Average Word Vector

Created by: Harsh Bari

From: SVNIT, Gujarat

Mtech Data Science - p23ds004 (2023-25)

Subject: NLP Project

Last Updated: 29/03/2024

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

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

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

Out[3]:

	tweets	class	tokens	word2vec
0	Be aware dirty step to get money #staylight ...	figurative	['be', 'aware', 'dirty', 'step', 'to', 'get', ...	[-0.08192246 0.44799908 0.19038523 -0.092749...
1	#sarcasm for #people who don't understand #diy...	figurative	['sarcasm', 'for', 'people', 'who', 'don', 'un...	[-0.39415668 0.14868878 -0.30105257 -0.247197...
2	@IminworkJeremy @medsingle #DailyMail readers ...	figurative	['iminworkjeremy', 'medsingle', 'dailymail', '...	[-0.38441235 0.36639656 -0.08122431 0.066615...
3	@wilw Why do I get the feeling you like games?...	figurative	['wilw', 'why', 'do', 'get', 'the', 'feeling',...	[-0.46358013 0.12668053 0.13927908 0.171643...
4	-@TeacherArthurG @rweingarten You probably jus...	figurative	['teacherarthurg', 'rweingarten', 'you', 'prob...	[-0.19652244 0.19875612 -0.15998856 -0.170760...
...	...	...	...	...
81403	Photo: Image via We Heart It http://t.co/ky8Nf...	sarcasm	['photo', 'image', 'via', 'we', 'heart', 'it',...	[-0.2547767 0.29173678 0.06745852 0.089999...
81404	I never knew..I better put this out to the Uni...	sarcasm	['never', 'knew', 'better', 'put', 'this', 'ou...	[-0.32126068 0.12298504 0.13224294 0.150191...
81405	hey just wanted to say thanks @ puberty for le...	sarcasm	['hey', 'just', 'wanted', 'to', 'say', 'thanks...	[-2.33878006e-01 1.36348982e-01 -7.76878343e-...
81406	I'm sure coverage like the Fox News Special “T...	sarcasm	['sure', 'coverage', 'like', 'the', 'fox', 'ne...	[-0.12494273 0.00411794 0.02370966 0.203509...
81407	@skeyno16 at u13?! I won't believe it until I ...	sarcasm	['skeyno', 'at', 'won', 'believe', 'it', 'unti...	[-0.0282643 -0.10591727 0.2379264 0.112892...

81408 rows × 4 columns

In [4]:
```
data['word2vec']
```

Out[4]:
```
0        [-0.08192246  0.44799908  0.19038523 -0.092749...
1        [-0.39415668  0.14868878 -0.30105257 -0.247197...
2        [-0.38441235  0.36639656 -0.08122431  0.066615...
3        [-0.46358013  0.12668053  0.13927908  0.171643...
4        [-0.19652244  0.19875612 -0.15998856 -0.170760...
                              ...
81403    [-0.2547767   0.29173678  0.06745852  0.089999...
81404    [-0.32126068  0.12298504  0.13224294  0.150191...
81405    [-2.33878006e-01  1.36348982e-01 -7.76878343e-...
81406    [-0.12494273  0.00411794  0.02370966  0.203509...
81407    [-0.0282643  -0.10591727  0.2379264   0.112892...
Name: word2vec, Length: 81408, dtype: object
```

