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 [ ]:

```
!pip install ipython-autotime
%load_ext autotime
```

```
Collecting ipython-autotime
  Downloading
https://files.pythonhosted.org/packages/e6/f9/0626bbdb322e3a078d968e87e3b01341e7890544de891d0cb6136
0e6/ipython-autotime-0.1.tar.bz2
Building wheels for collected packages: ipython-autotime
  Building wheel for ipython-autotime (setup.py) ... done
  Created wheel for ipython-autotime: filename=ipython_autotime-0.1-cp36-none-any.whl size=1831 sh
a256=9aad8a1eef47becf5e1c229e797f501a1306a756b710f0ee4ca1c7b9aed4e97e
  Stored in directory:
/root/.cache/pip/wheels/d2/df/81/2db1e54bc91002cec40334629bc39cfa86dff540b304ebcd6e
Successfully built ipython-autotime
Installing collected packages: ipython-autotime
Successfully installed ipython-autotime-0.1
```

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
```

```
time: 1.82 s
```

In [ ]:

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

Out[ ]:

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

```
time: 5.34 s
```

### Grader function 1

In [ ]:

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

Out[ ]:

```
True
```

```
time: 7.92 ms
```

# Part-1: Preprocessing

In [ ]:

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135 Safari/537.36" --header="A
ccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-data-
sets/18%2F2157%2Fbundle%2Farchive.zip?GoogleAccessId=gcp-kaggle-com@kaggle-
161607.iam.gserviceaccount.com&Expires=1598752807&Signature=QzbzV8Q1GXtivMKx7XiYaOKDMAsidky5HnYMmcC
QbmSTTU1AZh95esx1IBDuTTIzmQ3yHbmwUejOzeyjC8cIJR17CXWbk8TS2g1aqOqqq4i0vOK9plymr9X%2Bdpu%2BFwuhwX2EBI
vlnN4bYMmKt7M4q60GS5QhswIuJHW06cw8gEj04Aqcxu%2FGaYGIyoN3nhlUiiAOuF2nQ9QYY9%2F3W0LqftuHOlo3iKu416vxN
BO6WN2XWtcAnl08GpXeRNZ3sSRMAfbDpj6nKR8jITyv8HfIm%2B01f%2FO23NQ15LxtC1CwGGB5jpejSn%2FRIpjNLP7CJyoMMK
pHvA%3D%3D" -c -O '18_2157_bundle_archive.zip'
```

```
--2020-08-27 02:01:24--  https://storage.googleapis.com/kaggle-data-
sets/18%2F2157%2Fbundle%2Farchive.zip?GoogleAccessId=gcp-kaggle-com@kaggle-
161607.iam.gserviceaccount.com&Expires=1598752807&Signature=QzbzV8Q1GXtivMKx7XiYaOKDMAsidky5HnYMmcC
QbmSTTU1AZh95esx1IBDuTTIzmQ3yHbmwUejOzeyjC8cIJR17CXWbk8TS2g1aqOqqq4i0vOK9plymr9X%2Bdpu%2BFwuhwX2EBI
vlnN4bYMmKt7M4q60GS5QhswIuJHW06cw8gEj04Aqcxu%2FGaYGIyoN3nhlUiiAOuF2nQ9QYY9%2F3W0LqftuHOlo3iKu416vxN
BO6WN2XWtcAnl08GpXeRNZ3sSRMAfbDpj6nKR8jITyv8HfIm%2B01f%2FO23NQ15LxtC1CwGGB5jpejSn%2FRIpjNLP7CJyoMMK
pHvA%3D%3D
Resolving storage.googleapis.com (storage.googleapis.com)... 64.233.189.128, 108.177.97.128,
2404:6800:4008:c04::80, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|64.233.189.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 253873708 (242M) [application/zip]
Saving to: ‘18_2157_bundle_archive.zip’

18_2157_bundle_arch 100%[===================>] 242.11M  20.9MB/s    in 12s

2020-08-27 02:01:37 (20.9 MB/s) - ‘18_2157_bundle_archive.zip’ saved [253873708/253873708]

time: 13 s
```

In [ ]:

```
!unzip 18_2157_bundle_archive.zip
```

```
Archive:  18_2157_bundle_archive.zip
  inflating: Reviews.csv
  inflating: database.sqlite
  inflating: hashes.txt
time: 7.2 s
```

