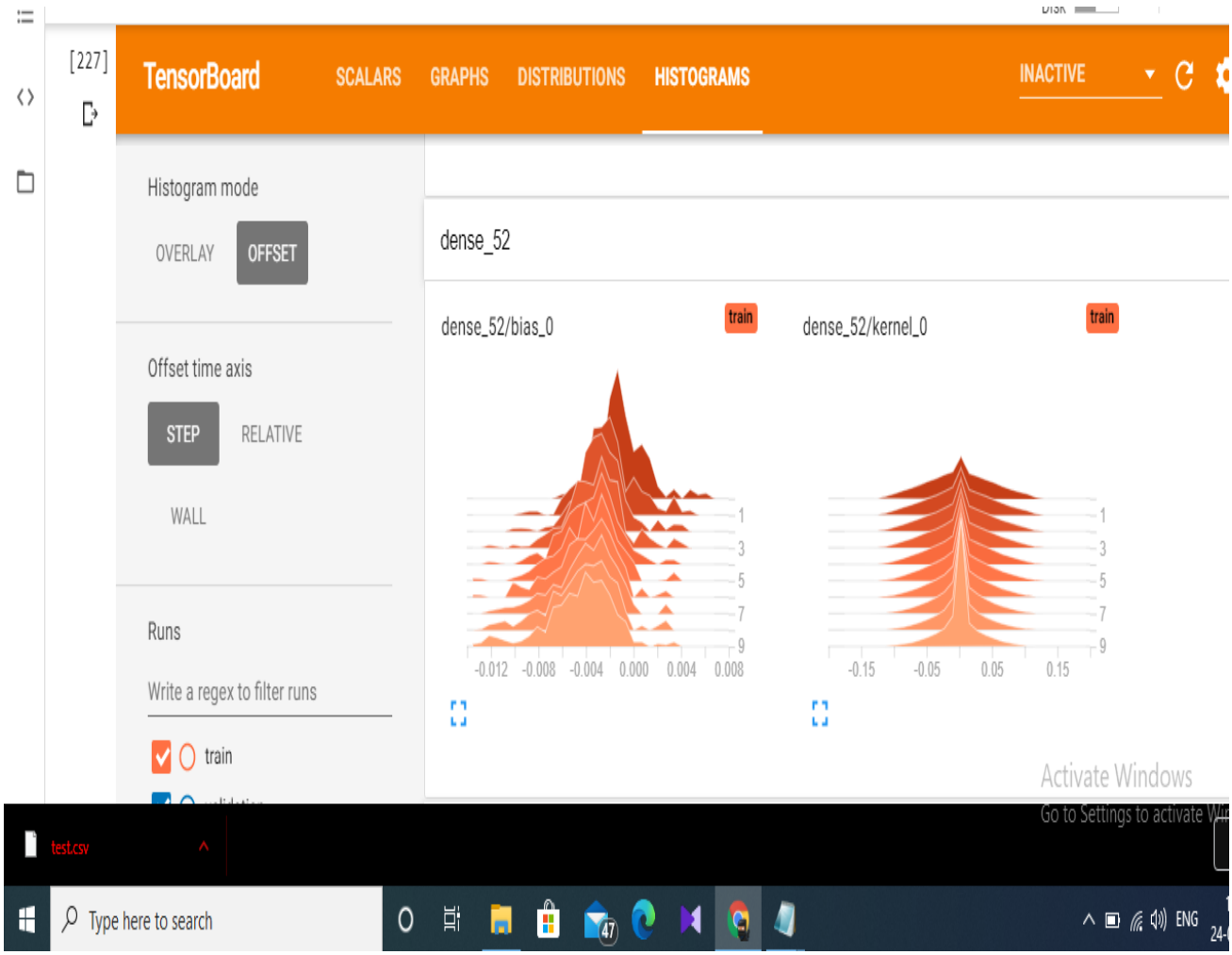

```
model.fit(X_train_pooled_output,y_train_ohe,epochs=10,batch_size=64,
validation_data=(X_test_pooled_output,y_test_ohe),callbacks=[tensorboard_callback])
```

```
Epoch 1/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.4917 - auc: 0.8756
Epoch 2/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.3650 - auc: 0.9377
Epoch 3/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.3155 - auc: 0.9552
Epoch 4/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2928 - auc: 0.9614
Epoch 5/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2774 - auc: 0.9653
Epoch 6/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2664 - auc: 0.9676
Epoch 7/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2577 - auc: 0.9696
Epoch 8/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2493 - auc: 0.9704
Epoch 9/10
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2430 - auc: 0.9713
Epoch 10/10
1250/1250 [==============================] - 8s 7ms/step - loss: 0.2361 - auc: 0.9722
<tensorflow.python.keras.callbacks.History at 0x7fe3e2fd9f60>
```

