# nb11a_airline_tweets_MLP_w_embeddings

November 1, 2023

## 1 CS 39AA - Notebook 11a: Airline Tweets MLP w/ Word Embeddings

We've started talking about word embeddings; what they are as well as how we both create and use them. We'll now see what it looks like to load and use pre-made embeddings on the Airline Tweet dataset.

The word embeddings we'll use are going to be the GloVe embeddings. There are several sets of GloVe embeddings that were created. The smallest has a vocabulary of 400k tokens/words with each embedding represented by a 50-element vector (50d). The largest has a vocabulary of over 2M tokens/words with each embedding being a 300-element vector (300d). On top of that, the embeddings were created with different datasets. So, depending on your use case you may want to use embeddings with a different vocabulary size, different embedding size, and/or created using a different dataset. You can read more about the GloVe embedding here: * https://nlp.stanford.edu/projects/glove/

We'll use the smallest set of embeddings and with the smallest vector sizes. This set of embeddings is downloaded in a zip file with the 100d, 200d, and 300d sets of embeddings, which altogether is over 800Mb in size. So, for practical reasons, you'll probably want to run this locally where you have more storage.

[178]:
```
import torch
import random
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchmetrics.functional import pairwise_cosine_similarity
from tqdm import notebook
```

[179]:
```
import torchtext as text
vec = text.vocab.GloVe(name='6B', dim=50)
```

Load the airline tweets and prepare to load embeddings for each tweet.

