In this notebook, You will do amazon review classification with BERT.[Download data from [this (https://www.kaggle.com/snap/amazon-fine-food-reviews/data)](https://www.kaggle.com/snap/amazon-fine-food-reviews/data) 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 [ ]:

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Mac
```

```
--2021-06-24 10:38:47--  https://storage.googleapis.com/kaggle-data-se
ts/18/2157/compressed/Reviews.csv.zip?X-Goog-Algorithm=GOOG4-RSA-SHA25
6&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccoun
t.com%2F20210531%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210531
T174719Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signat
ure=8975df44379c472dd1966622df11ff59455f4c16e7941312ff1a7624034a98a5a1
205addf3b8f5079a5ac76bfe6923b0e10b2bff7b13dead93095d8772c32f28ce332e9c
e29de25b5087b44c3d22b6db859579bb6d20ae79ea4bf4332037b6f1f16762a695cf2a
9e1b2ebece13d0769d19e50574fb412da4a7ffa1e15b8db1cae2486ef26f6ae3cdd25e
a956e0776f888f9ce3f474463d099f42e8b2dae06814f782f075db65610fc0859fc4db
24c1ce3cebbb822fa7e709874cba852f717fe0759b85b08fee88dec7c5b1ad35e034bd
3e59482e2198c11b909520ec8d52fc9a6fb7b7a22330c110ed9e3e1e152861ccf0596e
3dc4fa5bb644a136956742c178 (https://storage.googleapis.com/kaggle-data
-sets/18/2157/compressed/Reviews.csv.zip?X-Goog-Algorithm=GOOG4-RSA-SH
A256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceacco
unt.com%2F20210531%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=202105
31T174719Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Sign
ature=8975df44379c472dd1966622df11ff59455f4c16e7941312ff1a7624034a98a5
a1205addf3b8f5079a5ac76bfe6923b0e10b2bff7b13dead93095d8772c32f28ce332e
9ce29de25b5087b44c3d22b6db859579bb6d20ae79ea4bf4332037b6f1f16762a695cf
2a9e1b2ebece13d0769d19e50574fb412da4a7ffa1e15b8db1cae2486ef26f6ae3cdd2
5ea956e0776f888f9ce3f474463d099f42e8b2dae06814f782f075db65610fc0859fc4
db24c1ce3cebbb822fa7e709874cba852f717fe0759b85b08fee88dec7c5b1ad35e034
bd3e59482e2198c11b909520ec8d52fc9a6fb7b7a22330c110ed9e3e1e152861ccf059
6e3dc4fa5bb644a136956742c178)
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.20
3.128, 74.125.204.128, 64.233.188.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.2
03.128|:443... connected.
HTTP request sent, awaiting response... 400 Bad Request
2021-06-24 10:38:47 ERROR 400: Bad Request.
```

In [ ]:

```
!unzip /content/Reviews.csv.zip
```

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

In [1]:

```
#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
```

In [ ]:

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

Out[4]:

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

Grader function 1

In [ ]:

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

Out[5]:

```
True
```

Part-1: Preprocessing

In [ ]:

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

In [ ]:

```
#get only 2 columns - Text, Score
reviews = reviews[['Text','Score']]

#drop the NAN values
reviews.dropna()
reviews
```

Out[7]:

	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

In [ ]:

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

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

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

Out[8]:

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

525814 rows × 2 columns

Grader function 2

In [ ]:

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

Out[9]:

```
True
```

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)
```

In [ ]:

```
reviews.head()
## selected sample of 1L rows with len of less than 50
```

Out[11]:

	Text	Score	len
64117	The tea was of great quality and it tasted lik...	1	30
418112	My cat loves this. The pellets are nice and s...	1	31
357829	Great product. Does not completely get rid of ...	1	41
175872	This gum is my favorite! I would advise every...	1	27
178716	I also found out about this product because of...	1	22

In [ ]:

```
reviews.loc[10788]['Text']
```

Out[12]:

```
'royal canine is a great product Jake loves it he is now 3 months<br /
><br /><a href="http://www.amazon.com/gp/product/B001VIY9KW">Royal Can
in Dry Dog Food, Medium Puppy 32 Formula, 30-Pound Bag</a> old and 35
pounds boxer puppy'
```

