BERT

Created by: Harsh Bari

From: SVNIT, Gujarat

Mtech Data Science - p23ds004 (2023-25)

Subject: NLP Project

Last Updated:

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data = pd.read_csv("dataset.csv")
```

In [3]:
```
data
```

Out[3]:

	tweets	class	target	bert_vectors
0	Be aware dirty step to get money #staylight ...	figurative	0.0	[0.5142972 0.09474187 0.41039863 -0.086846...
1	#sarcasm for #people who don't understand #diy...	figurative	0.0	[0.60709447 0.24507785 0.30069906 -0.095903...
2	@IminworkJeremy @medsingle #DailyMail readers ...	figurative	0.0	[0.12725917 0.07618354 0.5803481 0.085881...
3	@wilw Why do I get the feeling you like games?...	figurative	0.0	[4.73182350e-01 5.66805065e-01 3.70653301e-...
4	-@TeacherArthurG @rweingarten You probably jus...	figurative	0.0	[0.50476366 0.16474849 0.34438178 0.067346...
...	...	...	...	...
81403	Photo: Image via We Heart It http://t.co/ky8Nf...	sarcasm	1.0	[2.27706909e-01 -4.00034375e-02 5.59347928e-...
81404	I never knew..I better put this out to the Uni...	sarcasm	1.0	[0.21134636 0.16012657 0.7607299 -0.195773...
81405	hey just wanted to say thanks @ puberty for le...	sarcasm	1.0	[0.39612132 0.26797482 0.8456364 -0.222941...
81406	I'm sure coverage like the Fox News Special “T...	sarcasm	1.0	[0.35054675 -0.17652354 0.36843812 -0.002357...
81407	@skeyno16 at u13?! I won't believe it until I ...	sarcasm	1.0	[0.34130692 0.04276937 0.69224215 -0.030625...

81408 rows × 4 columns

In [4]:
```
data['bert_vectors']
```

Out[4]:
```
0        [ 0.5142972   0.09474187  0.41039863 -0.086846...
1        [ 0.60709447  0.24507785  0.30069906 -0.095903...
2        [ 0.12725917  0.07618354  0.5803481   0.085881...
3        [ 4.73182350e-01  5.66805065e-01  3.70653301e-...
4        [ 0.50476366  0.16474849  0.34438178  0.067346...
                              ...
81403    [ 2.27706909e-01 -4.00034375e-02  5.59347928e-...
81404    [ 0.21134636  0.16012657  0.7607299  -0.195773...
81405    [ 0.39612132  0.26797482  0.8456364  -0.222941...
81406    [ 0.35054675 -0.17652354  0.36843812 -0.002357...
81407    [ 0.34130692  0.04276937  0.69224215 -0.030625...
Name: bert_vectors, Length: 81408, dtype: object
```

In [5]:
```
data['bert_vectors'][0]
```

Out[5]:
```
'[ 0.5142972   0.09474187  0.41039863 -0.08684609  0.2877095  -0.16383709
\n  0.40892476  0.44171068 -0.01723384 -0.26785442 -0.16827619 -0.22784133
\n -0.31953806  0.275655   -0.10393742  0.49432114 -0.02620821  0.2063558
\n -0.23674724  0.25816718  0.34353077  0.0214939  -0.13152978  0.11771113
\n  0.49273565 -0.15442434  0.10735898 -0.15738115 -0.3916646  -0.16478181
\n  0.49133274 -0.04270812  0.237843   -0.20905466  0.06037405 -0.17967163
\n -0.31194413 -0.0389487   0.29511118  0.44777897 -0.36570454 -0.5686758
\n  0.19408512  0.02260724 -0.31137243 -0.28884372  0.2681975   0.26582816
\n  0.33216658 -0.15465686 -0.2563196   0.3087412  -0.09471447 -0.08942274
\n  0.0777904   0.74242663  0.3745379  -0.45100695 -0.43329164 -0.02466331
\n -0.16693549  0.08048202  0.26112032 -0.30901015  0.12328291  0.21376407
\n -0.0388822   0.6823772  -1.0520166   0.28137955 -0.01561841 -0.03059365
\n -0.06478383  0.13605814  0.01955121  0.20722364 -0.10848905  0.6780405
\n  0.31807926 -0.0370811   0.05020353  0.16420896  0.10430315  0.38346\n
0.19076318  0.27409777  0.11541953 -0.00839414 -0.60080236  0.47035944\n -
0.05057348  0.08532551  0.19232303  0.20832938  0.40885463 -0.42770848\n
0.01001668 -0.3563449  -0.24444808  0.3981791   0.30680826 -0.26975936\n -
0.02268203  0.10180932 -0.46551192  0.16729726 -0.01927103 -0.12172183\n
0.12085328  0.32595316  0.14308026  0.17479685 -0.05908208 -0.4421207\n
0.24812964  0.16455555  0.15420035  0.206105    0.23889849  0.03915282\n -
0.02392515  0.22758889  0.21866032  0.78823656  0.20028736  0.3199013\n -
0.40904292  0.2744289   0.16341347  0.12637198  0.41490802  0.29418418\n -
0.08437694 -0.54106766 -0.11171681 -0.07831711 -0.02037072 -0.10411447\n -
0.11889234 -0.14834784 -0.08240354 -0.39731655  0.12649344  0.1883826\n
0.09964897 -0.01768736  0.12272178  0.03364825 -0.17410624  0.5015001 ]'
```

In [6]:
```
data['class'].value_counts()
```

Out[6]:
```
class
figurative    21238
irony         20894
sarcasm       20681
regular       18595
Name: count, dtype: int64
```

In [7]:
```
print(type(data['bert_vectors'][0]))
```