In [ ]:

```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv("/content/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
time: 3.48 s
```

In [ ]:

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

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

```
time: 75.8 ms
```

In [ ]:

```
reviews.dropna()
reviews.head(2)
```

Out[ ]:

| | Text | Score |
|---|---|---|
| **0** | I have bought several of the Vitality canned d... | 5 |
| **1** | Product arrived labeled as Jumbo Salted Peanut... | 1 |

```
time: 156 ms
```

In [ ]:

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

reviews.loc[reviews['Score'] < 3, 'Score'] = 0
reviews.loc[reviews['Score'] > 3, 'Score'] = 1
reviews = reviews.drop(reviews[reviews.Score == 3].index)
```

```
time: 78 ms
```

## 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
```

```
time: 20.2 ms
```

In [ ]:

```
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)
```

```
time: 2.42 s
```

In [ ]:

```
#remove HTML from the Text column and save in the Text column only
import re
for i in reviews['Text']:
  i=re.sub("[\<\(\[].*?[\)\]\>]", "", i)
```

```
time: 268 ms
```

In [ ]:

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

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 |

```
time: 22.7 ms
```

In [ ]:

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(reviews["Text"], reviews["Score"],stratify=revi
ews["Score"],train_size=0.80)
```

```
time: 510 ms
```

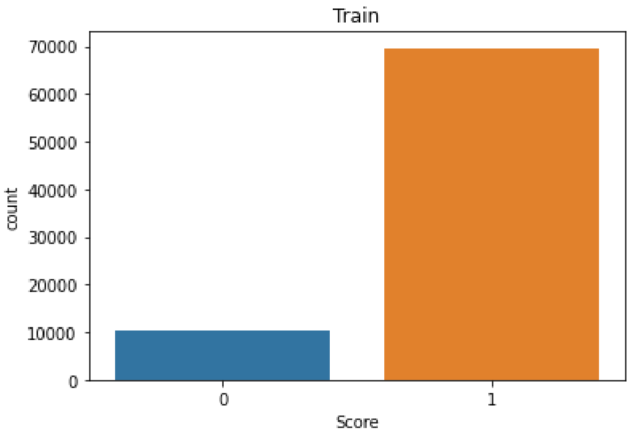
In [ ]:

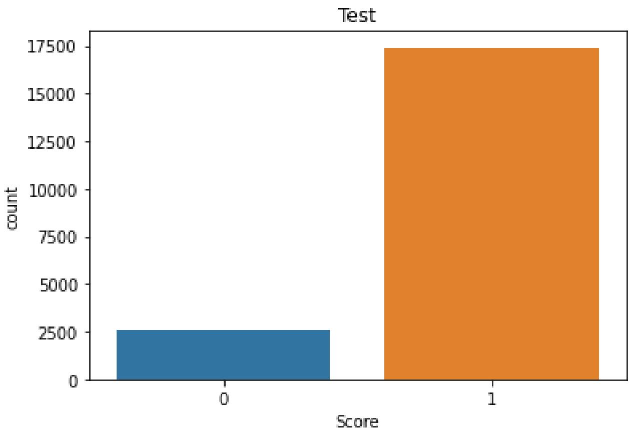
```
#plot bar graphs of y_train and y_test
import matplotlib.pyplot as plt
import seaborn as sns
y_train_plot=y_train.to_frame()
y_test_plot=y_test.to_frame()
```

```
y_test_plot=y_test.to_frame()
sns.countplot(x ='Score', data = y_train_plot)
plt.title("Train")
plt.show()

sns.countplot(x ='Score', data = y_test_plot)
plt.title("Test")
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```





```
time: 366 ms
```

In [ ]:

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

```
time: 1.65 s
```

# 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 [ ]:

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=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)
```

```
time: 18.8 s
```

In [ ]:

```
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
__________________________________________________________________________________________________
time: 20.9 ms
```

In [ ]:

```
bert_model.output
```

Out[ ]:

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

```
time: 6.03 ms
```

# 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()
```

```
time: 9.16 ms
```

In [ ]:

```
# While importing tokenization , it throws an error so, we have install the package sentencepiece
# ModuleNotFoundError: No module named 'sentencepiece'
!pip install sentencepiece
```

```
Collecting sentencepiece
  Downloading
https://files.pythonhosted.org/packages/d4/a4/d0a884c4300004a78cca907a6ff9a5e9fe4f090f5d95ab341c53c
c58/sentencepiece-0.1.91-cp36-cp36m-manylinux1_x86_64.whl (1.1MB)
     |████████████████████████████████| 1.1MB 2.8MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.91
time: 3.38 s
```

