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', ...	[-1.19050758e-02 1.16620226e-01 1.97997382e-...
1	#sarcasm for #people who don't understand #diy...	figurative	['sarcasm', 'for', 'people', 'who', 'don', 'un...	[2.94221377e-02 2.08143203e-01 1.72751102e-...
2	@IminworkJeremy @medsingle #DailyMail readers ...	figurative	['iminworkjeremy', 'medsingle', 'dailymail', '...	[-0.1235885 0.05067413 0.3733275 -0.227050...
3	@wilw Why do I get the feeling you like games?...	figurative	['wilw', 'why', 'do', 'get', 'the', 'feeling',...	[-0.02185489 0.3480542 0.624338 -0.582425...
4	-@TeacherArthurG @rweingarten You probably jus...	figurative	['teacherarthurg', 'rweingarten', 'you', 'prob...	[-0.05288212 0.1641525 0.54444376 -0.362433...
...	...	...	...	...
81403	Photo: Image via We Heart It http://t.co/ky8Nf...	sarcasm	['photo', 'image', 'via', 'we', 'heart', 'it',...	[-1.84435886e-01 1.83419669e-01 3.13743446e-...
81404	I never knew..I better put this out to the Uni...	sarcasm	['never', 'knew', 'better', 'put', 'this', 'ou...	[0.01726132 0.19902723 0.38343058 -0.256441...
81405	hey just wanted to say thanks @ puberty for le...	sarcasm	['hey', 'just', 'wanted', 'to', 'say', 'thanks...	[-7.33005936e-05 1.17733040e-02 3.56954102e-...
81406	I'm sure coverage like the Fox News Special “T...	sarcasm	['sure', 'coverage', 'like', 'the', 'fox', 'ne...	[-0.08389465 0.100885 0.36214375 -0.300751...
81407	@skeyno16 at u13?! I won't believe it until I ...	sarcasm	['skeyno', 'at', 'won', 'believe', 'it', 'unti...	[-0.02437943 0.19550255 0.49198555 -0.12781 ...

81408 rows × 4 columns

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

Out[4]:
```
0        [-1.19050758e-02  1.16620226e-01  1.97997382e-...
1        [ 2.94221377e-02  2.08143203e-01  1.72751102e-...
2        [-0.1235885   0.05067413  0.3733275  -0.227050...
3        [-0.02185489  0.3480542   0.624338   -0.582425...
4        [-0.05288212  0.1641525   0.54444376 -0.362433...
                               ...                        
81403    [-1.84435886e-01  1.83419669e-01  3.13743446e-...
81404    [ 0.01726132  0.19902723  0.38343058 -0.256441...
81405    [-7.33005936e-05  1.17733040e-02  3.56954102e-...
81406    [-0.08389465  0.100885    0.36214375 -0.300751...
81407    [-0.02437943  0.19550255  0.49198555 -0.12781 ...
Name: word2vec, Length: 81408, dtype: object
```

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

Out[5]:
```
'[-1.19050758e-02  1.16620226e-01  1.97997382e-01 -2.69489541e-01\n -7.667
56877e-02  1.27769003e-01 -2.55769373e-01  9.86336611e-02\n  8.49334672e-0
2 -6.09500741e-02  1.84933924e-01  1.45254004e-01\n -1.05722736e-03  4.072
91507e-02  4.14199961e-02 -1.04670005e-01\n  2.44177999e-01 -1.45667303e-0
1 -1.35545416e-01  2.11801241e-01\n  2.27879851e-01  1.34966927e-02  1.134
93863e-02 -8.83673140e-02\n  3.29575739e-02  1.20378999e-01 -1.00660015e-0
2 -1.57044374e-01\n  8.74430709e-02 -1.54583151e-01 -2.78526134e-02  3.281
45041e-01\n -2.69770028e-02  7.30823353e-03  2.09195139e-01  2.35272463e-0
1\n -8.58561351e-03 -3.60043399e-02  1.22844919e-01  2.76703938e-02\n -1.3
1999616e-01 -1.79826155e-01  4.65860034e-02 -3.88554156e-01\n -2.14142157e
-01  1.52666937e-02 -1.65338843e-01 -1.78316311e-01\n  2.49726155e-02 -6.3
0770764e-01  4.04885411e-02 -6.33034941e-03\n  1.35558544e-01  6.40117851e
-01 -1.91852385e-01 -1.19760076e+00\n  8.80471665e-02 -3.38173759e-02  7.9
6421535e-01 -2.04303846e-02\n -2.09942308e-01  3.04531188e-01 -3.64052308e
-01 -1.81030396e-01\n  5.83005078e-01 -1.24062463e-01  3.68763313e-01  3.2
7876763e-01\n  2.62113096e-02 -2.07590234e-01  6.34577597e-02 -2.89481154e
-01\n -1.36048535e-01 -2.46281767e-01 -7.24130926e-03 -2.36953685e-02\n  
5.65499784e-03 -4.31853690e-03 -3.92700460e-01 -8.04448489e-02\n  3.759901
49e-01 -6.47767699e-02 -1.60761079e-01  1.15441769e-01\n -9.60930782e-01 -
3.17082280e-02  8.96731489e-02  1.83563079e-01\n -3.53253998e-01 -4.028528
47e-01 -1.43927311e-01  1.59790049e-02\n  4.14393749e-02 -1.46675996e-01  
1.74810769e-01 -1.84741768e-01\n  2.63673067e-02 -1.14777002e-01  1.203179
25e-01  2.44135694e-01\n -1.19050758e-02  1.16620226e-01  1.97997382e-01 -
2.69489541e-01\n -7.66756877e-02  1.27769003e-01 -2.55769373e-01  9.863366
11e-02\n  8.49334672e-02 -6.09500741e-02  1.84933924e-01  1.45254004e-01\n
-1.05722736e-03  4.07291507e-02  4.14199961e-02 -1.04670005e-01\n  2.44177
999e-01 -1.45667303e-01 -1.35545416e-01  2.11801241e-01\n  2.27879851e-01  
1.34966927e-02  1.13493863e-02 -8.83673140e-02\n  3.29575739e-02  1.203789
99e-01 -1.00660015e-02 -1.57044374e-01\n  8.74430709e-02 -1.54583151e-01 -
2.78526134e-02  3.28145041e-01\n -2.69770028e-02  7.30823353e-03  2.091951
39e-01  2.35272463e-01\n -8.58561351e-03 -3.60043399e-02  1.22844919e-01  
2.76703938e-02\n -1.31999616e-01 -1.79826155e-01  4.65860034e-02 -3.885541
56e-01\n -2.14142157e-01  1.52666937e-02 -1.65338843e-01 -1.78316311e-01\n
2.49726155e-02 -6.30770764e-01]'
```