```
<class 'str'>
```

In [8]:
```
def str_to_vec(input_string):
    # Your input string
    # input_string = "[ 0.47350284 0.17998783 0.52607228 -0.21903914 0.5047

    # Remove brackets and newline characters
    cleaned_string = input_string.replace('[', '').replace(']', '').replace

    # Split the string into a list of strings
    string_values = cleaned_string.split()

    # Convert each string value to a float
    float_values = [float(value) for value in string_values]

    # Convert the list of floats to a NumPy array
    vector = np.array(float_values)

    # Now, 'vector' is a NumPy array representing your vector
    return vector
```

In [9]:
```
dataset = pd.DataFrame()
```

In [10]:
```
dataset['bert_vectors'] = data['bert_vectors'].apply(str_to_vec)
```

In [11]:
```
dataset
```

Out[11]:

	bert_vectors
0	[0.5142972, 0.09474187, 0.41039863, -0.0868460...
1	[0.60709447, 0.24507785, 0.30069906, -0.095903...
2	[0.12725917, 0.07618354, 0.5803481, 0.0858818,...
3	[0.47318235, 0.566805065, 0.370653301, -0.0819...
4	[0.50476366, 0.16474849, 0.34438178, 0.0673463...
...	...
81403	[0.227706909, -0.0400034375, 0.559347928, -0.1...
81404	[0.21134636, 0.16012657, 0.7607299, -0.1957732...
81405	[0.39612132, 0.26797482, 0.8456364, -0.2229413...
81406	[0.35054675, -0.17652354, 0.36843812, -0.00235...
81407	[0.34130692, 0.04276937, 0.69224215, -0.030625...

81408 rows × 1 columns

In [12]:
```
print(len(data['bert_vectors']))
```