In [ ]:

```
!wget --header="Host: doc-14-5k-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135
Safari/537.36" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://drive.google.com/drive/folders/1RBD3mM5Ea5aH6HS6d32glqDMOJA0ciLm" --header="Coo
kie: AUTH_gnb78hdmdiks9t0b8kec09hpa7nncs5e_nonce=79hmk03b4fg0e; _ga=GA1.2.1804417035.1594643089; N
ID=204=WE4tnDNREeWA-
i7VkayXZRT0nZVp1JeYYk6hLWon_UP0rMptB4l1jfZkhNOvbLUPVxDowH066xA42Zz173_rsIlQAnYpv2qrlmQZ9MjqZVljYcd
wd0ZAN7SEaKwZ40sU9zdkP95PVkKfH4uFw5BkGh4qZanzjr9Y-b7iDM" --header="Connection: keep-alive"
"https://doc-14-5k-
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/1tucrmhioet355kq68274s05a
qbk/1598494200000/03515051603858730688/03515051603858730688/1-SiBIaHKwoSznjRd68TXpjTYyXl52bDM?e=do
wnload&authuser=0&nonce=79hmk03b4fg0e&user=03515051603858730688&hash=gf7v8pbslm7srs34pveq58vn05lqh
-c -O 'tokenization.py'
```

```
--2020-08-27 02:10:38--  https://doc-14-5k-
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/1tucrmhioet355kq68274s05a
qbk/1598494200000/03515051603858730688/03515051603858730688/1-SiBIaHKwoSznjRd68TXpjTYyXl52bDM?e=do
wnload&authuser=0&nonce=79hmk03b4fg0e&user=03515051603858730688&hash=gf7v8pbslm7srs34pveq58vn05lqhk

Resolving doc-14-5k-docs.googleusercontent.com (doc-14-5k-docs.googleusercontent.com)...
64.233.188.132, 2404:6800:4008:c06::84
Connecting to doc-14-5k-docs.googleusercontent.com (doc-14-5k-
docs.googleusercontent.com)|64.233.188.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 17318 (17K) [text/x-python]
Saving to: ‘tokenization.py’

tokenization.py     100%[===================>]  16.91K  --.-KB/s    in 0s

2020-08-27 02:10:38 (86.8 MB/s) - ‘tokenization.py’ saved [17318/17318]

time: 478 ms
```

In [ ]:

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

```
time: 13.9 ms
```

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=bert_tokenizer.FullTokenizer(vocab_file, do_lower_case)
```

```
time: 119 ms
```

## 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
```

```
time: 8.9 ms
```

In [ ]:

```
# Create train and test tokens (X_train_tokens, X_test_tokens) from (X_train, X_test) using Tokeni
zer 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 sha
pe 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
```

In [ ]:

```
import numpy as np
from keras.preprocessing.sequence import pad_sequences
```

```
time: 55.2 ms
```

In [ ]:

```
def tokenization(data):
  X_tokens=np.empty((0,55), int)
  X_mask=np.empty((0,55), int)
  X_segment=np.zeros([len(data),55])
  for idx,ele in enumerate(data):
    token = tokenizer.tokenize(ele)
```

```
      token = tokenizer.tokenize(ele)
      if (len(token)>max_seq_length-2) or (len(token)==max_seq_length-2):
        token=token[0:max_seq_length-2] #truncate
        token=['[CLS]',*token,'[SEP]'] #total length 55
        mask_token=np.array([1]*(len(token)))
      elif (len(token)<max_seq_length-2):
        mask_length=len(token)
        token=np.array(token+['[PAD]']*(max_seq_length-len(token)-2))
        token=['[CLS]',*token,'[SEP]'] #total length 55
        mask_token=np.array([1]*(mask_length+2)+[0]*(max_seq_length-mask_length-2))
      token=np.array(tokenizer.convert_tokens_to_ids(token))
      X_tokens = np.append(X_tokens, [token], axis=0)
      X_mask = np.append(X_mask, [mask_token], axis=0)
    return X_tokens,X_mask,X_segment
```

```
time: 21.5 ms
```

