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) 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 wil
l 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 t
he concept behind the code that you are writing.

    5. We are giving instructions at each section if necessary, please follow th
em.
```

Every Grader function has to return True.

In [ ]:

```
!pip3 install tensorflow==2.2.0
```

In [1]:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow.keras.models import Model
import re
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
print(tf.__version__)
```

```
2.2.0
```

In [2]:

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

Out[2]:

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

Grader function 1

In [3]:

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

Out[3]:

```
True
```

Part-1: Preprocessing

In [3]:

```
#Read the dataset - Amazon fine food reviews
reviews = pd.read_csv(r"/content/drive/My Drive/Reviews.csv")
#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 [4]:

```
#get only 2 columns - Text, Score
#drop the NAN values
reviews=reviews[['Text','Score']]
reviews.dropna()
```

Out[4]:

	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 [5]:

```
#if score> 3, set score = 1
#if score<=2, set score = 0
#if score == 3, remove the rows.
x=reviews['Score']
reviews['Score']=reviews['Score'].apply(lambda x : 1 if x > 3 else (0 if x < 3 else x))
review=reviews[reviews['Score'] == 3].index
reviews=reviews.drop(review)
```

Grader function 2

In [6]:

```
def grader_reviews():
    temp_shape = (reviews.shape == (525814, 2)) and (reviews.Score.value_counts()[1]==4
43777)
    assert(temp_shape == True)
    return True
grader_reviews()
```

Out[6]:

```
True
```

In [6]:

```
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
reviews.Text.apply(lambda x : re.sub('<[^<]+?>', '', str(reviews['Text'])))
```

In [8]:

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

Out[8]:

	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 [9]:

```
X=reviews['Text']
y=reviews['Score']
#split the data into train and test data(20%) with Stratify sampling, random state 33,
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3
3)
```

In [10]:

```
#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/)](https://jalammar.github.io/illustrated-transformer/), [BERT Paper (https://arxiv.org/abs/1810.04805)](https://arxiv.org/abs/1810.04805) and, [This blog (https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/)](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)](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 [11]:

```
## 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)
```

In [12]:

```
bert_model.summary()
```

```
Model: "model"
________________________________________________________________________________
________________________
Layer (type)                    Output Shape         Param #     Connected
to
================================================================================
========================
input_word_ids (InputLayer)     [(None, 55)]         0
________________________________________________________________________________
________________________
input_mask (InputLayer)         [(None, 55)]         0
________________________________________________________________________________
________________________
segment_ids (InputLayer)        [(None, 55)]         0
________________________________________________________________________________
________________________
keras_layer (KerasLayer)        [(None, 768), (None, 109482241   input_wor
d_ids[0][0]
                                                                 input_mas
k[0][0]
                                                                 segment_i
ds[0][0]
================================================================================
========================
Total params: 109,482,241
Trainable params: 0
Non-trainable params: 109,482,241
________________________________________________________________________________
________________________
```

In [13]:

```
bert_model.output
```

Out[13]:

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

Part-3: Tokenization

In [14]:

```
#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 [15]:

```
!pip3 install tf_sentencepiece
```

```
Requirement already satisfied: tf_sentencepiece in /usr/local/lib/python3.
6/dist-packages (0.1.90)
```

In [16]:

```
#import tokenization - using tokenization.py file
import tokenization
def create_tokenizer(vocab_file, do_lower_case):
    return tokenization.FullTokenizer(vocab_file=vocab_file, do_lower_case=do_lower_cas
e)

tokenizer = create_tokenizer(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[ ]:

```
True
```

