In [ ]: In [ ]:

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.

1

*#all imports*

2

**import** numpy **as** np

3

**import** pandas **as** pd

4

**import** tensorflow **as** tf

5

**import** tensorflow\_hub **as** hub

6

**from** tensorflow.keras.models **import** Model

1

tf.test.gpu\_device\_name()

Grader function 1

In [ ]:

1

**def** grader\_tf\_version():

2

**assert**((tf.\_\_version\_\_)**>**'2') 3

**return True**

4

grader\_tf\_version()

In [ ]:

In [ ]: In [ ]:

Part-1: Preprocessing

1

*#Read the dataset - Amazon fine food reviews*

2

reviews **=** pd.read\_csv(r"D:\ML\Internal DL\NLP\amazon-fine-food-reviews\Revie 3

*#check the info of the dataset*

4

reviews.info()

1

*#get only 2 columns - Text, Score*

2

*#drop the NAN values*

1

*#if score> 3, set score = 1*

2

*#if score<=2, set score = 0*

3

*#if score == 3, remove the rows.*

Grader function 2

In [ ]: In [ ]:

In [ ]: In [ ]: In [ ]: In [ ]: In [ ]:

1

**def** grader\_reviews():

2

temp\_shape **=** (reviews.shape **==** (525814, 2)) **and** (reviews.Score.value\_cou 3

**assert**(temp\_shape **== True**)

4

**return True**

5

grader\_reviews()

1

**def** get\_wordlen(x):

2

**return** len(x.split())

3

reviews['len'] **=** reviews.Text.apply(get\_wordlen)

4

reviews **=** reviews[reviews.len**<**50]

5

reviews **=** reviews.sample(n**=**100000, random\_state**=**30)

1

*#remove HTML from the Text column and save in the Text column only*

1

*#print head 5*

1

*#split the data into train and valudation data(20%) with Stratify sampling,*

1

*#plot bar graphs of y\_train and y\_test*

1

*#saving to disk. if we need, we can load preprocessed data directly.* 2

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 (http s://arxiv.org/abs/1810.04805) and, This blog (https://jalammar.github.i

In [ ]:

In [ ]: In [ ]:

In [ ]: In [ ]:

o/a-visual-guide-to-using-bert-for-the-first-time/).

For this assignment, we are using BERT uncased Base model (https://tfhu b.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 atte ntion heads.

1

*## Loading the Pretrained Model from tensorflow HUB*

2

tf.keras.backend.clear\_session()

3

4

*# maximum length of a seq in the data we have, for now i am making it as 300* 5

max\_seq\_length **=** 55

6

7

*#BERT takes 3 inputs*

8

9

*#this is input words. Sequence of words represented as integers* 10

input\_word\_ids **=** tf.keras.layers.Input(shape**=**(max\_seq\_length,), dtype**=**tf.int 11

12

*#mask vector if you are padding anything*

13

input\_mask **=** tf.keras.layers.Input(shape**=**(max\_seq\_length,), dtype**=**tf.int32, 14

15

*#segment vectors. If you are giving only one sentence for the classification* 16

*#If you are giving two sentenced with [sep] token separated, first seq segme* 17

*#second seq segment vector are 1's*

18

segment\_ids **=** tf.keras.layers.Input(shape**=**(max\_seq\_length,), dtype**=**tf.int32, 19

20

*#bert layer*

21

bert\_layer **=** hub.KerasLayer("https://tfhub.dev/tensorflow/bert\_en\_uncased\_L 22

pooled\_output, sequence\_output **=** bert\_layer([input\_word\_ids, input\_mask, seg 23

24

*#Bert model*

25

*#We are using only pooled output not sequence out.*

26

*#If you want to know about those, please read https://www.kaggle.com/questio* 27

bert\_model **=** Model(inputs**=**[input\_word\_ids, input\_mask, segment\_ids], outputs 28

1

bert\_model.summary()

1

bert\_model.output

Part-3: Tokenization

1

*#getting Vocab file*

2

vocab\_file **=** bert\_layer.resolved\_object.vocab\_file.asset\_path.numpy() 3

do\_lower\_case **=** bert\_layer.resolved\_object.do\_lower\_case.numpy()

1

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

In [ ]:

1

*# Create tokenizer " Instantiate FullTokenizer"*

2

*# name must be "tokenizer"*

3

*# the FullTokenizer takes two parameters 1. vocab\_file and 2. do\_lower\_case* 4

*# we have created these in the above cell ex: FullTokenizer(vocab\_file, do\_l* 5

*# please check the "tokenization.py" file the complete implementation*

Grader function 3

In [ ]: In [ ]:

1

*#it has to give no error*

2

**def** grader\_tokenize(tokenizer):

3

out **= False**

4

**try**:

5

out**=**('[CLS]' **in** tokenizer.vocab) **and** ('[SEP]' **in** tokenizer.vocab) 6

**except**:

7

out **= False**

8

**assert**(out**==True**)

9

**return** out

10

grader\_tokenize(tokenizer)

1

*# Create train and test tokens (X\_train\_tokens, X\_test\_tokens) from (X\_train* 2

3

*# add '[CLS]' at start of the Tokens and '[SEP]' at the end of the tokens.* 4

5

*# maximum number of tokens is 55(We already given this to BERT layer above)* 6

7

*# if it is less than 55, add '[PAD]' token else truncate the tokens length.(* 8

9

*# Based on padding, create the mask for Train and Test ( 1 for real token, 0* 10

*# it will also same shape as input tokens (None, 55) save those in X\_train\_m* 11