In [ ]:

```
#remove HTML from the Text column and save in the Text column only
# [re.sub(r'[\n\r]*','', str(x)) for x in df['team']]
import re
reviews['Text'] = [re.sub(r'<.*?>',' ', str(x)) for x in reviews['Text']]
reviews.loc[10788]['Text']
```

Out[13]:

```
'royal canine is a great product Jake loves it he is now 3 months   Ro
yal Canin Dry Dog Food, Medium Puppy 32 Formula, 30-Pound Bag  old and
35  pounds boxer puppy'
```

In [ ]:

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

Out[14]:

	Text	Score	len
64117	The tea was of great quality and it tasted lik...	1	30
418112	My cat loves this. The pellets are nice and s...	1	31
357829	Great product. Does not completely get rid of ...	1	41
175872	This gum is my favorite! I would advise every...	1	27
178716	I also found out about this product because of...	1	22

In [ ]:

```
Y = reviews['Score']
X = reviews.drop(['Score'], axis=1)
```

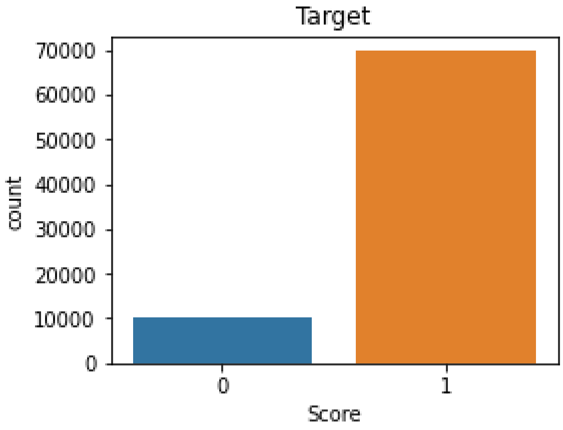
In [ ]:

```
#split the data into train and test data(20%) with Stratify sampling, random state 3
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, stratify=
```

In [ ]:

```
#plot bar graphs of y_train and y_test
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(4,3))
sns.countplot(Y_train)
plt.title('Target')
plt.show()
```

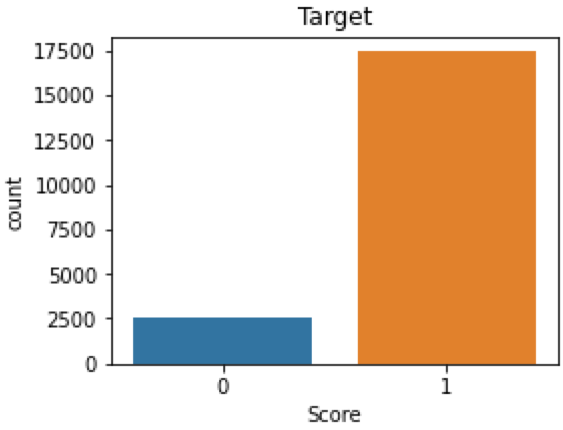
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futu
reWarning: Pass the following variable as a keyword arg: x. From versi
on 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error
or misinterpretation.
  FutureWarning
```



In [ ]:

```
plt.figure(figsize=(4,3))
sns.countplot(Y_test)
plt.title('Target')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futu
reWarning: Pass the following variable as a keyword arg: x. From versi
on 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error
or misinterpretation.
  FutureWarning
```



In [ ]:

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

Part-2: Creating BERT Model

If you want to know more about BERT, You can watch live sessions on Transformers and BERt.
we will strongly recommend you to read Transformers (https://jalammar.github.io/illustrated-transformer/), BERT Paper (https://arxiv.org/abs/1810.04805) and, This blog (https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/).

For this assignment, we are using BERT uncased Base model (https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1).
It uses L=12 hidden layers (i.e., Transformer blocks), a hidden size of H=768, and A=12 attention heads.

In [2]:

```
## 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 car
max_seq_length = 256