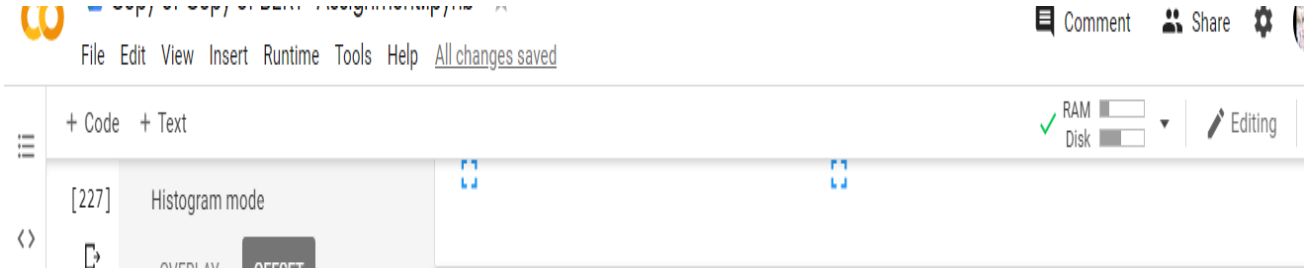
```
from IPython.display import Image
Image('/content/Screenshot (72).png',width=800,height=500)
```


```
Image('/content/Screenshot (73).png',width=800,height=500)
```

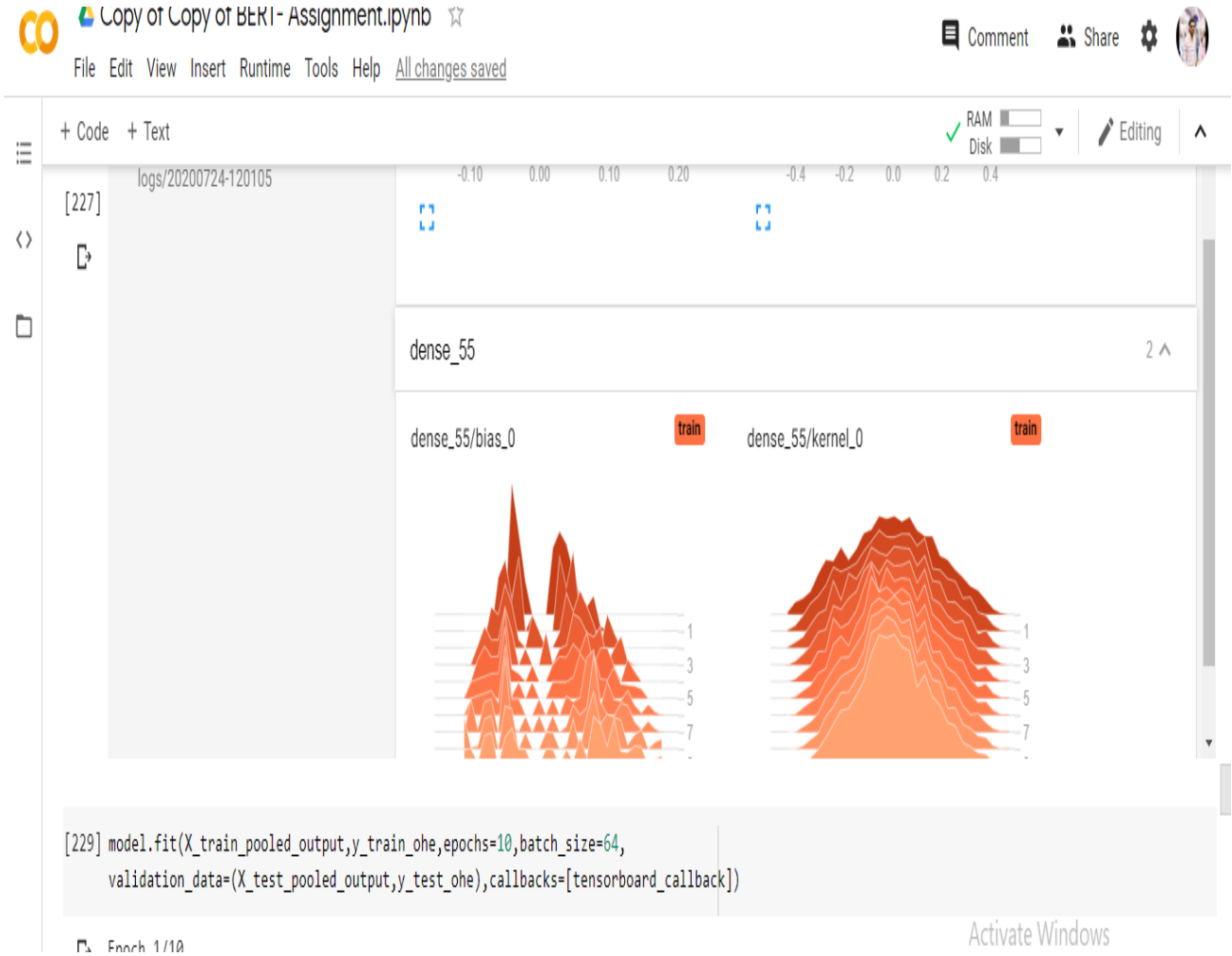
```
Image('/content/Screenshot (77).png',width=800,height=500)
```




```
Image('/content/Screenshot (78).png',width=800,height=500)
```




```
Image('/content/Screenshot (79).png',width=800,height=500)
```



Observation

1. For semtiment analysis of Amazon find food review we are using transfer learning with BERT uncased base line model.
2. On top of it we are using dense layer and observing in tensorboared.
3. AUC score of train and test data is almost close to each other so, model is not overfitting we can also see that loss is almost same for train and test data, So model is not overfiting.