12

*# Create a segment input for train and test. We are using only one sentence* 13

14

*# type of all the above arrays should be numpy arrays*

15

16

*# after execution of this cell, you have to get*

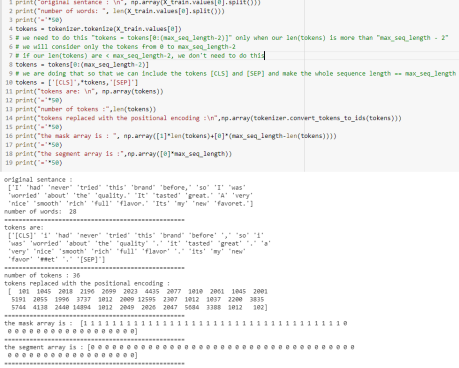
17

*# X\_train\_tokens, X\_train\_mask, X\_train\_segment*

18

*# X\_test\_tokens, X\_test\_mask, X\_test\_segment*

**Example**

****

In [ ]: In [ ]:

In [ ]:

1

**import** pickle

1

*##save all your results to disk so that, no need to run all again.* 2

pickle.dump((X\_train, X\_train\_tokens, X\_train\_mask, X\_train\_segment, y\_train 3

pickle.dump((X\_test, X\_test\_tokens, X\_test\_mask, X\_test\_segment, y\_test),ope

1

*#you can load from disk*

2

*#X\_train, X\_train\_tokens, X\_train\_mask, X\_train\_segment, y\_train = pickle.lo* 3

*#X\_test, X\_test\_tokens, X\_test\_mask, X\_test\_segment, y\_test = pickle.load(op*

Grader function 4

In [ ]:

1

**def** grader\_alltokens\_train():

2

out **= False**

3

4

**if** type(X\_train\_tokens) **==** np.ndarray:

5

6

temp\_shapes **=** (X\_train\_tokens.shape[1]**==**max\_seq\_length) **and** (X\_train\_ 7

(X\_train\_segment.shape[1]**==**max\_seq\_length)

8

9

segment\_temp **= not** np.any(X\_train\_segment)

10

11

mask\_temp **=** np.sum(X\_train\_mask**==**0) **==** np.sum(X\_train\_tokens**==**0) 12

13

no\_cls **=** np.sum(X\_train\_tokens**==**tokenizer.vocab['[CLS]'])**==**X\_train\_t 14

15

no\_sep **=** np.sum(X\_train\_tokens**==**tokenizer.vocab['[SEP]'])**==**X\_train\_t 16

17

out **=** temp\_shapes **and** segment\_temp **and** mask\_temp **and** no\_cls **and** no\_s 18

19

**else**:

20

print('Type of all above token arrays should be list not numpy array 21

out **= False**

22

**assert**(out**==True**)

23

**return** out

24

25

grader\_alltokens\_train()

Grader function 5

In [ ]:

1

**def** grader\_alltokens\_test():

2

out **= False**

3

**if** type(X\_test\_tokens) **==** np.ndarray:

4

5

temp\_shapes **=** (X\_test\_tokens.shape[1]**==**max\_seq\_length) **and** (X\_test\_m 6

(X\_test\_segment.shape[1]**==**max\_seq\_length)

7

8

segment\_temp **= not** np.any(X\_test\_segment)

9

10

mask\_temp **=** np.sum(X\_test\_mask**==**0) **==** np.sum(X\_test\_tokens**==**0) 11

12

no\_cls **=** np.sum(X\_test\_tokens**==**tokenizer.vocab['[CLS]'])**==**X\_test\_tok 13

14

no\_sep **=** np.sum(X\_test\_tokens**==**tokenizer.vocab['[SEP]'])**==**X\_test\_tok 15

16

out **=** temp\_shapes **and** segment\_temp **and** mask\_temp **and** no\_cls **and** no\_s 17

18

**else**:

19

print('Type of all above token arrays should be list not numpy array 20

out **= False**

21

**assert**(out**==True**)

22

**return** out

23

grader\_alltokens\_test()

Part-4: Getting Embeddings from BERT M

In [ ]: In [ ]: In [ ]:

In [ ]: In [ ]: In [ ]:

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.

1

bert\_model.input

1

bert\_model.output

1

*# get the train output, BERT model will give one output so save in* 2

*# X\_train\_pooled\_output*

1

*# get the test output, BERT model will give one output so save in* 2

*# X\_test\_pooled\_output*

1

*##save all your results to disk so that, no need to run all again.* 2

pickle.dump((X\_train\_pooled\_output, X\_test\_pooled\_output),open('final\_output

1

*#X\_train\_pooled\_output, X\_test\_pooled\_output= pickle.load(open('final\_output*

Grader function 6

In [ ]:

1

*#now we have X\_train\_pooled\_output, y\_train*

2

*#X\_test\_pooled\_ouput, y\_test*

3

4

*#please use this grader to evaluate*

5

**def** greader\_output():

6

**assert**(X\_train\_pooled\_output.shape[1]**==**768)

7

**assert**(len(y\_train)**==**len(X\_train\_pooled\_output))

8

**assert**(X\_test\_pooled\_output.shape[1]**==**768)

9

**assert**(len(y\_test)**==**len(X\_test\_pooled\_output))

10

**assert**(len(y\_train.shape)**==**1)

11

**assert**(len(X\_train\_pooled\_output.shape)**==**2)

12

**assert**(len(y\_test.shape)**==**1)

13

**assert**(len(X\_test\_pooled\_output.shape)**==**2)

14

**return True**

15

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.

In [ ]:

In [ ]: In [ ]:

4. Print the loss and metric at every epoch.

5. You have to submit without overfitting and underfitting.

1

*##imports*

2

**from** tensorflow.keras.layers **import** Input, Dense, Activation, Dropout 3

**from** tensorflow.keras.models **import** Model

1

*##create an NN and*

2

1