In [17]:

```
# Create train and test tokens (X_train_tokens, X_test_tokens) from (X_train, X_test) u
sing Tokenizer 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 '[PA
D]'),
# 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 zero
s. This shape 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

#Ref:https://medium.com/@vineet.mundhra/loading-bert-with-tensorflow-hub-7f5a1c722565

def convert_sentence_to_features(sentence, tokenizer, max_seq_len):
    tokens = ['[CLS]']
    tokens.extend(tokenizer.tokenize(sentence))
    if len(tokens) > max_seq_len-1:
        tokens = tokens[:max_seq_len-1]
    tokens.append('[SEP]')

    segment_ids = [0] * len(tokens)
    input_ids = tokenizer.convert_tokens_to_ids(tokens)
    input_mask = [1] * len(input_ids)

    #Zero Mask till seq_length
    zero_mask = [0] * (max_seq_len-len(tokens))
    input_ids.extend(zero_mask)
    input_mask.extend(zero_mask)
    segment_ids.extend(zero_mask)

    return input_ids, input_mask, segment_ids

def convert_sentences_to_features(sentences, tokenizer, max_seq_len=55):
    all_input_ids = []
    all_input_mask = []
    all_segment_ids = []

    for sentence in sentences:
        input_ids, input_mask, segment_ids = convert_sentence_to_features(sentence, tok
enizer, max_seq_len)
        all_input_ids.append(input_ids)
        all_input_mask.append(input_mask)
        all_segment_ids.append(segment_ids)

    return np.asarray(all_input_ids), np.asarray(all_input_mask), np.asarray(all_segmen
```

```
t_ids)

X_train_tokens, X_train_mask, X_train_segment = convert_sentences_to_features(X_train, 
tokenizer, 55)
X_test_tokens, X_test_mask, X_test_segment = convert_sentences_to_features(X_test, toke
nizer, 55)
```

Example

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

```
original sentance :
 ['I' 'had' 'never' 'tried' 'this' 'brand' 'before,' 'so' 'I' 'was'
 'worried' 'about' 'the' 'quality.' 'It' 'tasted' 'great.' 'A' 'very'
 'nice' 'smooth' 'rich' 'full' 'flavor.' 'Its' 'my' 'new' 'favoret.']
number of words:  28
==================================================
tokens are:
 ['[CLS]' 'i' 'had' 'never' 'tried' 'this' 'brand' 'before' ',' 'so' 'i'
 'was' 'worried' 'about' 'the' 'quality' '.' 'it' 'tasted' 'great' '.' 'a'
 'very' 'nice' 'smooth' 'rich' 'full' 'flavor' '.' 'its' 'my' 'new'
 'favor' '##et' '.' '[SEP]']
==================================================
number of tokens : 36
tokens replaced with the positional encoding :
 [  101  1045  2018  2196  2699  2023  4435  2077  1010  2061  1045  2001
  5191  2055  1996  3737  1012  2009 12595  2307  1012  1037  2200  3835
  5744  4138  2440 14894  1012  2049  2026  2047  5684  3388  1012   102]
==================================================
the mask array is :  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 [18]:

```
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('tr
ain_data.pkl','wb'))
#pickle.dump((X_test, X_test_tokens, X_test_mask, X_test_segment, y_test),open('test_da
ta.pkl','wb'))
```

In [19]:

```
#you can load from disk
X_train, X_train_tokens, X_train_mask, X_train_segment, y_train = pickle.load(open("/co
ntent/drive/My Drive/Files/train_data.pkl", 'rb'))
X_test, X_test_tokens, X_test_mask, X_test_segment, y_test = pickle.load(open("/conten
t/drive/My Drive/Files/test_data.pkl", 'rb'))
```

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
```

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
```

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 [20]:

```
bert_model.input
```

Out[20]:

```
[<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>]
```

In [21]:

```
bert_model.output
```

Out[21]:

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

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])
```

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('final_output.pkl','w
b'))
```

In [22]:

```
X_train_pooled_output, X_test_pooled_output= pickle.load(open('/content/drive/My Drive/
Files/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
```

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 [23]:

```
#imports
from tensorflow.keras.layers import Input, Dense, Activation, Dropout
from tensorflow.keras.models import Model
```

In [24]:

```
##create an NN and
#Ref:https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-model
s-python-keras/

# Use scikit-learn to grid search the batch size and epochs
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.metrics import roc_auc_score


def auc( y_true, y_pred ) :
    score = tf.py_function( lambda y_true, y_pred : roc_auc_score( y_true, y_pred, aver
age='macro', sample_weight=None).astype('float32'),
                        [y_true, y_pred],
                        'float32',
                        name='sklearnAUC' )
    return score


# Function to create model, required for KerasClassifier
def create_model():
        # create model
        model = Sequential()
        model.add(Dense(64, input_dim=768, activation='relu'))
    #model.add(Dropout(dropout_rate))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
, auc])
        return model