In [5]:
```
data['word2vec'][0]
```

Out[5]:
```
'[-0.08192246  0.44799908  0.19038523 -0.09274998 -0.08060522  0.13320853
\n  0.21015547  0.15539696 -0.23566357  0.21636164  0.0283454   0.28280759
\n  0.17291894 -0.13524526  0.04833376 -0.2451405  -0.01642638 -0.07876031
\n -0.05630376  0.14641642  0.00160968  0.06245695 -0.37698133  0.08342589
\n  0.05364852 -0.04640223 -0.03445123 -0.09461063  0.23467198  0.0450404
\n  0.0438591   0.21541848  0.38158504 -0.06952128  0.2148452  -0.02609837
\n -0.07601524 -0.10320545  0.04465004  0.03037429 -0.08223748 -0.00512799
\n  0.31767438  0.09984008 -0.27293106 -0.0822356   0.04032159  0.13450514
\n -0.01024036  0.07759087 -0.13633434  0.39601669 -0.18086388  0.07863257
\n -0.12142318 -0.18982297 -0.13750716  0.02909118 -0.23583742 -0.04784711
\n  0.01999614  0.20932461  0.0254652   0.26366392  0.00856102  0.08058868
\n -0.20782008 -0.1359599  -0.23500545  0.03740927  0.11888506  0.17102422
\n  0.21671027  0.09068703 -0.25568805  0.02423541  0.11943042 -0.51753351
\n -0.16202727  0.22873774  0.06798634  0.22152784  0.0840674  -0.04993204
\n -0.26843767  0.07260146 -0.30092282  0.15355953 -0.0522157  -0.10862874
\n  0.04560524 -0.16612259 -0.06226176 -0.11781129 -0.07521048  0.00399866
\n  0.31480429  0.12684158  0.0491577   0.03760581 -0.12408901  0.16498946
\n  0.0691526   0.41420899 -0.31107269 -0.19319433 -0.04998738 -0.13795248
\n  0.01528367 -0.06783712 -0.02698278 -0.30714476 -0.04999141  0.15458899
\n -0.03083627  0.34201905  0.23672756  0.31692924  0.0461333  -0.08079044
\n -0.08898347  0.17961326 -0.38170894 -0.23787375 -0.10015603  0.22698843
\n -0.15538733  0.39900836  0.1796477  -0.05579911 -0.1960262   0.07388732
\n -0.18282236  0.13711534 -0.09072778  0.05247565  0.38945165  0.25997175
\n -0.39301072 -0.19569835 -0.176655   -0.01782703  0.07201633  0.11289259
\n  0.04637413 -0.17495102 -0.09149356  0.12997283  0.04181852  0.0023226
2]'
```

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['word2vec'][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['word2vec'] = data['word2vec'].apply(str_to_vec)
```

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

Out[11]:

	word2vec
0	[-0.08192246, 0.44799908, 0.19038523, -0.09274...
1	[-0.39415668, 0.14868878, -0.30105257, -0.2471...
2	[-0.38441235, 0.36639656, -0.08122431, 0.06661...
3	[-0.46358013, 0.12668053, 0.13927908, 0.171643...
4	[-0.19652244, 0.19875612, -0.15998856, -0.1707...
...	...
81403	[-0.2547767, 0.29173678, 0.06745852, 0.0899995...
81404	[-0.32126068, 0.12298504, 0.13224294, 0.150191...
81405	[-0.233878006, 0.136348982, -0.0776878343, -0....
81406	[-0.12494273, 0.00411794, 0.02370966, 0.203509...
81407	[-0.0282643, -0.10591727, 0.2379264, 0.1128929...

81408 rows × 1 columns

In [12]:
```
print(len(data['word2vec']))
```

```
81408
```

In [13]:
```
dataset['word2vec'][0]
```

Out[13]:
```
array([-0.08192246,  0.44799908,  0.19038523, -0.09274998, -0.08060522,
        0.13320853,  0.21015547,  0.15539696, -0.23566357,  0.21636164,
        0.0283454 ,  0.28280759,  0.17291894, -0.13524526,  0.04833376,
       -0.2451405 , -0.01642638, -0.07876031, -0.05630376,  0.14641642,
        0.00160968,  0.06245695, -0.37698133,  0.08342589,  0.05364852,
       -0.04640223, -0.03445123, -0.09461063,  0.23467198,  0.0450404 ,
        0.0438591 ,  0.21541848,  0.38158504, -0.06952128,  0.2148452 ,
       -0.02609837, -0.07601524, -0.10320545,  0.04465004,  0.03037429,
       -0.08223748, -0.00512799,  0.31767438,  0.09984008, -0.27293106,
       -0.0822356 ,  0.04032159,  0.13450514, -0.01024036,  0.07759087,
       -0.13633434,  0.39601669, -0.18086388,  0.07863257, -0.12142318,
       -0.18982297, -0.13750716,  0.02909118, -0.23583742, -0.04784711,
        0.01999614,  0.20932461,  0.0254652 ,  0.26366392,  0.00856102,
        0.08058868, -0.20782008, -0.1359599 , -0.23500545,  0.03740927,
        0.11888506,  0.17102422,  0.21671027,  0.09068703, -0.25568805,
        0.02423541,  0.11943042, -0.51753351, -0.16202727,  0.22873774,
        0.06798634,  0.22152784,  0.0840674 , -0.04993204, -0.26843767,
        0.07260146, -0.30092282,  0.15355953, -0.0522157 , -0.10862874,
        0.04560524, -0.16612259, -0.06226176, -0.11781129, -0.07521048,
        0.00399866,  0.31480429,  0.12684158,  0.0491577 ,  0.03760581,
       -0.12408901,  0.16498946,  0.0691526 ,  0.41420899, -0.31107269,
       -0.19319433, -0.04998738, -0.13795248,  0.01528367, -0.06783712,
       -0.02698278, -0.30714476, -0.04999141,  0.15458899, -0.03083627,
        0.34201905,  0.23672756,  0.31692924,  0.0461333 , -0.08079044,
       -0.08898347,  0.17961326, -0.38170894, -0.23787375, -0.10015603,
        0.22698843, -0.15538733,  0.39900836,  0.1796477 , -0.05579911,
       -0.1960262 ,  0.07388732, -0.18282236,  0.13711534, -0.09072778,
        0.05247565,  0.38945165,  0.25997175, -0.39301072, -0.19569835,
       -0.176655  , -0.01782703,  0.07201633,  0.11289259,  0.04637413,
       -0.17495102, -0.09149356,  0.12997283,  0.04181852,  0.00232262])
```

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['word2vec'])
```

