In this notebook, You will do amazon review classification with BERT. It contains 5 parts as below. Detailed instrctions are given in the eac h 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 t he BERT.
5. Using the embedding data apply NN and classify the reviews.

instructions:

1. Don't change any Grader Functions. Don't manipulate any Grader fu nctions. If you manipulate any, it will be considered as plagiarised.

2. Please read the instructions on the code cells and markdown cell s. We will explain what to write.

3. please return outputs in the same format what we asked. Eg. Don't return List of 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 f ollow them.

Every Grader function has to return True.

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
```

In [ ]:
```
tf.test.gpu_device_name()
```

Grader function 1

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

Part-1: Preprocessing

In [ ]:
```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv(r"D:\ML\Internal DL\NLP\amazon-fine-food-reviews\Revie
#check the info of the dataset
reviews.info()
```

In [ ]:
```
#get only 2 columns - Text, Score
#drop the NAN values
```

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

Grader function 2

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

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 [ ]:
```
#remove HTML from the Text column and save in the Text column only
```

In [ ]:
```
#print head 5
```

In [ ]:
```
#split the data into train and valudation data(20%) with Stratify sampling,
```

In [ ]:
```
#plot bar graphs of y_train and y_test
```

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 Tran sformers 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.i]()

o/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 [ ]:
```
## 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 300
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.int

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

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

#bert layer
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-
pooled_output, sequence_output = bert_layer([input_word_ids, input_mask, seg

#Bert model
#We are using only pooled output not sequence out.
#If you want to know about those, please read https://www.kaggle.com/questio
bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs

```

In [ ]:
```
bert_model.summary()
```

In [ ]:
```
bert_model.output
```

Part-3: Tokenization

In [ ]:
```
#getting Vocab file
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
```

In [ ]:
```
#import tokenization - We have given tokenization.py file
```

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

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)
```

In [ ]:

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

# 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)

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

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

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

# 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
```

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]
==================================================
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]
==================================================
```

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
pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),ope
```

In [ ]:
```
#you can load from disk
#X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.lo
#X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(op
```

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
        (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_t

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

        out = temp_shapes and segment_temp and mask_temp and no_cls and no_s

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

grader_alltokens_train()
```

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_m
        (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_tok

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

        out = temp_shapes and segment_temp and mask_temp and no_cls and no_s

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

Part-4: Getting Embeddings from BERT M

odel

We already created the BERT model in the part-2 and input data in the pa
rt-3. We will utlize those two and will get the embeddings for each sent
ence in the Train and Validation data.

In [ ]:
```
bert_model.input
```

In [ ]:
```
bert_model.output
```

In [ ]:
```
# get the train output, BERT model will give one output so save in
# X_train_pooled_output
```

In [ ]:
```
# get the test output, BERT model will give one output so save in
# X_test_pooled_output
```

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
```

In [ ]:
```
#X_train_pooled_output, X_test_pooled_output= pickle.load(open('final_output
```

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()
```

Part-5: Training a NN with 786 feature
s

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 h
ave 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
```

In [ ]:
```
##create an NN and

```

In [ ]:
```

```