# create model
model = KerasClassifier(build_fn=create_model, verbose=0)

#Hyperparameter tuning bach size and epoch
# define the grid search parameters
batch_size = [1024]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train_pooled_output, y_train)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_execu
tor.py:691: UserWarning: A worker stopped while some jobs were given to th
e executor. This can be caused by a too short worker timeout or by a memor
y leak.
  "timeout or by a memory leak.", UserWarning

Best: 0.927913 using {'batch_size': 1024, 'epochs': 100}
```

In [25]:

```
#Hyperparameter number of hidden neurons
# Function to create model, required for KerasClassifier
def create_model(neurons=1):
        # create model
        model = Sequential()
        model.add(Dense(neurons, input_dim=768, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
,auc])
        return model

# create model
model = KerasClassifier(build_fn=create_model,epochs=100,batch_size=1024, verbose=0)

#Hyperparameter tuning number of hidden neurons
# define the grid search parameters
neurons =  [16,32,64,128]
param_grid = dict(neurons=neurons)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train_pooled_output, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_execu
tor.py:691: UserWarning: A worker stopped while some jobs were given to th
e executor. This can be caused by a too short worker timeout or by a memor
y leak.
  "timeout or by a memory leak.", UserWarning

Best: 0.930900 using {'neurons': 32}
```

In [26]:

```
# tensor-board in colab
# Refer: https://www.tensorflow.org/tensorboard/get_started
import os
import datetime

! rm -rf ./logs/
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
print(logdir)
```

```
logs/20201007-113034
```

In [27]:

```
%load_ext tensorboard
%tensorboard --logdir $logdir
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
```

In [28]:

```
#Train Neural network with best hyper parameters
# Function to create model, required for KerasClassifier
def create_model(neurons=1):
        # create model
        model = Sequential()
        model.add(Dense(128, input_dim=768, activation='relu'))
  #model.add(Dropout(dropout_rate))
        model.add(Dense(1, activation='sigmoid'))
        return model

# create model
model = create_model()
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy',auc])

model.fit(X_train_pooled_output, y_train, validation_split=0.33,epochs=100, batch_size=
1024,callbacks=[tensorboard_callback])
```

```
Epoch 1/100
53/53 [==============================] - 1s 14ms/step - loss: 0.4175 - acc
uracy: 0.8541 - auc: 0.6695 - val_loss: 0.3532 - val_accuracy: 0.8710 - va
l_auc: 0.7802
Epoch 2/100
53/53 [==============================] - 1s 10ms/step - loss: 0.3426 - acc
uracy: 0.8684 - auc: 0.8008 - val_loss: 0.3189 - val_accuracy: 0.8717 - va
l_auc: 0.8328
Epoch 3/100
53/53 [==============================] - 1s 11ms/step - loss: 0.3119 - acc
uracy: 0.8714 - auc: 0.8468 - val_loss: 0.2908 - val_accuracy: 0.8756 - va
l_auc: 0.8705
Epoch 4/100
53/53 [==============================] - 1s 10ms/step - loss: 0.2850 - acc
uracy: 0.8776 - auc: 0.8786 - val_loss: 0.2666 - val_accuracy: 0.8846 - va
l_auc: 0.8950
Epoch 5/100
53/53 [==============================] - 1s 10ms/step - loss: 0.2622 - acc
uracy: 0.8859 - auc: 0.9001 - val_loss: 0.2495 - val_accuracy: 0.9001 - va
l_auc: 0.9105
Epoch 6/100
53/53 [==============================] - 1s 11ms/step - loss: 0.2442 - acc
uracy: 0.8944 - auc: 0.9142 - val_loss: 0.2308 - val_accuracy: 0.9019 - va
l_auc: 0.9222
Epoch 7/100
53/53 [==============================] - 1s 10ms/step - loss: 0.2300 - acc
uracy: 0.9026 - auc: 0.9240 - val_loss: 0.2216 - val_accuracy: 0.9026 - va
l_auc: 0.9288
Epoch 8/100
53/53 [==============================] - 1s 11ms/step - loss: 0.2230 - acc
uracy: 0.9070 - auc: 0.9296 - val_loss: 0.2103 - val_accuracy: 0.9135 - va
l_auc: 0.9335
Epoch 9/100
53/53 [==============================] - 1s 10ms/step - loss: 0.2167 - acc
uracy: 0.9090 - auc: 0.9327 - val_loss: 0.2044 - val_accuracy: 0.9160 - va
l_auc: 0.9364
Epoch 10/100
53/53 [==============================] - 1s 10ms/step - loss: 0.2075 - acc
uracy: 0.9143 - auc: 0.9360 - val_loss: 0.2031 - val_accuracy: 0.9197 - va
l_auc: 0.9388
Epoch 11/100
53/53 [==============================] - 1s 11ms/step - loss: 0.2034 - acc
uracy: 0.9162 - auc: 0.9383 - val_loss: 0.1971 - val_accuracy: 0.9182 - va
l_auc: 0.9405
Epoch 12/100
53/53 [==============================] - 1s 10ms/step - loss: 0.2016 - acc
uracy: 0.9166 - auc: 0.9399 - val_loss: 0.1953 - val_accuracy: 0.9226 - va
l_auc: 0.9418
Epoch 13/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1983 - acc
uracy: 0.9176 - auc: 0.9414 - val_loss: 0.1938 - val_accuracy: 0.9193 - va
l_auc: 0.9426
Epoch 14/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1974 - acc
uracy: 0.9183 - auc: 0.9423 - val_loss: 0.1905 - val_accuracy: 0.9241 - va
l_auc: 0.9440
Epoch 15/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1960 - acc
uracy: 0.9191 - auc: 0.9432 - val_loss: 0.1879 - val_accuracy: 0.9248 - va
l_auc: 0.9449
Epoch 16/100
```