```
81408
```

In [13]:
```
dataset['bert_vectors'][0]
```

Out[13]:
```
array([ 0.5142972 ,  0.09474187,  0.41039863, -0.08684609,  0.2877095 ,
       -0.16383709,  0.40892476,  0.44171068, -0.01723384, -0.26785442,
       -0.16827619, -0.22784133, -0.31953806,  0.275655  , -0.10393742,
        0.49432114, -0.02620821,  0.2063558 , -0.23674724,  0.25816718,
        0.34353077,  0.0214939 , -0.13152978,  0.11771113,  0.49273565,
       -0.15442434,  0.10735898, -0.15738115, -0.3916646 , -0.16478181,
        0.49133274, -0.04270812,  0.237843  , -0.20905466,  0.06037405,
       -0.17967163, -0.31194413, -0.0389487 ,  0.29511118,  0.44777897,
       -0.36570454, -0.5686758 ,  0.19408512,  0.02260724, -0.31137243,
       -0.28884372,  0.2681975 ,  0.26582816,  0.33216658, -0.15465686,
       -0.2563196 ,  0.3087412 , -0.09471447, -0.08942274,  0.0777904 ,
        0.74242663,  0.3745379 , -0.45100695, -0.43329164, -0.02466331,
       -0.16693549,  0.08048202,  0.26112032, -0.30901015,  0.12328291,
        0.21376407, -0.0388822 ,  0.6823772 , -1.0520166 ,  0.28137955,
       -0.01561841, -0.03059365, -0.06478383,  0.13605814,  0.01955121,
        0.20722364, -0.10848905,  0.6780405 ,  0.31807926, -0.0370811 ,
        0.05020353,  0.16420896,  0.10430315,  0.38346   ,  0.19076318,
        0.27409777,  0.11541953, -0.00839414, -0.60080236,  0.47035944,
       -0.05057348,  0.08532551,  0.19232303,  0.20832938,  0.40885463,
       -0.42770848,  0.01001668, -0.3563449 , -0.24444808,  0.3981791 ,
        0.30680826, -0.26975936, -0.02268203,  0.10180932, -0.46551192,
        0.16729726, -0.01927103, -0.12172183,  0.12085328,  0.32595316,
        0.14308026,  0.17479685, -0.05908208, -0.4421207 ,  0.24812964,
        0.16455555,  0.15420035,  0.206105  ,  0.23889849,  0.03915282,
       -0.02392515,  0.22758889,  0.21866032,  0.78823656,  0.20028736,
        0.3199013 , -0.40904292,  0.2744289 ,  0.16341347,  0.12637198,
        0.41490802,  0.29418418, -0.08437694, -0.54106766, -0.11171681,
       -0.07831711, -0.02037072, -0.10411447, -0.11889234, -0.14834784,
       -0.08240354, -0.39731655,  0.12649344,  0.1883826 ,  0.09964897,
       -0.01768736,  0.12272178,  0.03364825, -0.17410624,  0.5015001 ])
```

Create Input and Output

In [14]:
```
def create_input_vectors(vector):
    n = len(vector)
    array_2d = np.empty((n, len(vector[0])), dtype=object)

    # Create and insert 10 random 5D arrays into the 2D array
    for i in range(n):
        array_150d = vector[i]
        array_2d[i] = array_150d

    return array_2d
```

In [15]:
```
input_vec = create_input_vectors(dataset['bert_vectors'])
```

In [16]:
```
print(type(input_vec))
```

```
<class 'numpy.ndarray'>
```

In [17]:
```
print(input_vec.ndim)
```

```
2
```

In [18]:
```
mapping_dict = {'figurative': 0, 'irony': 1, 'sarcasm': 1, 'regular': 0}

# Use map to replace string values with numerical values
output = data['class'].map(mapping_dict).to_numpy()
```

In [19]:
```
print(output.ndim)
```

```
1
```

In [20]:
```
print(type(output))
```

```
<class 'numpy.ndarray'>
```

In [21]:
```
output
```

Out[21]:
```
array([0, 0, 0, ..., 1, 1, 1], dtype=int64)
```

Data Splitting

In [22]:
```
def split_data(array_2d, ranges_to_copy):
    copied_ranges = []

    # Loop through each range and copy the corresponding elements
    for start, end in ranges_to_copy:
        copied_range = array_2d[start:end+1]  # Adjust end index to include
        copied_ranges.append(copied_range)

    # Concatenate the copied ranges along the first axis to create the fina
    copied_array = np.concatenate(copied_ranges, axis=0)

    return copied_array
```

In [23]:
```
x_train = split_data(input_vec, [(0, 16989), (21238, 37952), (42132, 57007)
x_test = split_data(input_vec, [(16990, 21237), (37953, 42131), (57008, 607
```

In [24]:
```
print("x train:", len(x_train))
print("x test:", len(x_test))
print("Total:", len(x_train) + len(x_test))
```

```
x train: 65125
x test: 16283
Total: 81408
```

In [25]:
```
y_train = np.concatenate((np.zeros(16990), np.ones(31591), np.zeros(16544))
y_test = np.concatenate((np.zeros(4248), np.ones(7898), np.zeros(4137)))
```

In [26]:
```
print("train:", len(y_train))
print("test:", len(y_test))
print("total:", len(y_train) + len(y_test))
```

```
train: 65125
test: 16283
total: 81408
```

Training With Neural Network

In [27]:
```
import tensorflow as tf
from tensorflow import keras
```

```
WARNING:tensorflow:From C:\Users\Harsh Bari\AppData\Local\Programs\Python
\Python310\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.
sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losse
s.sparse_softmax_cross_entropy instead.
```

Neural Network for Average Word Embedding

In [28]:
```
awe = keras.Sequential([
    keras.layers.Dense(256, input_shape = (150, ), activation = 'relu'),
    keras.layers.Dense(128, activation = 'relu'),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dense(32, activation = 'relu'),
    keras.layers.Dense(16, activation=keras.layers.LeakyReLU(alpha=0.1)),
    keras.layers.Dense(8, activation=keras.layers.LeakyReLU(alpha=0.1)),
    keras.layers.Dense(2, activation = 'sigmoid')

])

awe.compile(optimizer = 'adam',
                      loss = 'sparse_categorical_crossentropy',
                      metrics = ['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\Harsh Bari\AppData\Local\Programs\Python
\Python310\lib\site-packages\keras\src\backend.py:873: The name tf.get_def
ault_graph is deprecated. Please use tf.compat.v1.get_default_graph instea
d.