In [ ]:

```
X_train_tokens, X_train_mask, X_train_segment=tokenization(X_train)
```

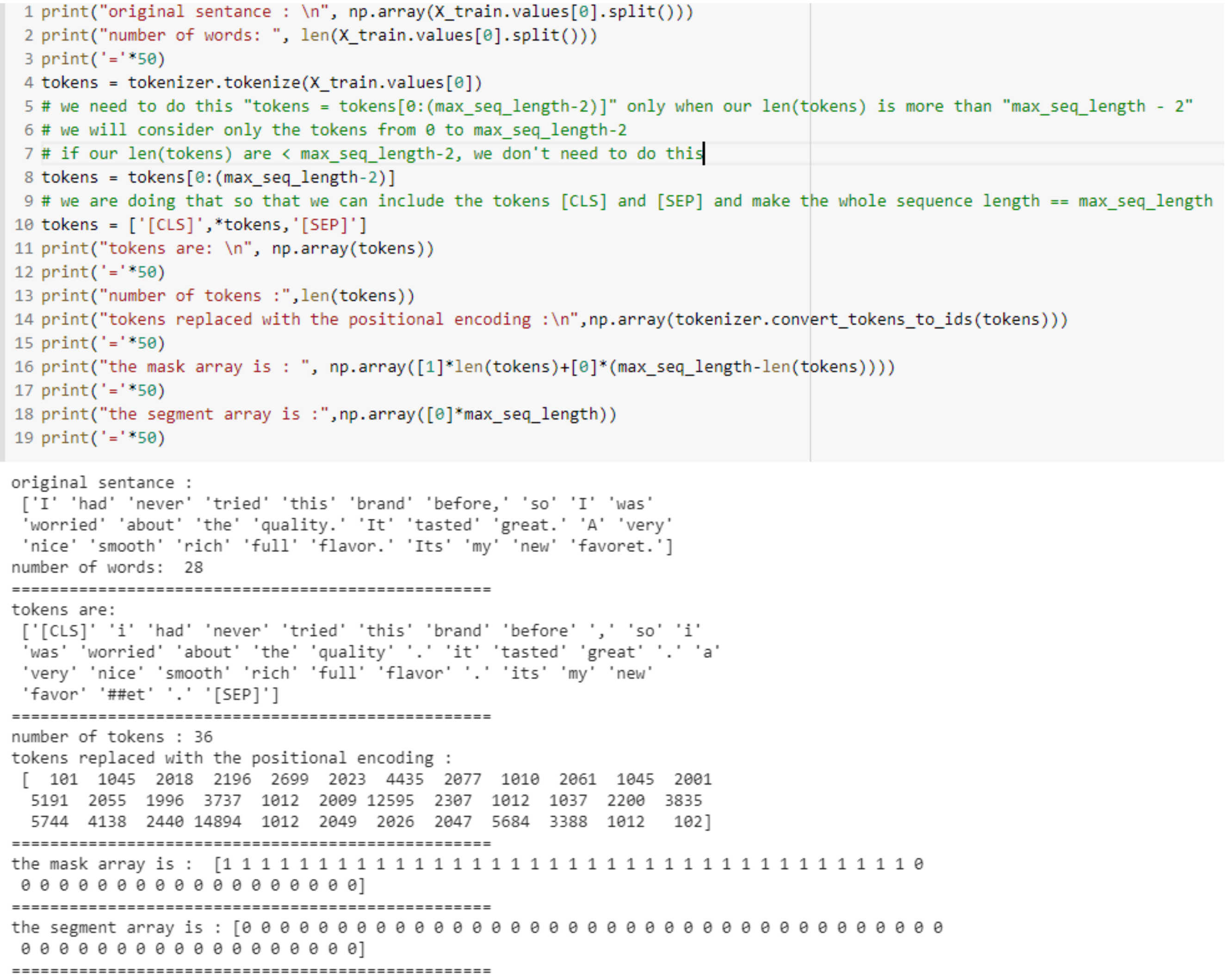
```
time: 11min 54s
```

In [ ]:

```
X_test_tokens, X_test_mask, X_test_segment=tokenization(X_test)
```

```
time: 40.6 s
```

### 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
```

```
time: 1.01 ms
```

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')
)
```

```
time: 365 ms
```