```
53/53 [==============================] - 1s 10ms/step - loss: 0.1925 - acc
uracy: 0.9212 - auc: 0.9441 - val_loss: 0.1868 - val_accuracy: 0.9256 - va
l_auc: 0.9454
Epoch 17/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1937 - acc
uracy: 0.9207 - auc: 0.9451 - val_loss: 0.1913 - val_accuracy: 0.9256 - va
l_auc: 0.9462
Epoch 18/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1896 - acc
uracy: 0.9225 - auc: 0.9459 - val_loss: 0.1878 - val_accuracy: 0.9233 - va
l_auc: 0.9467
Epoch 19/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1943 - acc
uracy: 0.9197 - auc: 0.9462 - val_loss: 0.1856 - val_accuracy: 0.9243 - va
l_auc: 0.9470
Epoch 20/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1887 - acc
uracy: 0.9221 - auc: 0.9468 - val_loss: 0.1850 - val_accuracy: 0.9248 - va
l_auc: 0.9477
Epoch 21/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1862 - acc
uracy: 0.9243 - auc: 0.9472 - val_loss: 0.1846 - val_accuracy: 0.9281 - va
l_auc: 0.9477
Epoch 22/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1848 - acc
uracy: 0.9246 - auc: 0.9480 - val_loss: 0.1811 - val_accuracy: 0.9278 - va
l_auc: 0.9485
Epoch 23/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1833 - acc
uracy: 0.9261 - auc: 0.9485 - val_loss: 0.1896 - val_accuracy: 0.9249 - va
l_auc: 0.9487
Epoch 24/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1852 - acc
uracy: 0.9238 - auc: 0.9488 - val_loss: 0.1873 - val_accuracy: 0.9258 - va
l_auc: 0.9495
Epoch 25/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1823 - acc
uracy: 0.9259 - auc: 0.9495 - val_loss: 0.1875 - val_accuracy: 0.9255 - va
l_auc: 0.9499
Epoch 26/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1815 - acc
uracy: 0.9262 - auc: 0.9497 - val_loss: 0.1796 - val_accuracy: 0.9289 - va
l_auc: 0.9501
Epoch 27/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1809 - acc
uracy: 0.9266 - auc: 0.9504 - val_loss: 0.1788 - val_accuracy: 0.9286 - va
l_auc: 0.9504
Epoch 28/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1797 - acc
uracy: 0.9266 - auc: 0.9506 - val_loss: 0.1770 - val_accuracy: 0.9305 - va
l_auc: 0.9508
Epoch 29/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1794 - acc
uracy: 0.9274 - auc: 0.9510 - val_loss: 0.1768 - val_accuracy: 0.9299 - va
l_auc: 0.9511
Epoch 30/100
53/53 [==============================] - 1s 13ms/step - loss: 0.1810 - acc
uracy: 0.9265 - auc: 0.9510 - val_loss: 0.1761 - val_accuracy: 0.9307 - va
l_auc: 0.9514
Epoch 31/100
53/53 [==============================] - 1s 13ms/step - loss: 0.1778 - acc
```