WARNING:tensorflow:From C:\Users\Harsh Bari\AppData\Local\Programs\Python
\Python310\lib\site-packages\keras\src\optimizers\__init__.py:309: The nam
e tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimize
r instead.
```

Check Model Summary

In [29]:
```
awe.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               38656

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 8)                 136

 dense_6 (Dense)             (None, 2)                 18

=================================================================
Total params: 82570 (322.54 KB)
Trainable params: 82570 (322.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Train Model

In [30]:
```
awe.fit(x_train.astype(np.float32), y_train.astype(np.float32), epochs=22)
```

```
Epoch 1/22
WARNING:tensorflow:From C:\Users\Harsh Bari\AppData\Local\Programs\Python
\Python310\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.
ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.Rag
gedTensorValue instead.

WARNING:tensorflow:From C:\Users\Harsh Bari\AppData\Local\Programs\Python
\Python310\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The
name tf.executing_eagerly_outside_functions is deprecated. Please use tf.c
ompat.v1.executing_eagerly_outside_functions instead.

2036/2036 [==============================] - 9s 3ms/step - loss: 0.4457 -
accuracy: 0.7695
Epoch 2/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.4040 -
accuracy: 0.7932
Epoch 3/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3901 -
accuracy: 0.7989
Epoch 4/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3812 -
accuracy: 0.8031
Epoch 5/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3734 -
accuracy: 0.8069
Epoch 6/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3664 -
accuracy: 0.8109
Epoch 7/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3597 -
accuracy: 0.8131
Epoch 8/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3552 -
accuracy: 0.8154
Epoch 9/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3509 -
accuracy: 0.8165
Epoch 10/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3450 -
accuracy: 0.8195
Epoch 11/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3401 -
accuracy: 0.8212
Epoch 12/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3355 -
accuracy: 0.8237
Epoch 13/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3319 -
accuracy: 0.8258
Epoch 14/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3267 -
accuracy: 0.8286
Epoch 15/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3242 -
accuracy: 0.8302
Epoch 16/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3197 -
accuracy: 0.8313
Epoch 17/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3164 -
accuracy: 0.8338
```

```
Epoch 18/22
2036/2036 [==============================] - 6s 3ms/step - loss: 0.3128 -
accuracy: 0.8339
Epoch 19/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3102 -
accuracy: 0.8360
Epoch 20/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3063 -
accuracy: 0.8383
Epoch 21/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3048 -
accuracy: 0.8376
Epoch 22/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3024 -
accuracy: 0.8391
```

Out[30]: <keras.src.callbacks.History at 0x152a6742e90>

Training Accuracy

In [31]:
```
awe.evaluate(x_train.astype(np.float32), y_train.astype(np.float32))
```

```
2036/2036 [==============================] - 6s 3ms/step - loss: 0.2876 -
accuracy: 0.8467
```

Out[31]: [0.28764575719833374, 0.8467255234718323]

Testing Accuracy

In [32]:
```
prediction = awe.predict(x_test.astype(np.float32))
```

```
509/509 [==============================] - 1s 2ms/step
```

In [33]:
```
prediction = np.argmax(prediction, axis = 1)
```

In [34]:
```
from sklearn.metrics import classification_report, confusion_matrix, accura
```

In [35]:
```
print(classification_report(y_test.astype(np.float32), prediction))
print()
print("Confusion Matrix: \n", confusion_matrix(y_test.astype(np.float32), p
print("\nAccuracy: \n", accuracy_score(y_test.astype(np.float32), predictio
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.76      0.81      8385
         1.0       0.77      0.90      0.83      7898

    accuracy                           0.82     16283
   macro avg       0.83      0.83      0.82     16283
weighted avg       0.83      0.82      0.82     16283


Confusion Matrix:
 [[6331 2054]
 [ 827 7071]]

Accuracy:
 0.8230670023951361
```