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", 'rb'))
#X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(open("test_data.pkl", 'r
b'))
```

```
time: 458 ms
```

## 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_length) and \
        (X_train_segment.shape[1]==max_seq_length)

        segment_temp = not np.any(X_train_segment)

        mask_temp = np.sum(X_train_mask==0) == np.sum(X_train_tokens==0)

        no_cls = np.sum(X_train_tokens==tokenizer.vocab['[CLS]'])==X_train_tokens.shape[0]

        no_sep = np.sum(X_train_tokens==tokenizer.vocab['[SEP]'])==X_train_tokens.shape[0]

        out = temp_shapes and segment_temp and mask_temp and no_cls and no_sep

    else:
        print('Type of all above token arrays should be numpy array not list')
        out = False
    assert(out==True)
    return out

grader_alltokens_train()
```

Out[ ]:

```
True
```

```
time: 53.4 ms
```

## 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) and \
        (X_test_segment.shape[1]==max_seq_length)

        segment_temp = not np.any(X_test_segment)

        mask_temp = np.sum(X_test_mask==0) == np.sum(X_test_tokens==0)

        no_cls = np.sum(X_test_tokens==tokenizer.vocab['[CLS]'])==X_test_tokens.shape[0]

        no_sep = np.sum(X_test_tokens==tokenizer.vocab['[SEP]'])==X_test_tokens.shape[0]

        out = temp_shapes and segment_temp and mask_temp and no_cls and no_sep

    else:
        print('Type of all above token arrays should be numpy array not list')
        out = False
    assert(out==True)
    return out
grader_alltokens_test()
```

Out[ ]:

```
True
```

```
time: 30.3 ms
```

## 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[ ]:

```
[<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>]
```

```
time: 5.12 ms
```

In [ ]:

```
bert_model.output
```

Out[ ]:

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

```
time: 8.52 ms
```

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])
```

```
time: 9min 27s
```

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])
```

```
time: 2min 20s
```

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'))
```

```
time: 860 ms
```

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
```

```
time: 13.8 ms
```

# 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
from tensorflow.keras.models import Model
import keras
from keras import models, layers
from keras.models import Model
from keras.layers import BatchNormalization, Activation, Flatten,Dense,Dropout,Input
from keras.optimizers import Adam,SGD
from keras.callbacks import Callback
from keras.initializers import HeNormal
from keras.initializers import HeNormal
from keras.layers import GlobalAveragePooling1D
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
import tensorflow
```

```
time: 6.67 ms
```

In [ ]:

```
input_Layer             =  layers.Input(shape=(768,))
dense_layer_1           =  layers.Dense(512,activation='relu')(input_Layer)
dropout_Layer           =  Dropout(0.2)(dense_layer_1)
dense_layer_2           =  layers.Dense(256,activation='relu')(dropout_Layer)
dense_layer_3           =  layers.Dense(64,activation='relu')(dense_layer_2)
drop_out_Layer          =  Dropout(0.5)(dense_layer_3)
dense_layer_4           =  layers.Dense(2,activation='softmax')(drop_out_Layer)
model = Model(inputs=[input_Layer], outputs=[dense_layer_4])
```

```
time: 84.3 ms
```

In [ ]:

```
model.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 768)]             0         
_________________________________________________________________
dense (Dense)                (None, 512)               393728    
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328    
_________________________________________________________________
dense_2 (Dense)              (None, 64)                16448     
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 130       
=================================================================
Total params: 541,634
Trainable params: 541,634
Non-trainable params: 0
_________________________________________________________________
time: 3.89 ms
```

In [ ]:

```
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=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.
2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fww
ogleapis.com%2fauth%2fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
time: 22.7 s
```

In [ ]:

```
import os
os.chdir('/content/drive/My Drive/Applied AI Course/Assignments/27. NLP with Transfer Learning')
!pwd
```

```
/content/drive/My Drive/Applied AI Course/Assignments/27. NLP with Transfer Learning
time: 1.01 s
```