```
uracy: 0.9281 - auc: 0.9516 - val_loss: 0.1794 - val_accuracy: 0.9287 - va
l_auc: 0.9516
Epoch 32/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1793 - acc
uracy: 0.9271 - auc: 0.9519 - val_loss: 0.1888 - val_accuracy: 0.9234 - va
l_auc: 0.9518
Epoch 33/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1805 - acc
uracy: 0.9265 - auc: 0.9521 - val_loss: 0.1757 - val_accuracy: 0.9299 - va
l_auc: 0.9518
Epoch 34/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1772 - acc
uracy: 0.9286 - auc: 0.9528 - val_loss: 0.1750 - val_accuracy: 0.9304 - va
l_auc: 0.9518
Epoch 35/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1788 - acc
uracy: 0.9275 - auc: 0.9527 - val_loss: 0.1810 - val_accuracy: 0.9275 - va
l_auc: 0.9520
Epoch 36/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1785 - acc
uracy: 0.9271 - auc: 0.9526 - val_loss: 0.1762 - val_accuracy: 0.9304 - va
l_auc: 0.9525
Epoch 37/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1822 - acc
uracy: 0.9256 - auc: 0.9527 - val_loss: 0.1735 - val_accuracy: 0.9314 - va
l_auc: 0.9526
Epoch 38/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1798 - acc
uracy: 0.9269 - auc: 0.9535 - val_loss: 0.1734 - val_accuracy: 0.9315 - va
l_auc: 0.9528
Epoch 39/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1756 - acc
uracy: 0.9291 - auc: 0.9531 - val_loss: 0.1777 - val_accuracy: 0.9295 - va
l_auc: 0.9531
Epoch 40/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1731 - acc
uracy: 0.9305 - auc: 0.9539 - val_loss: 0.1751 - val_accuracy: 0.9306 - va
l_auc: 0.9532
Epoch 41/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1732 - acc
uracy: 0.9300 - auc: 0.9542 - val_loss: 0.1736 - val_accuracy: 0.9312 - va
l_auc: 0.9531
Epoch 42/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1742 - acc
uracy: 0.9299 - auc: 0.9539 - val_loss: 0.1805 - val_accuracy: 0.9267 - va
l_auc: 0.9535
Epoch 43/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1727 - acc
uracy: 0.9301 - auc: 0.9545 - val_loss: 0.1728 - val_accuracy: 0.9317 - va
l_auc: 0.9538
Epoch 44/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1735 - acc
uracy: 0.9300 - auc: 0.9546 - val_loss: 0.1731 - val_accuracy: 0.9314 - va
l_auc: 0.9540
Epoch 45/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1743 - acc
uracy: 0.9296 - auc: 0.9552 - val_loss: 0.1709 - val_accuracy: 0.9332 - va
l_auc: 0.9541
Epoch 46/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1752 - acc
uracy: 0.9291 - auc: 0.9549 - val_loss: 0.1785 - val_accuracy: 0.9288 - va
```