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(64, activation=keras.layers.LeakyReLU(alpha=0.1)),
    keras.layers.Dense(32, 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, 64)                2112

 dense_5 (Dense)             (None, 32)                2080

 dense_6 (Dense)             (None, 2)                 66

=================================================================
Total params: 86146 (336.51 KB)
Trainable params: 86146 (336.51 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 [==============================] - 11s 4ms/step - loss: 0.3047 -
accuracy: 0.8424
Epoch 2/22
2036/2036 [==============================] - 8s 4ms/step - loss: 0.2721 -
accuracy: 0.8564
Epoch 3/22
2036/2036 [==============================] - 8s 4ms/step - loss: 0.2700 -
accuracy: 0.8574
Epoch 4/22
2036/2036 [==============================] - 9s 4ms/step - loss: 0.2671 -
accuracy: 0.8582
Epoch 5/22
2036/2036 [==============================] - 12s 6ms/step - loss: 0.2651 -
accuracy: 0.8593
Epoch 6/22
2036/2036 [==============================] - 15s 7ms/step - loss: 0.2639 -
accuracy: 0.8597
Epoch 7/22
2036/2036 [==============================] - 19s 9ms/step - loss: 0.2620 -
accuracy: 0.8601
Epoch 8/22
2036/2036 [==============================] - 18s 9ms/step - loss: 0.2621 -
accuracy: 0.8604
Epoch 9/22
2036/2036 [==============================] - 18s 9ms/step - loss: 0.2608 -
accuracy: 0.8608
Epoch 10/22
2036/2036 [==============================] - 18s 9ms/step - loss: 0.2606 -
accuracy: 0.8610
Epoch 11/22
2036/2036 [==============================] - 20s 10ms/step - loss: 0.2606
- accuracy: 0.8607
Epoch 12/22
2036/2036 [==============================] - 21s 10ms/step - loss: 0.2595
- accuracy: 0.8612
Epoch 13/22
2036/2036 [==============================] - 21s 11ms/step - loss: 0.2598
- accuracy: 0.8615
Epoch 14/22
2036/2036 [==============================] - 19s 9ms/step - loss: 0.2589 -
accuracy: 0.8612
Epoch 15/22
2036/2036 [==============================] - 20s 10ms/step - loss: 0.2596
- accuracy: 0.8612
Epoch 16/22
2036/2036 [==============================] - 18s 9ms/step - loss: 0.2586 -
accuracy: 0.8617
Epoch 17/22
2036/2036 [==============================] - 18s 9ms/step - loss: 0.2586 -
accuracy: 0.8619
```

```
Epoch 18/22
2036/2036 [==============================] - 18s 9ms/step - loss: 0.2582 -
accuracy: 0.8618
Epoch 19/22
2036/2036 [==============================] - 17s 8ms/step - loss: 0.2588 -
accuracy: 0.8619
Epoch 20/22
2036/2036 [==============================] - 16s 8ms/step - loss: 0.2575 -
accuracy: 0.8623
Epoch 21/22
2036/2036 [==============================] - 18s 9ms/step - loss: 0.2584 -
accuracy: 0.8618
Epoch 22/22
2036/2036 [==============================] - 21s 10ms/step - loss: 0.2578
- accuracy: 0.8624
```

Out[30]: <keras.src.callbacks.History at 0x26a10157ca0>

Training Accuracy

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

```
2036/2036 [==============================] - 18s 8ms/step - loss: 0.2587 -
accuracy: 0.8622
```

Out[31]: [0.25865301489830017, 0.86215740442276]

Testing Accuracy

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

```
509/509 [==============================] - 4s 7ms/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       1.00      0.79      0.88      8385
         1.0       0.82      1.00      0.90      7898

    accuracy                           0.89     16283
   macro avg       0.91      0.89      0.89     16283
weighted avg       0.91      0.89      0.89     16283


Confusion Matrix:
 [[6637 1748]
 [  22 7876]]

Accuracy:
 0.8912976724190874
```