In [ ]:

```
!mkdir -p logs/fit/
```

```
time: 165 ms
```

In [ ]:

```
!mkdir save_model
```

```
time: 160 ms
```

In [ ]:

```
##create an NN and

def NN(model,X_train_pooled_output,y_train,X_test_pooled_output,y_test):
  auc_metric=tf.keras.metrics.AUC(name="AUC")
  model.compile(loss='categorical_crossentropy',
                optimizer=Adam(),
                metrics=['accuracy',auc_metric])
  class Custom_callback(tensorflow.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
      auc = logs.get('AUC')
      if (auc > 0.97) :
        print("\n We have Reached the accepted criteria of AUC -::- {0} , so stopping training!!".
format(auc))
        self.model.stop_training = True

  filepath = "/content/drive/My Drive/Applied AI Course/Assignments/27. NLP with Transfer
Learning/save_model/{epoch:03d}.hdf5"
  model_chkpt = keras.callbacks.ModelCheckpoint(filepath, monitor='val_accuracy', verbose=0, save_b
est_only=True,mode='max')
  log_dir="/content/drive/My Drive/Applied AI Course/Assignments/27. NLP with Transfer
Learning/logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, write_gra
ph=True,write_grads=True)
  model.fit(X_train_pooled_output, y_train,
            batch_size=64,
            epochs=50,
            verbose=1,
            validation_data=(X_test_pooled_output, y_test),
            callbacks = [model_chkpt,Custom_callback(),tensorboard_callback])
  return model
```

```
time: 10.6 ms
```

## 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 [ ]:

```
!wget --header="Host: doc-14-bo-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135
Safari/537.36" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://drive.google.com/" --header="Cookie:
AUTH_gnb78hdmdiks9t0b8kec09hpa7nncs5e_nonce=8sol7b307osqe; _ga=GA1.2.1804417035.1594643089; NID=20
4=WE4tnDNREeWA-
i7VkayXZRT0nZVp1JeYYk6hLWon_UP0rMptB4l1jfZkhNOvbLUPVxDowH066xA42Zz173_rsIlQAnYpv2qrlmQZ9MjqZVljYcd
wd0ZAN7SEaKwZ40sU9zdkP95PVkKfH4uFw5BkGh4qZanzjr9Y-b7iDM" --header="Connection: keep-alive"
"https://doc-14-bo-
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/3n701fema2iljh0hh5v4pnun
0vm/1598496150000/00484516897554883881/03515051603858730688/1QwjqTsqTX2vdy7fTmeXjxP3dq8IAVLpo?e=do
wnload&authuser=0&nonce=8sol7b307osqe&user=03515051603858730688&hash=inqo8dmtikkboj8ic1q8qe4na64fa
-c -O 'test.csv'
```

```
--2020-08-27 02:43:46--  https://doc-14-bo-
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/3n701fema2iljh0hh5v4pnun9
0vm/1598496150000/00484516897554883881/03515051603858730688/1QwjqTsqTX2vdy7fTmeXjxP3dq8IAVLpo?e=do
wnload&authuser=0&nonce=8sol7b307osqe&user=03515051603858730688&hash=inqo8dmtikkboj8ic1q8qe4na64fa1

Resolving doc-14-bo-docs.googleusercontent.com (doc-14-bo-docs.googleusercontent.com)...
64.233.188.132, 2404:6800:4008:c06::84
Connecting to doc-14-bo-docs.googleusercontent.com (doc-14-bo-
docs.googleusercontent.com)|64.233.188.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 62100 (61K) [text/csv]
Saving to: ‘test.csv’

test.csv            100%[===================>]  60.64K  --.-KB/s    in 0.005s

2020-08-27 02:43:47 (11.2 MB/s) - ‘test.csv’ saved [62100/62100]

time: 675 ms
```

In [ ]:

```
test_csv = pd.read_csv("test.csv")
```

```
time: 18.3 ms
```

In [ ]:

```
test_data=test_csv['Text'] # convert dataframe to series pandas
print("Shape of test file:",test_data.shape)
```

```
Shape of test file: (352,)
time: 1.6 ms
```

In [ ]:

In [ ]:

```
# Remove all the html tag
for e in test_data:
  e=re.sub("[\<\(\[].*?[\)\]\>]", "",e)
```

```
time: 3.57 ms
```

In [ ]:

```
test_tokens, test_mask, test_segment=tokenization(test_data)
```

```
time: 240 ms
```

In [ ]:

```
test_pooled_output=bert_model.predict([test_tokens,test_mask,test_segment])
```

```
time: 2.69 s
```