```
l_auc: 0.9540
Epoch 47/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1768 - acc
uracy: 0.9286 - auc: 0.9553 - val_loss: 0.1708 - val_accuracy: 0.9327 - va
l_auc: 0.9541
Epoch 48/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1740 - acc
uracy: 0.9294 - auc: 0.9553 - val_loss: 0.1728 - val_accuracy: 0.9314 - va
l_auc: 0.9544
Epoch 49/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1734 - acc
uracy: 0.9296 - auc: 0.9559 - val_loss: 0.1715 - val_accuracy: 0.9319 - va
l_auc: 0.9546
Epoch 50/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1707 - acc
uracy: 0.9306 - auc: 0.9557 - val_loss: 0.1727 - val_accuracy: 0.9314 - va
l_auc: 0.9546
Epoch 51/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1715 - acc
uracy: 0.9303 - auc: 0.9556 - val_loss: 0.1701 - val_accuracy: 0.9325 - va
l_auc: 0.9547
Epoch 52/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1735 - acc
uracy: 0.9300 - auc: 0.9558 - val_loss: 0.1800 - val_accuracy: 0.9265 - va
l_auc: 0.9548
Epoch 53/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1731 - acc
uracy: 0.9297 - auc: 0.9561 - val_loss: 0.1695 - val_accuracy: 0.9332 - va
l_auc: 0.9550
Epoch 54/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1745 - acc
uracy: 0.9294 - auc: 0.9563 - val_loss: 0.1784 - val_accuracy: 0.9289 - va
l_auc: 0.9547
Epoch 55/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1727 - acc
uracy: 0.9294 - auc: 0.9563 - val_loss: 0.1751 - val_accuracy: 0.9313 - va
l_auc: 0.9550
Epoch 56/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1690 - acc
uracy: 0.9320 - auc: 0.9567 - val_loss: 0.1700 - val_accuracy: 0.9333 - va
l_auc: 0.9553
Epoch 57/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1705 - acc
uracy: 0.9310 - auc: 0.9570 - val_loss: 0.1713 - val_accuracy: 0.9317 - va
l_auc: 0.9554
Epoch 58/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1689 - acc
uracy: 0.9315 - auc: 0.9572 - val_loss: 0.1923 - val_accuracy: 0.9201 - va
l_auc: 0.9553
Epoch 59/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1690 - acc
uracy: 0.9313 - auc: 0.9575 - val_loss: 0.1685 - val_accuracy: 0.9334 - va
l_auc: 0.9555
Epoch 60/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1676 - acc
uracy: 0.9329 - auc: 0.9570 - val_loss: 0.1692 - val_accuracy: 0.9330 - va
l_auc: 0.9553
Epoch 61/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1687 - acc
uracy: 0.9313 - auc: 0.9572 - val_loss: 0.1692 - val_accuracy: 0.9336 - va
l_auc: 0.9553
```

```
Epoch 62/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1709 - acc
uracy: 0.9312 - auc: 0.9574 - val_loss: 0.1799 - val_accuracy: 0.9278 - va
l_auc: 0.9559
Epoch 63/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1687 - acc
uracy: 0.9318 - auc: 0.9579 - val_loss: 0.1679 - val_accuracy: 0.9337 - va
l_auc: 0.9560
Epoch 64/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1679 - acc
uracy: 0.9322 - auc: 0.9579 - val_loss: 0.1682 - val_accuracy: 0.9331 - va
l_auc: 0.9558
Epoch 65/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1676 - acc
uracy: 0.9326 - auc: 0.9580 - val_loss: 0.1733 - val_accuracy: 0.9309 - va
l_auc: 0.9561
Epoch 66/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1675 - acc
uracy: 0.9326 - auc: 0.9582 - val_loss: 0.1782 - val_accuracy: 0.9280 - va
l_auc: 0.9560
Epoch 67/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1666 - acc
uracy: 0.9327 - auc: 0.9584 - val_loss: 0.1710 - val_accuracy: 0.9321 - va
l_auc: 0.9560
Epoch 68/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1734 - acc
uracy: 0.9304 - auc: 0.9585 - val_loss: 0.1715 - val_accuracy: 0.9322 - va
l_auc: 0.9557
Epoch 69/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1675 - acc
uracy: 0.9330 - auc: 0.9586 - val_loss: 0.1675 - val_accuracy: 0.9344 - va
l_auc: 0.9561
Epoch 70/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1703 - acc
uracy: 0.9303 - auc: 0.9588 - val_loss: 0.1670 - val_accuracy: 0.9333 - va
l_auc: 0.9563
Epoch 71/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1657 - acc
uracy: 0.9328 - auc: 0.9586 - val_loss: 0.1699 - val_accuracy: 0.9321 - va
l_auc: 0.9566
Epoch 72/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1675 - acc
uracy: 0.9329 - auc: 0.9585 - val_loss: 0.1714 - val_accuracy: 0.9326 - va
l_auc: 0.9564
Epoch 73/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1680 - acc
uracy: 0.9323 - auc: 0.9595 - val_loss: 0.1700 - val_accuracy: 0.9325 - va
l_auc: 0.9564
Epoch 74/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1670 - acc
uracy: 0.9319 - auc: 0.9591 - val_loss: 0.1884 - val_accuracy: 0.9255 - va
l_auc: 0.9559
Epoch 75/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1685 - acc
uracy: 0.9323 - auc: 0.9594 - val_loss: 0.1672 - val_accuracy: 0.9345 - va
l_auc: 0.9566
Epoch 76/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1661 - acc
uracy: 0.9334 - auc: 0.9596 - val_loss: 0.1711 - val_accuracy: 0.9323 - va
l_auc: 0.9565
Epoch 77/100
```