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

```
X_test1 = pd.read_csv(r"test.csv")
#check the info of the dataset
X_test1.head()
```

	Text
0	Just opened Greenies Joint Care (individually ...
1	This product rocks :) My mom was very happy w/...
2	The product was fine, but the cost of shipping...
3	I love this soup. It's great as part of a meal...
4	Getting ready to order again. These are great ...

```
#remove HTML from the Text column and save in the Text column only
from bs4 import BeautifulSoup
from tqdm import tqdm
preprocessed_test = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test1['Text'].values):
    clean = re.compile('<.*?>')
    sentance=re.sub(clean, '',sentance )
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    #https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split())
    preprocessed_test.append(sentance.strip())
```

```
100%|██████████| 352/352 [00:00<00:00, 2929.66it/s]
```

```
X_test1['Text'] = preprocessed_test
```

```
X_test1.shape
```

```
(352, 1)
```

```
max_seq_length=55
X_test1_tokens=np.zeros((352,55))
X_test1_mask=np.zeros((352,55))
X_test1_segment=np.zeros((352,55))
for i in tqdm(range(X_test1.shape[0])):
  tokens=tokenizer.tokenize(X_test1['Text'].values[i])
  tokens=['[CLS]',*tokens,'[SEP]']
  #print(max_seq_length-len(tokens)+len(tokens))
  #mask=np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens))
  if len(tokens) < max_seq_length:
        # Add the ['PAD'] token
        tokens = tokens + ['[PAD]' for item in range(max_seq_length-len(tokens))]
  else:
        # Truncate the tokens at maxLen - 1 and add a '[SEP]' tag.
        tokens = tokens[:max_seq_length-1] + ['[SEP]']
  #mask=[1 if x!=0 else 0 for x in tokens]
  tokens=np.array(tokenizer.convert_tokens_to_ids(tokens))
  mask=[1 if x!=0 else 0 for x in tokens]
  #mask=np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
  segment=np.array([0]*max_seq_length)
  X_test1_tokens[i]=tokens
  X_test1_mask[i]=np.array(mask)
  X_test1_segment[i]=segment
```

```
100%|██████████| 352/352 [00:00<00:00, 2126.36it/s]
```

```
# get the train output, BERT model will give one output so save in
# X_train_pooled_output
X_test1=bert_model.predict([X_test1_tokens,X_test1_mask,X_test1_segment])
```

```
X_test1.shape
```

```
(352, 768)
```

```
Pred_output=model.predict(X_test1)
score_zero=0
score_one=0
for i in Pred_output[:,1]:
  if i > 0.5:
    score_one+=1

  else  :
    score_zero+=1
```

```
print('occurences of score zero is : ',score_zero)
print('occurences of score one is :',score_one)
```

```
occurences of score zero is :  44
occurences of score one is : 308
```

Observation:-

step 1 Data Preprocessing: First of all we have to do data preprocessing we are only considering review text and score field. --> We are dropping all the rows which are having scores=3,if scores is more than 3 then assign it with 1,if scores is less than 3 then assign it with 0. --> Text preprocessing is done with the help of regular expression and beautifulsoup such as decontraction removal of html tag and getting final text which is having only A to Z and a to z finally we are converting all the text into lower text.

step 2 Train text split: We are splitting the data into 80:20 ratio.

step 3 Creating BERT model: We are creating BERT baseline model with max input length=55, We are importing BERT model from hub.keras Layer.

step 4 Tokenizing and masking: Now we are tokenizing and masking review text and we are also creating X_Train segment.

step 5 Getting Embedding from BERT model: From BERT model we are passing tokenize mask and segment value and we are creating X_Train pooled output.

step 6 Creating Neural Network: In Neural Network we are giving X_Train pooled output and X_Text pooled output and we are getting 97% accuracy on text.

step 6 Creating data pipeline for BERT model: From the text.csv we are first creating token, mask and segment vector from that we are getting imbedded output from BERT model, That we are giving input in model.predict function from that we are finding occurence of 0 and 1.

```
from tabulate import tabulate
import pandas as pd
weather_data = [("BERT uncased Base model ", 'NN', 0.96)

               ]
df = pd.DataFrame(weather_data, columns=['Embeddings', 'Model', 'AUC'])


#Ref: https://pypi.org/project/tabulate/
print(tabulate(df, headers='keys', tablefmt='github'))
```

```
|    | Embeddings              | Model   |   AUC |
|----|-------------------------|---------|-------|
|  0 | BERT uncased Base model | NN      |  0.96 |
```