In [ ]:

```
test_pooled_output.shape
```

Out[ ]:

```
(352, 768)
```

```
time: 3.24 ms
```

In [ ]:

```
ytrain_cat = keras.utils.to_categorical(y_train, num_classes=2)
ytest_cat = keras.utils.to_categorical(y_test, num_classes=2)
```

```
time: 4.68 ms
```

In [ ]:

```
%load_ext tensorboard
```

```
time: 15.3 ms
```

In [ ]:

```
import datetime
trained_model=NN(model,X_train_pooled_output,ytrain_cat,X_test_pooled_output,ytest_cat)
```

```
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 1/50
   1/1250 [..............................] - ETA: 0s - loss: 0.5900 - accuracy: 0.7812 - AUC: 0.76
79WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after 2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after 2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
```

```
   2/1250 [..............................] - ETA: 1:14 - loss: 0.5092 - accuracy: 0.8203 - AUC: 0.
8275WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time (b
atch time: 0.0121s vs `on_train_batch_end` time: 0.1071s). Check your callbacks.
```

```
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time (batch
time: 0.0121s vs `on_train_batch_end` time: 0.1071s). Check your callbacks.
```

```
1250/1250 [==============================] - 8s 6ms/step - loss: 0.3108 - accuracy: 0.8812 - AUC:
0.9400 - val_loss: 0.2566 - val_accuracy: 0.9038 - val_AUC: 0.9652
Epoch 2/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2642 - accuracy: 0.8958 - AUC:
0.9571 - val_loss: 0.2422 - val_accuracy: 0.8985 - val_AUC: 0.9645
Epoch 3/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2558 - accuracy: 0.8986 - AUC:
0.9599 - val_loss: 0.2355 - val_accuracy: 0.9020 - val_AUC: 0.9667
Epoch 4/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2468 - accuracy: 0.9012 - AUC:
0.9627 - val_loss: 0.2223 - val_accuracy: 0.9111 - val_AUC: 0.9698
Epoch 5/50
1250/1250 [==============================] - 9s 7ms/step - loss: 0.2417 - accuracy: 0.9027 - AUC:
0.9643 - val_loss: 0.2230 - val_accuracy: 0.9129 - val_AUC: 0.9697
Epoch 6/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2385 - accuracy: 0.9036 - AUC:
0.9652 - val_loss: 0.2148 - val_accuracy: 0.9140 - val_AUC: 0.9720
Epoch 7/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2351 - accuracy: 0.9047 - AUC:
0.9663 - val_loss: 0.2326 - val_accuracy: 0.9082 - val_AUC: 0.9701
Epoch 8/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2314 - accuracy: 0.9074 - AUC:
0.9672 - val_loss: 0.2188 - val_accuracy: 0.9094 - val_AUC: 0.9707
Epoch 9/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2303 - accuracy: 0.9078 - AUC:
0.9675 - val_loss: 0.2286 - val_accuracy: 0.9050 - val_AUC: 0.9688
Epoch 10/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2304 - accuracy: 0.9076 - AUC:
0.9675 - val_loss: 0.2285 - val_accuracy: 0.9122 - val_AUC: 0.9691
Epoch 11/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2286 - accuracy: 0.9079 - AUC:
0.9681 - val_loss: 0.2158 - val_accuracy: 0.9161 - val_AUC: 0.9715
Epoch 12/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2265 - accuracy: 0.9082 - AUC:
0.9687 - val_loss: 0.2113 - val_accuracy: 0.9166 - val_AUC: 0.9732
Epoch 13/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2266 - accuracy: 0.9084 - AUC:
0.9685 - val_loss: 0.2122 - val_accuracy: 0.9161 - val_AUC: 0.9726
Epoch 14/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2244 - accuracy: 0.9100 - AUC:
0.9692 - val_loss: 0.2134 - val_accuracy: 0.9114 - val_AUC: 0.9724
Epoch 15/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2238 - accuracy: 0.9100 - AUC:
0.9693 - val_loss: 0.2081 - val_accuracy: 0.9179 - val_AUC: 0.9736
Epoch 16/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.2217 - accuracy: 0.9104 - AUC:
0.9699 - val_loss: 0.2078 - val_accuracy: 0.9165 - val_AUC: 0.9736
Epoch 17/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2236 - accuracy: 0.9093 - AUC:
0.9694 - val_loss: 0.2045 - val_accuracy: 0.9154 - val_AUC: 0.9744
Epoch 18/50
1241/1250 [============================>.] - ETA: 0s - loss: 0.2193 - accuracy: 0.9112 - AUC: 0.97
06
 We have Reached the accepted criteria of AUC -::- 0.9705479741096497 , so stopping training!!
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2194 - accuracy: 0.9112 - AUC:
0.9705 - val_loss: 0.2100 - val_accuracy: 0.9150 - val_AUC: 0.9741
time: 2min 18s
```