```
53/53 [==============================] - 1s 11ms/step - loss: 0.1649 - acc
uracy: 0.9339 - auc: 0.9597 - val_loss: 0.1773 - val_accuracy: 0.9279 - va
l_auc: 0.9566
Epoch 78/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1656 - acc
uracy: 0.9329 - auc: 0.9601 - val_loss: 0.1707 - val_accuracy: 0.9323 - va
l_auc: 0.9570
Epoch 79/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1639 - acc
uracy: 0.9346 - auc: 0.9596 - val_loss: 0.1664 - val_accuracy: 0.9350 - va
l_auc: 0.9569
Epoch 80/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1620 - acc
uracy: 0.9347 - auc: 0.9601 - val_loss: 0.1658 - val_accuracy: 0.9348 - va
l_auc: 0.9570
Epoch 81/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1634 - acc
uracy: 0.9340 - auc: 0.9601 - val_loss: 0.1719 - val_accuracy: 0.9319 - va
l_auc: 0.9568
Epoch 82/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1634 - acc
uracy: 0.9344 - auc: 0.9603 - val_loss: 0.1659 - val_accuracy: 0.9345 - va
l_auc: 0.9572
Epoch 83/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1657 - acc
uracy: 0.9332 - auc: 0.9600 - val_loss: 0.1806 - val_accuracy: 0.9262 - va
l_auc: 0.9569
Epoch 84/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1623 - acc
uracy: 0.9344 - auc: 0.9604 - val_loss: 0.1656 - val_accuracy: 0.9344 - va
l_auc: 0.9577
Epoch 85/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1603 - acc
uracy: 0.9351 - auc: 0.9605 - val_loss: 0.1654 - val_accuracy: 0.9345 - va
l_auc: 0.9571
Epoch 86/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1621 - acc
uracy: 0.9343 - auc: 0.9606 - val_loss: 0.1664 - val_accuracy: 0.9346 - va
l_auc: 0.9575
Epoch 87/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1632 - acc
uracy: 0.9340 - auc: 0.9609 - val_loss: 0.1792 - val_accuracy: 0.9274 - va
l_auc: 0.9574
Epoch 88/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1614 - acc
uracy: 0.9353 - auc: 0.9611 - val_loss: 0.1668 - val_accuracy: 0.9343 - va
l_auc: 0.9576
Epoch 89/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1595 - acc
uracy: 0.9352 - auc: 0.9610 - val_loss: 0.1681 - val_accuracy: 0.9338 - va
l_auc: 0.9576
Epoch 90/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1606 - acc
uracy: 0.9353 - auc: 0.9613 - val_loss: 0.1677 - val_accuracy: 0.9339 - va
l_auc: 0.9577
Epoch 91/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1596 - acc
uracy: 0.9360 - auc: 0.9614 - val_loss: 0.1675 - val_accuracy: 0.9338 - va
l_auc: 0.9574
Epoch 92/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1612 - acc
```

```
uracy: 0.9351 - auc: 0.9614 - val_loss: 0.1650 - val_accuracy: 0.9349 - va
l_auc: 0.9581
Epoch 93/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1615 - acc
uracy: 0.9345 - auc: 0.9620 - val_loss: 0.1642 - val_accuracy: 0.9356 - va
l_auc: 0.9579
Epoch 94/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1623 - acc
uracy: 0.9343 - auc: 0.9619 - val_loss: 0.1667 - val_accuracy: 0.9344 - va
l_auc: 0.9574
Epoch 95/100
53/53 [==============================] - 1s 10ms/step - loss: 0.1614 - acc
uracy: 0.9343 - auc: 0.9616 - val_loss: 0.1649 - val_accuracy: 0.9350 - va
l_auc: 0.9575
Epoch 96/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1595 - acc
uracy: 0.9359 - auc: 0.9620 - val_loss: 0.1799 - val_accuracy: 0.9264 - va
l_auc: 0.9583
Epoch 97/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1619 - acc
uracy: 0.9349 - auc: 0.9616 - val_loss: 0.1739 - val_accuracy: 0.9307 - va
l_auc: 0.9579
Epoch 98/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1654 - acc
uracy: 0.9329 - auc: 0.9623 - val_loss: 0.1708 - val_accuracy: 0.9325 - va
l_auc: 0.9580
Epoch 99/100
53/53 [==============================] - 1s 11ms/step - loss: 0.1636 - acc
uracy: 0.9338 - auc: 0.9624 - val_loss: 0.1647 - val_accuracy: 0.9340 - va
l_auc: 0.9586
Epoch 100/100
53/53 [==============================] - 1s 12ms/step - loss: 0.1607 - acc
uracy: 0.9356 - auc: 0.9622 - val_loss: 0.1734 - val_accuracy: 0.9305 - va
l_auc: 0.9579
```

Out[28]:

```
<tensorflow.python.keras.callbacks.History at 0x7f201d823048>
```

In [29]:

```
! mkdir -p saved_model
model.save('saved_model/model_bert_classifier_latest')
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/
python/ops/resource_variable_ops.py:1817: calling BaseResourceVariable.__i
nit__ (from tensorflow.python.ops.resource_variable_ops) with constraint i
s deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```


```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/
python/ops/resource_variable_ops.py:1817: calling BaseResourceVariable.__i
nit__ (from tensorflow.python.ops.resource_variable_ops) with constraint i
s deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```


```
INFO:tensorflow:Assets written to: saved_model/model_bert_classifier_lates
t/assets
```


```
INFO:tensorflow:Assets written to: saved_model/model_bert_classifier_lates
t/assets
```

Part-6: Creating a Data pipeline for BERT Model

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

</pre>

In [30]:

```
#Read the dataset - Amazon fine food reviews
reviews_test = pd.read_csv(r"/content/test.csv")

reviews_test.Text.apply(lambda x : re.sub('<[^<]+?>', '', str(reviews_test['Text'])))

#Tokenization
X_test_tokens, X_test_mask, X_test_segment = convert_sentences_to_features(reviews_test
['Text'], tokenizer, 55)

#Embedding from bert
X_test_output=bert_model.predict([X_test_tokens,X_test_mask,X_test_segment])

#Evaluate neural network model
print(X_test_output.shape)
y_pred=model.predict_classes(X_test_output)
print(y_pred)
```

```
(352, 768)
WARNING:tensorflow:From <ipython-input-30-713aba0e91b9>:14: Sequential.pre
dict_classes (from tensorflow.python.keras.engine.sequential) is deprecate
d and will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your mod
el does multi-class classification   (e.g. if it uses a `softmax` last-lay
er activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your mod
el does binary classification   (e.g. if it uses a `sigmoid` last-layer ac
tivation).
```


```
WARNING:tensorflow:From <ipython-input-30-713aba0e91b9>:14: Sequential.pre
dict_classes (from tensorflow.python.keras.engine.sequential) is deprecate
d and will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your mod
el does multi-class classification   (e.g. if it uses a `softmax` last-lay
er activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your mod
el does binary classification   (e.g. if it uses a `sigmoid` last-layer ac
tivation).
```

```
[[0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [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]
[1]
[1]
[0]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[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]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
```

```
[1]
[1]
[1]
[1]
[0]
[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]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
```

```
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
```

```
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]
 [1]]
```

In [37]:

```python
y_1=np.count_nonzero(y_pred, axis=0)
y_0=len(y_pred)- y_1

print("Total Positive reviews {}".format(y_1))
print("Total Negative reviews {}".format(y_0))
```

```
Total Positive reviews [319]
Total Negative reviews [33]
```

Summary

1.Pre process the Amazon fine food review dataset.Extract the TEXt and Score features and remove special characters

2.Create a BERT model by downloading pre trained model from Tensor HUB

3.Tokenenize the preprocessed reviews dataset using tokenizer.py file, convert the Text feature to tokenized features understandable by the BERT model

4.For the tokenized reviews get the pre trained embedding vectors using the defined BERT model

5.Use this embedding vector as input to NN model binary classifier

6.Hyper paramater tune the NN model to avoid over fitting and under fitting

7.Use the trained model to predict the classification of reviews

8.Use the test reviews datset,preprocess,Tokenize,get BERT embedding vectors,predict the class of embedding vectors using the trained NN model.

9.Display the predicted classes.