#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

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

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

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

In [3]:

```
bert_model.summary()
```

```
Model: "model"
______________________________________________________________________
____________________________
Layer (type)                    Output Shape         Param #     Conne
cted to
======================================================================
============================
input_word_ids (InputLayer)     [(None, 256)]        0
______________________________________________________________________
____________________________
input_mask (InputLayer)         [(None, 256)]        0
______________________________________________________________________
____________________________
segment_ids (InputLayer)        [(None, 256)]        0
______________________________________________________________________
____________________________
keras_layer (KerasLayer)        [(None, 768), (None, 109482241   input
_word_ids[0][0]
                                                                 input
_mask[0][0]
                                                                 segme
nt_ids[0][0]
======================================================================
============================
Total params: 109,482,241
Trainable params: 0
Non-trainable params: 109,482,241
______________________________________________________________________
____________________________
```

In [4]:

```
bert_model.output
```

Out[4]:

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

Part-3: Tokenization

In [5]:

```
#getting Vocab file
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
```

In [6]:

```
!pip install sentencepiece
```

```
Collecting sentencepiece
  Downloading https://files.pythonhosted.org/packages/ac/aa/1437691b0c
7c83086ebb79ce2da16e00bef024f24fec2a5161c35476f499/sentencepiece-0.1.9
6-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (https://f
iles.pythonhosted.org/packages/ac/aa/1437691b0c7c83086ebb79ce2da16e00b
ef024f24fec2a5161c35476f499/sentencepiece-0.1.96-cp37-cp37m-manylinux_
2_17_x86_64.manylinux2014_x86_64.whl) (1.2MB)
     |████████████████████████████████| 1.2MB 7.4MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.96
```

In [7]:

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

In [8]:

```
# 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_cas
# please check the "tokenization.py" file the complete implementation

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

Grader function 3

In [ ]:

```
#it has to give no error
def grader_tokenize(tokenizer):
    out = False
    try:
        out=('[CLS]' in tokenizer.vocab) and ('[SEP]' in tokenizer.vocab)
    except:
        out = False
    assert(out==True)
    return out
grader_tokenize(tokenizer)
```

Out[28]:

```
True
```

In [9]:

```
from tqdm import tqdm

# Create train and test tokens (X_train_tokens, X_test_tokens) from (X_train, X_test

# 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

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

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

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

# type of all the above arrays should be numpy arrays

def apply_tokenization(corpus):
    corpus_tokens = []
    corpus_mask = []
    corpus_segment = []
    for txt in tqdm(corpus):
        tokens = tokenizer.tokenize(txt)
        if len(tokens) >= max_seq_length-2 :
            tokens = tokens[0:(max_seq_length-2)]
        tokens = ['[CLS]',*tokens,'[SEP]']
        token_mask = np.array([1]*len(tokens)+[0]*(max_seq_length - len(tokens)))

        if len(tokens) < max_seq_length:
            tokens += (max_seq_length - len(tokens)) * ['[PAD]']

        tokens = np.array(tokenizer.convert_tokens_to_ids(tokens))

        segment = np.array([0]*max_seq_length)

        corpus_tokens.append(tokens)
        corpus_mask.append(token_mask)
        corpus_segment.append(segment)

    corpus_tokens = np.asarray(corpus_tokens)
    corpus_mask = np.asarray(corpus_mask)
    corpus_segment = np.asarray(corpus_segment)
    print(corpus_tokens.shape, corpus_mask.shape, corpus_segment.shape)
    return corpus_tokens, corpus_mask, corpus_segment
```

In [ ]:

```
# after execution of this cell, you have to get
X_train_tokens, X_train_mask, X_train_segment = apply_tokenization(X_train['Text'])
X_test_tokens, X_test_mask, X_test_segment = apply_tokenization(X_test['Text'])
```

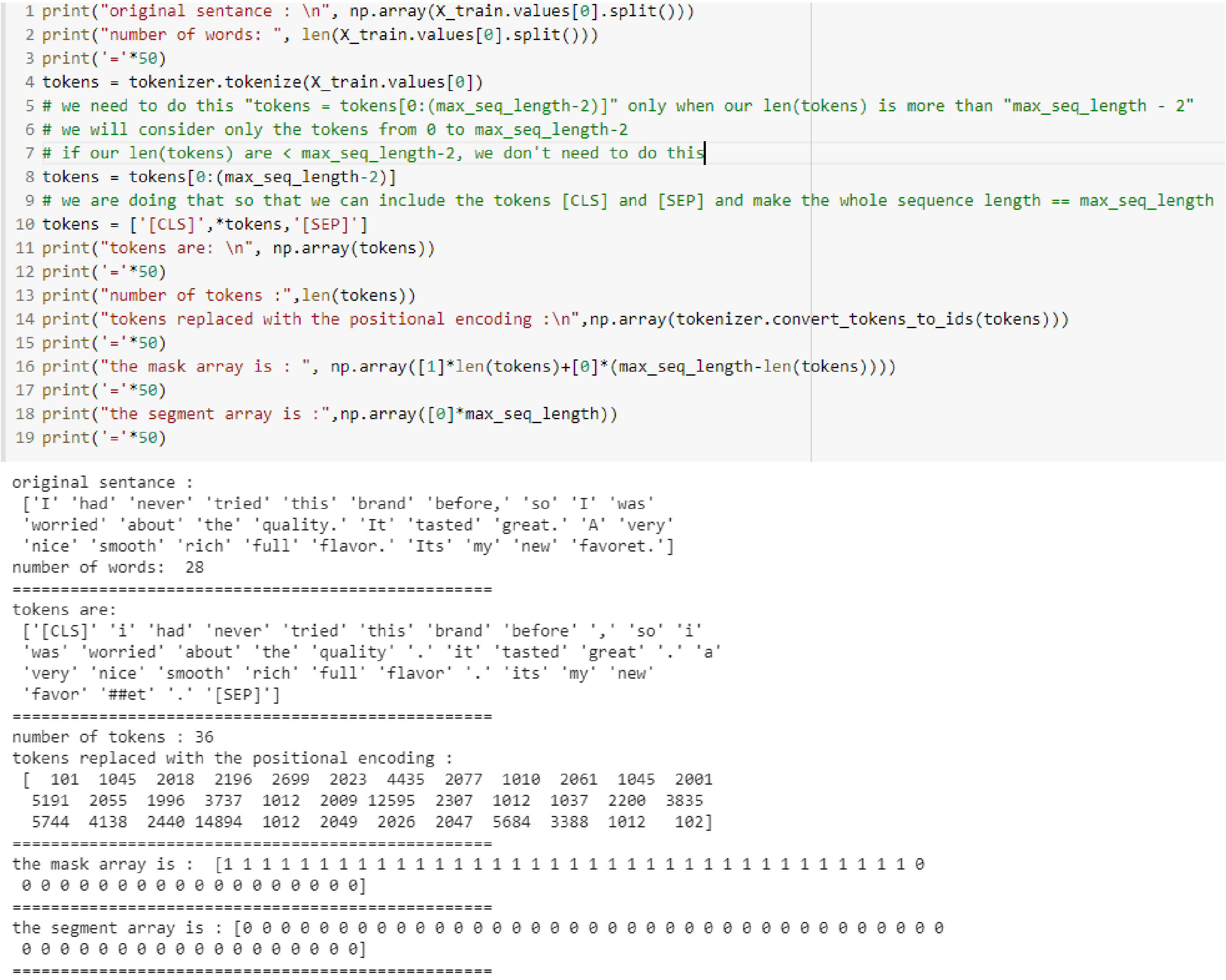
```
100%|██████████| 80000/80000 [01:03<00:00, 1269.63it/s]
  1%|          | 122/20000 [00:00<00:16, 1214.47it/s]

(80000, 256) (80000, 256) (80000, 256)

100%|██████████| 20000/20000 [00:15<00:00, 1275.46it/s]

(20000, 256) (20000, 256) (20000, 256)
```

In [ ]:

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

```
(20000, 256) (20000, 256) (20000, 256)
```

Example



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

```
original sentance :
 ['I' 'had' 'never' 'tried' 'this' 'brand' 'before,' 'so' 'I' 'was'
 'worried' 'about' 'the' 'quality.' 'It' 'tasted' 'great.' 'A' 'very'
 'nice' 'smooth' 'rich' 'full' 'flavor.' 'Its' 'my' 'new' 'favoret.']
number of words:  28
==================================================
tokens are:
 ['[CLS]' 'i' 'had' 'never' 'tried' 'this' 'brand' 'before' ',' 'so' 'i'
 'was' 'worried' 'about' 'the' 'quality' '.' 'it' 'tasted' 'great' '.' 'a'
 'very' 'nice' 'smooth' 'rich' 'full' 'flavor' '.' 'its' 'my' 'new'
 'favor' '##et' '.' '[SEP]']
==================================================
number of tokens : 36
tokens replaced with the positional encoding :
 [  101  1045  2018  2196  2699  2023  4435  2077  1010  2061  1045  2001
  5191  2055  1996  3737  1012  2009 12595  2307  1012  1037  2200  3835
  5744  4138  2440 14894  1012  2049  2026  2047  5684  3388  1012   102]
==================================================
the mask array is :  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
==================================================
the segment array is : [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
==================================================
```

In [ ]:

```
import pickle
```

In [ ]:

```
##save all your results to disk so that, no need to run all again.
pickle.dump((X_train, X_train_tokens, X_train_mask, X_train_segment, Y_train),open('
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, Y_test),open('/cont
```

In [ ]:

```
#you can load from disk
# X_train, X_train_tokens, X_train_mask, X_train_segment, Y_train = pickle.load(oper
# X_test, X_test_tokens, X_test_mask, X_test_segment, Y_test = pickle.load(open("tes
```

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.sh
        (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.sh

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

        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[35]:

```
True
```

Grader function 5

In [ ]:

```
def grader_alltokens_test():
    out = False
    if type(X_test_tokens) == np.ndarray:

        temp_shapes = (X_test_tokens.shape[1]==max_seq_length) and (X_test_mask.shap
        (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.shap

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

        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[36]:

```
True
```

Part-4: Getting Embeddings from BERT Model

We already created the BERT model in the part-2 and input data in the part-3.
We will utlize those two and will get the embeddings for each sentence in the
Train and test data.

In [ ]:

```
bert_model.input
```

Out[10]:

```
[<KerasTensor: shape=(None, 256) dtype=int32 (created by layer 'input_
word_ids')>,
 <KerasTensor: shape=(None, 256) dtype=int32 (created by layer 'input_
mask')>,
 <KerasTensor: shape=(None, 256) dtype=int32 (created by layer 'segmen
t_ids')>]
```

In [ ]:

```
bert_model.output
```

Out[11]:

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

In [ ]:

```
# get the train output, BERT model will give one output so save in
# X_train_pooled_output
X_train_pooled_output=bert_model.predict([X_train_tokens,X_train_mask,X_train_segmen
```

In [ ]:

```
# get the test output, BERT model will give one output so save in
# X_test_pooled_output
X_test_pooled_output=bert_model.predict([X_test_tokens,X_test_mask,X_test_segment])
```

In [ ]:

```
##save all your results to disk so that, no need to run all again.
pickle.dump((X_train_pooled_output, X_test_pooled_output),open('/content/drive/MyDri
```

In [ ]:

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

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[44]:

```
True
```

Part-5: Training a NN with 768 features

Create a NN and train the NN.

1. **You have to use AUC as metric.**

```
2. You can use any architecture you want.
3. You have to use tensorboard to log all your metrics and Losses. You have
 to send those logs.
4. Print the loss and metric at every epoch.
5. You have to submit without overfitting and underfitting.
```

In [10]:

```
import pickle

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

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

In [11]:

```
X_train_pooled_output, X_test_pooled_output= pickle.load(open('/content/drive/MyDriv
```

In [12]:

```
X_train_pooled_output.shape
```

Out[12]:

```
(80000, 768)
```

In [13]:

```
X_train, X_train_tokens, X_train_mask, X_train_segment, Y_train = pickle.load(open('
```

In [14]:

```
Y_train.shape
```

Out[14]:

```
(80000,)
```

In [15]:

```
##imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout, Flatten
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.initializers import GlorotUniform
from tensorflow.keras.regularizers import L2
import tensorflow as tf
from sklearn.metrics import roc_auc_score
```

In [16]:

```
def auc_func(y_true, y_pred):
    _auc = roc_auc_score(y_true, y_pred, average='micro')
    return _auc

def auc_score(y_true, y_pred):
    return tf.py_function(auc_func, (y_true, y_pred), tf.double)
```

In [17]:

```
##create an NN and

model = Sequential()
model.add(Input(shape = (768)))
model.add(Flatten())
for i in [512, 256, 128, 64, 32, 16, 8, 4]:
    model.add(Dense(i, activation = 'relu', kernel_initializer = tf.keras.initialize
model.add(Dense(1, activation = 'sigmoid', kernel_initializer = tf.keras.initializer
opt = tf.keras.optimizers.Adam(learning_rate=0.001, decay = 1e-4)
model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = [auc_score])
model.summary()
```

```
WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `ke
ras.Input` to Sequential model. `keras.Input` is intended to be used b
y Functional model.

WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `ke
ras.Input` to Sequential model. `keras.Input` is intended to be used b
y Functional model.

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 768)               0
_________________________________________________________________
dense (Dense)                (None, 512)               393728
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_5 (Dense)              (None, 16)                528
_________________________________________________________________
dense_6 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_7 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 5
=================================================================
Total params: 568,993
Trainable params: 568,993
Non-trainable params: 0
_________________________________________________________________
```

In [18]:

```
import numpy as np

Y_train = np.asarray(Y_train).astype('float32').reshape((-1,1))
```

In [19]:

```
Y_train.shape
```

Out[19]:

```
(80000, 1)
```

In [20]:

```
import datetime
log_dir = "logs/fit/model1_" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_fre

callback=[tensorboard_callback]
model.fit(X_train_pooled_output, Y_train, batch_size=128, epochs=20, validation_spli
```

```
Epoch 1/20
469/469 [==============================] - 7s 8ms/step - loss: 0.2724
- auc_score: 0.8821 - val_loss: 0.2285 - val_auc_score: 0.9425
Epoch 2/20
469/469 [==============================] - 3s 6ms/step - loss: 0.2034
- auc_score: 0.9477 - val_loss: 0.2246 - val_auc_score: 0.9509
Epoch 3/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1959
- auc_score: 0.9520 - val_loss: 0.1886 - val_auc_score: 0.9529
Epoch 4/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1847
- auc_score: 0.9547 - val_loss: 0.1749 - val_auc_score: 0.9552
Epoch 5/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1817
- auc_score: 0.9561 - val_loss: 0.1732 - val_auc_score: 0.9561
Epoch 6/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1757
- auc_score: 0.9573 - val_loss: 0.1693 - val_auc_score: 0.9574
Epoch 7/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1751
- auc_score: 0.9589 - val_loss: 0.1972 - val_auc_score: 0.9573
Epoch 8/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1739
- auc_score: 0.9594 - val_loss: 0.1689 - val_auc_score: 0.9585
Epoch 9/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1738
- auc_score: 0.9603 - val_loss: 0.1650 - val_auc_score: 0.9593
Epoch 10/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1706
- auc_score: 0.9601 - val_loss: 0.1619 - val_auc_score: 0.9594
Epoch 11/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1696
- auc_score: 0.9614 - val_loss: 0.1926 - val_auc_score: 0.9597
Epoch 12/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1662
- auc_score: 0.9620 - val_loss: 0.1592 - val_auc_score: 0.9610
Epoch 13/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1643
- auc_score: 0.9624 - val_loss: 0.1591 - val_auc_score: 0.9607
Epoch 14/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1669
- auc_score: 0.9629 - val_loss: 0.1618 - val_auc_score: 0.9611
Epoch 15/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1658
- auc_score: 0.9627 - val_loss: 0.1585 - val_auc_score: 0.9614
Epoch 16/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1653
- auc_score: 0.9631 - val_loss: 0.1706 - val_auc_score: 0.9614
Epoch 17/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1600
- auc_score: 0.9639 - val_loss: 0.1693 - val_auc_score: 0.9617
Epoch 18/20
```

```
469/469 [==============================] - 3s 6ms/step - loss: 0.1606
- auc_score: 0.9640 - val_loss: 0.1700 - val_auc_score: 0.9621
Epoch 19/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1603
- auc_score: 0.9647 - val_loss: 0.1552 - val_auc_score: 0.9625
Epoch 20/20
469/469 [==============================] - 3s 6ms/step - loss: 0.1618
- auc_score: 0.9644 - val_loss: 0.2263 - val_auc_score: 0.9600
```

Out[20]:

```
<tensorflow.python.keras.callbacks.History at 0x7fd85b3a1250>
```

In [21]:

```
%load_ext tensorboard
%tensorboard --logdir /content/logs/fit/model1_20210630-073517
```

```
<IPython.core.display.Javascript object>
```

In [ ]:

```
model.save('/content/drive/MyDrive/Data Science Assignments/BERT_Assignment/trained_
```

Part-6: Creating a Data pipeline for BERT Model

1. Download data from here (https://drive.google.com/file/d/1QwjqTsqTX2vdy7fTmeXjxP3dq8IAVLpo/view?usp=sharing)
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

In [22]:

```
test = pd.read_csv('/content/drive/MyDrive/Data Science Assignments/BERT_Assignment/
```

In [23]:

```
test.head()
```

Out[23]:

	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 ...

In [24]:

```
test['Text'][2]
```

Out[24]:

```
"The product was fine, but the cost of shipping was more than the cost
<br />of the tea.  Won't make that mistake again."
```

In [25]:

```
test['Text'].str.findall(r'<.*?>').head()
```

Out[25]:

```
0          []
1          []
2    [<br />]
3          []
4          []
Name: Text, dtype: object
```

In [26]:

```
import re
test['Text'] = [re.sub(r'<.*?>',' ', str(x)) for x in test['Text']]
test.loc[2]['Text']
```

Out[26]:

```
"The product was fine, but the cost of shipping was more than the cost
of the tea.  Won't make that mistake again."
```

In [27]:

```
test_tokens, test_mask, test_segment = apply_tokenization(test['Text'])
```

```
100%|██████████| 352/352 [00:00<00:00, 1512.61it/s]

(352, 256) (352, 256) (352, 256)
```

In [28]:

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

In [29]:

```
X_test.shape
```

Out[29]:

```
(352, 768)
```

In [30]:

```
predictions = model.predict(X_test)
```

In [31]:

```
test['predictions'] = predictions
```

In [32]:

```
test.head(10)
```

Out[32]:

	Text	predictions
0	Just opened Greenies Joint Care (individually ...	0.130973
1	This product rocks :) My mom was very happy w/...	0.997140
2	The product was fine, but the cost of shipping...	0.053868
3	I love this soup. It's great as part of a meal...	0.449001
4	Getting ready to order again. These are great ...	0.954965
5	These were delicious, but not wrapped as well ...	0.162330
6	I will never again even CONSIDER a dog food wi...	0.014322
7	If you need something to take with you to keep...	0.996134
8	My husband puts this on everything. It is very...	0.996713
9	This is a movie the whole family can watch tog...	0.996506

In [33]:

```
test['score'] = np.where(test['predictions'] > 0.5, 1,0)
```

In [36]:

```
test.head(10)
```

Out[36]:

	Text	predictions	score
0	Just opened Greenies Joint Care (individually ...	0.130973	0
1	This product rocks :) My mom was very happy w/...	0.997140	1
2	The product was fine, but the cost of shipping...	0.053868	0
3	I love this soup. It's great as part of a meal...	0.449001	0
4	Getting ready to order again. These are great ...	0.954965	1
5	These were delicious, but not wrapped as well ...	0.162330	0
6	I will never again even CONSIDER a dog food wi...	0.014322	0
7	If you need something to take with you to keep...	0.996134	1
8	My husband puts this on everything. It is very...	0.996713	1
9	This is a movie the whole family can watch tog...	0.996506	1

In [38]:

```
test.groupby('score').size()
```

Out[38]:

```
score
0     82
1    270
dtype: int64
```

Observation and Explanation

1. In preprocessing of data, the score of review text having <=2 is changed to 0 and having >3 is changed to 1; to change into binary classification problem
2. Removed all the html tags using regex
3. Created a Bert model using a pre trained uncased Bert model and adding an output layer to it
4. A tokeniser is created using tokenization.py file and the parameters from Bert model
5. I have changed the max sequence length to 256 to cover all texts tokens
6. [CLS] and [SEP] is added at the each end of tokens
7. Added [PAD] token if the length is less than max sequence length
8. Created a mask array that contains 1 for real token and 0 for padded token and a segment array that has 0 and length of max sequence length
9. Token array, mask array and segment array for train dataset is used as an input to Bert model created earlier
10. Predict the pooled output from Bert model
11. Created a NN with 768 features
12. Trained this NN on the pooled output and got auc_score of 0.96 on validation set
13. Preprocessed test.csv with all the same steps as train dataset
14. Predicted the output using the trained NN and received score of

0 : 82

1 : 270