In [ ]:

```
%tensorboard --logdir logs/fit
```

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

In [ ]:

```
y_test_csv=trained_model.predict(test_pooled_output,verbose=1)
```

```
11/11 [==============================] - 0s 2ms/step
time: 115 ms
```

In [ ]:

```
y_classes = np.argmax(y_test_csv,axis=1)
```

```
time: 1.03 ms
```

In [ ]:

```
counts=np.unique(y_classes,return_counts=True)  ## Returns sorted unique values in the from of Tuple.
```

```
time: 2.01 ms
```

In [ ]:

```
print("Zero (",counts[0][0],") Count:",counts[1][0])
print("One (",counts[0][1],") Count:",counts[1][1])
```

```
Zero ( 0 ) Count: 26
One ( 1 ) Count: 326
time: 4.12 ms
```

# Summary

In [ ]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Training AUC","Test / Validation AUC"]
x.add_row([0.9705,0.9741 ])
print(x)
```

```
+--------------+-----------------------+
| Training AUC | Test / Validation AUC |
+--------------+-----------------------+
|    0.9705    |         0.9741        |
+--------------+-----------------------+
time: 13.8 ms
```

#### Observations and Steps Followed

#### Preprocessing

1. User tensorflow 2.0 to train the model.
2. Downloaded and extracted the Reviews.csv from “Amazon Fine Food Reviews” Kaggle website.
3. We see there are 9 columns in total but as per instructions we took only two columns['Text','Score'] for assignment.
4. Dropped the null rows and scored == 3
5. Manipulated the score columns into 1 and 0 from range(0,5)
6. updated the dataset by removing all the rows where the length of words in text is greater than 50.
7. Removed all HTML tags from text column.
8. split the data into train and test with 80% train.
9. As the count plots work on Data frame, converted the targets into data frame and plotted the counts in bar graph.

Source :: https://medium.com/@aieeshashafique/feature-extraction-from-bert-25887ed2152a

**Creating a BERT model from the Tensorflow HUB.**

1. We can choose any max length, BERT has a constraint on the maximum length of a sequence after tokenizing. For any BERT model, the maximum sequence length after tokenization is 512. But we can set any sequence length equal to or below this value.
2. Created three input layers with the names [input_word_ids , input_mask, segment_ids]
3. - input token ids (tokenizer converts tokens using vocab file)
   - input masks (1 for useful tokens, 0 for padding)
   - segment ids (for 2 text training: 0 for the first one, 1 for the second one)

1. Created a model with concatenation of three input layers and pooled output layer (pooled output of shape [batch_size, 768] with representations for the entire input sequences)

**3. Tokenization**

1. Install “sentencepiece” library to import tokenization.py file.
2. Import tokenizer using the original vocab file, do lower case all the word pieces and then tokenize the sentences.
3. Defined function “tokenization” to generate Id’s, segments and masks based on the original BERT
4. Creating tokens, mask, and segment by using tokenization function for X_train and X-test.

**4. getting the pretrained embedding Vector for a given review from the BERT.**

1. Generating the X_train_pooled_output and X_test_pooled_output Embeddings using the pre-trained model using Id’s, segments and masks.

**5. Using the embedding data apply NN and classify the reviews.**

1. As the “test_pooled_output’ shape is (352, 768) , here we are defining the model with input layer shape of (768,) followed with dense layers and dropout’s.
2. Creating a neural network function with 5 parameters NN(model,X_train_pooled_output,y_train,X_test_pooled_output,y_test)
3. Defining the callbacks based on few conditions.

**6. Creating a Data pipeline for BERT Model.**

1. Downloading test data
2. Creating the test Id’s , mask and segments
3. Converting into categories.
4. Training the model with train pooled data and Y_categories.
5. Predicting the values for y_test
6. defining the y_classes for the y_test with numpy library argmax function.
7. Counting the predicted values (counts=np.unique(y_classes,return_counts=True) ## Returns sorted unique values in the from of Tuple. )