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    [-0.0119050758, 0.116620226, 0.197997382, -0.2...
     1   [0.0294221377, 0.208143203, 0.172751102, -0.41...
     2    [-0.1235885, 0.05067413, 0.3733275, -0.2270506...
     3      [-0.02185489, 0.3480542, 0.624338, -0.582425, ...
     4    [-0.05288212, 0.1641525, 0.54444376, -0.362433...
   ...                                                  ...
 81403    [-0.184435886, 0.183419669, 0.313743446, -0.29...
 81404   [0.01726132, 0.19902723, 0.38343058, -0.256441...
 81405    [-7.33005936e-05, 0.011773304, 0.356954102, -0...
 81406    [-0.08389465, 0.100885, 0.36214375, -0.3007513...
 81407    [-0.02437943, 0.19550255, 0.49198555, -0.12781...

81408 rows × 1 columns
```

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

```
81408
```

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

Out[13]:
```
array([-1.19050758e-02,  1.16620226e-01,  1.97997382e-01, -2.69489541e-01,
       -7.66756877e-02,  1.27769003e-01, -2.55769373e-01,  9.86336611e-02,
        8.49334672e-02, -6.09500741e-02,  1.84933924e-01,  1.45254004e-01,
       -1.05722736e-03,  4.07291507e-02,  4.14199961e-02, -1.04670005e-01,
        2.44177999e-01, -1.45667303e-01, -1.35545416e-01,  2.11801241e-01,
        2.27879851e-01,  1.34966927e-02,  1.13493863e-02, -8.83673140e-02,
        3.29575739e-02,  1.20378999e-01, -1.00660015e-02, -1.57044374e-01,
        8.74430709e-02, -1.54583151e-01, -2.78526134e-02,  3.28145041e-01,
       -2.69770028e-02,  7.30823353e-03,  2.09195139e-01,  2.35272463e-01,
       -8.58561351e-03, -3.60043399e-02,  1.22844919e-01,  2.76703938e-02,
       -1.31999616e-01, -1.79826155e-01,  4.65860034e-02, -3.88554156e-01,
       -2.14142157e-01,  1.52666937e-02, -1.65338843e-01, -1.78316311e-01,
        2.49726155e-02, -6.30770764e-01,  4.04885411e-02, -6.33034941e-03,
        1.35558544e-01,  6.40117851e-01, -1.91852385e-01, -1.19760076e+00,
        8.80471665e-02, -3.38173759e-02,  7.96421535e-01, -2.04303846e-02,
       -2.09942308e-01,  3.04531188e-01, -3.64052308e-01, -1.81030396e-01,
        5.83005078e-01, -1.24062463e-01,  3.68763313e-01,  3.27876763e-01,
        2.62113096e-02, -2.07590234e-01,  6.34577597e-02, -2.89481154e-01,
       -1.36048535e-01, -2.46281767e-01, -7.24130926e-03, -2.36953685e-02,
        5.65499784e-03, -4.31853690e-03, -3.92700460e-01, -8.04448489e-02,
        3.75990149e-01, -6.47767699e-02, -1.60761079e-01,  1.15441769e-01,
       -9.60930782e-01, -3.17082280e-02,  8.96731489e-02,  1.83563079e-01,
       -3.53253998e-01, -4.02852847e-01, -1.43927311e-01,  1.59790049e-02,
        4.14393749e-02, -1.46675996e-01,  1.74810769e-01, -1.84741768e-01,
        2.63673067e-02, -1.14777002e-01,  1.20317925e-01,  2.44135694e-01,
       -1.19050758e-02,  1.16620226e-01,  1.97997382e-01, -2.69489541e-01,
       -7.66756877e-02,  1.27769003e-01, -2.55769373e-01,  9.86336611e-02,
        8.49334672e-02, -6.09500741e-02,  1.84933924e-01,  1.45254004e-01,
       -1.05722736e-03,  4.07291507e-02,  4.14199961e-02, -1.04670005e-01,
        2.44177999e-01, -1.45667303e-01, -1.35545416e-01,  2.11801241e-01,
        2.27879851e-01,  1.34966927e-02,  1.13493863e-02, -8.83673140e-02,
        3.29575739e-02,  1.20378999e-01, -1.00660015e-02, -1.57044374e-01,
        8.74430709e-02, -1.54583151e-01, -2.78526134e-02,  3.28145041e-01,
       -2.69770028e-02,  7.30823353e-03,  2.09195139e-01,  2.35272463e-01,
       -8.58561351e-03, -3.60043399e-02,  1.22844919e-01,  2.76703938e-02,
       -1.31999616e-01, -1.79826155e-01,  4.65860034e-02, -3.88554156e-01,
       -2.14142157e-01,  1.52666937e-02, -1.65338843e-01, -1.78316311e-01,
        2.49726155e-02, -6.30770764e-01])
```

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(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: 0.4297 -
accuracy: 0.7776
Epoch 2/22
2036/2036 [==============================] - 6s 3ms/step - loss: 0.3914 -
accuracy: 0.7971
Epoch 3/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3798 -
accuracy: 0.8028
Epoch 4/22
2036/2036 [==============================] - 6s 3ms/step - loss: 0.3720 -
accuracy: 0.8065
Epoch 5/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3656 -
accuracy: 0.8086
Epoch 6/22
2036/2036 [==============================] - 8s 4ms/step - loss: 0.3604 -
accuracy: 0.8100
Epoch 7/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3548 -
accuracy: 0.8144
Epoch 8/22
2036/2036 [==============================] - 8s 4ms/step - loss: 0.3515 -
accuracy: 0.8161
Epoch 9/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3474 -
accuracy: 0.8168
Epoch 10/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3418 -
accuracy: 0.8205
Epoch 11/22
2036/2036 [==============================] - 8s 4ms/step - loss: 0.3394 -
accuracy: 0.8204
Epoch 12/22
2036/2036 [==============================] - 8s 4ms/step - loss: 0.3360 -
accuracy: 0.8218
Epoch 13/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3330 -
accuracy: 0.8238
Epoch 14/22
2036/2036 [==============================] - 6s 3ms/step - loss: 0.3288 -
accuracy: 0.8249
Epoch 15/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3270 -
accuracy: 0.8267
Epoch 16/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3246 -
accuracy: 0.8276
Epoch 17/22
2036/2036 [==============================] - 6s 3ms/step - loss: 0.3209 -
accuracy: 0.8284
```

```
Epoch 18/22
2036/2036 [==============================] - 6s 3ms/step - loss: 0.3189 -
accuracy: 0.8294
Epoch 19/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3160 -
accuracy: 0.8310
Epoch 20/22
2036/2036 [==============================] - 6s 3ms/step - loss: 0.3138 -
accuracy: 0.8318
Epoch 21/22
2036/2036 [==============================] - 7s 4ms/step - loss: 0.3112 -
accuracy: 0.8324
Epoch 22/22
2036/2036 [==============================] - 7s 3ms/step - loss: 0.3084 -
accuracy: 0.8348
```

Out[30]:

```
<keras.src.callbacks.History at 0x1aff962e1d0>
```

Training Accuracy

In [31]:

```
awe.evaluate(x_train.astype(np.float32), y_train.astype(np.float32))
```

```
2036/2036 [==============================] - 5s 2ms/step - loss: 0.3047 -
accuracy: 0.8377
```

Out[31]:

```
[0.30468904972076416, 0.837727427482605]
```

Testing Accuracy

In [32]:

```
prediction = awe.predict(x_test.astype(np.float32))
```

```
509/509 [==============================] - 2s 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       0.90      0.76      0.83      8385
         1.0       0.78      0.91      0.84      7898

    accuracy                           0.84     16283
   macro avg       0.84      0.84      0.83     16283
weighted avg       0.84      0.84      0.83     16283


Confusion Matrix: 
 [[6407 1978]
 [ 703 7195]]

Accuracy: 
 0.8353497512743352
```