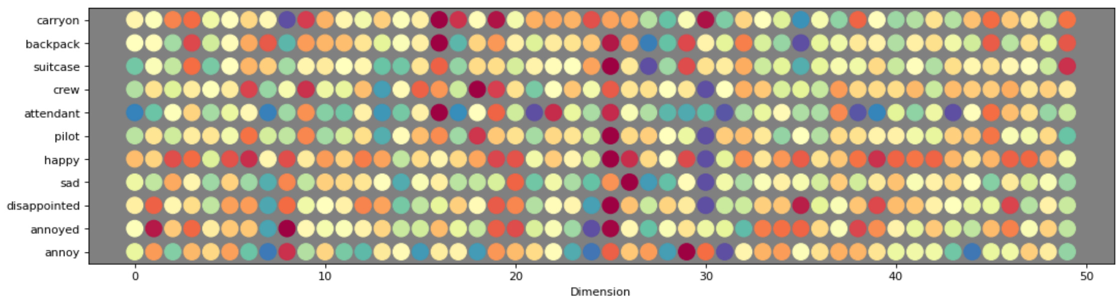
```
[180]: examples = ['annoy', 'annoyed', 'disappointed', 'sad', 'happy', 'pilot',␣
        ↪'attendant', 'crew', 'suitcase', 'backpack', 'carryon']
       embeddings = vec.get_vecs_by_tokens(examples, lower_case_backup=True)
       embeddings
```

```
[180]: tensor([[ 2.3457e-01, -4.7683e-01,  6.3459e-01, -3.6475e-01, -2.1981e-01,
                -4.5539e-01,  7.0779e-01,  1.0140e+00, -9.1157e-01,  4.6997e-01,
                -2.2969e-01,  6.6490e-01,  7.6746e-01, -1.0760e-01,  3.6551e-03,
                 8.9326e-01, -9.5184e-02, -5.2423e-02,  8.8386e-01, -4.9152e-01,
                -3.5541e-01, -1.9993e-01,  1.9838e-02,  8.1985e-01,  1.0315e+00,
                -6.9975e-01, -2.9327e-01, -4.7181e-01,  8.7062e-01, -1.0971e+00,
                -6.3421e-01,  1.1884e+00, -3.3743e-02, -3.8841e-01, -4.6839e-01,
                 8.8104e-02, -1.6746e-01, -4.0789e-01, -3.7836e-01, -1.4252e-01,
                 2.0980e-01,  1.7340e-01,  1.9545e-01,  6.8907e-01,  1.0228e+00,
                 2.3531e-01,  1.9601e-01, -2.6078e-01, -8.8978e-02,  5.7906e-01],
               [ 1.9271e-01, -9.8134e-01, -1.9098e-01, -6.1861e-01,  2.4468e-02,
                -2.0539e-01, -2.0984e-01,  1.0406e+00, -1.0876e+00,  1.1150e-01,
                 2.6504e-01,  2.1416e-01,  5.5952e-02, -1.3927e-01,  5.4169e-01,
                 5.0256e-01, -2.6933e-01,  3.7559e-01,  5.4813e-01, -6.5889e-01,
                -7.6380e-01,  4.5609e-01,  2.2816e-01,  7.0066e-01,  1.4124e+00,
                -1.0646e+00,  1.0088e-01,  9.1145e-01,  2.2551e-01, -5.0154e-02,
                 3.3136e-01,  3.9667e-01,  9.3441e-01, -5.2839e-01, -5.7883e-01,
                -6.3778e-01, -1.1350e-01,  2.2020e-01, -7.8774e-01, -4.5987e-01,
                 7.3540e-02,  3.8382e-01, -1.9287e-01,  4.3775e-01,  5.8059e-01,
                -2.6786e-01, -5.0261e-01,  2.2724e-01, -1.5704e-01,  5.7799e-01],
               [-2.2685e-02, -8.5923e-01,  4.9477e-02, -1.7606e-01,  6.4941e-01,
                -5.4191e-01, -5.5731e-01,  1.2037e+00, -7.9421e-01,  3.2672e-01,
                 1.3117e-01, -6.6106e-03, -6.7498e-01, -5.3080e-01,  9.8269e-01,
                 6.5216e-01,  4.1866e-01, -2.6490e-01,  8.6192e-02, -8.9927e-01,
                -6.6251e-01,  7.0842e-01,  1.0279e-01,  3.7234e-02,  1.2416e+00,
                -1.3999e+00, -3.4743e-01,  5.8947e-01, -2.4608e-01, -5.8111e-02,
                 1.6744e+00,  5.8394e-01,  5.6990e-01, -2.6174e-01, -3.3512e-01,
                -1.2532e+00,  4.4094e-01,  2.2765e-01, -3.4198e-01, -1.0227e+00,
                -3.9604e-01, -3.1511e-01,  3.5769e-03,  1.5152e-01,  3.8344e-01,
                 3.9369e-01, -1.0501e+00,  6.9915e-01, -7.8364e-04,  6.0356e-01],
               [ 1.8822e-01,  5.2772e-01, -8.0729e-01, -1.8974e-01,  7.3361e-01,
                -5.2599e-01,  7.3379e-01,  1.1510e+00, -1.0057e+00,  5.3222e-01,
                -5.3503e-01, -4.6232e-01, -3.5761e-01, -8.9558e-02,  1.1745e+00,
                 2.5105e-02, -2.6076e-01,  5.7176e-01,  5.1661e-01,  3.9261e-01,
                -1.2262e+00,  9.6739e-01,  1.4591e-01,  6.7439e-01,  1.0324e+00,
                -9.3460e-01, -1.8862e+00,  1.2702e+00,  1.0383e+00, -9.3612e-02,
                 1.7631e+00,  1.3482e-01,  6.5860e-01,  1.7446e-02, -2.3751e-01,
                 8.0928e-02,  4.0966e-01, -5.6527e-01,  4.3035e-01, -3.0735e-01,
                -6.3660e-01,  4.2546e-02, -2.3112e-01, -4.6408e-01, -4.1270e-02,
                 8.6248e-01, -3.1139e-01,  3.7836e-01,  3.7122e-02,  7.4944e-01],
```

```
[ 9.2086e-02,  2.5710e-01, -5.8693e-01, -3.7029e-01,  1.0828e+00,
 -5.5466e-01, -7.8142e-01,  5.8696e-01, -5.8714e-01,  4.6318e-01,
 -1.1267e-01,  2.6060e-01, -2.6928e-01, -7.2466e-02,  1.2470e+00,
  3.0571e-01,  5.6731e-01,  3.0509e-01, -5.0312e-02, -6.4443e-01,
 -5.4513e-01,  8.6429e-01,  2.0914e-01,  5.6334e-01,  1.1228e+00,
 -1.0516e+00, -7.8105e-01,  2.9656e-01,  7.2610e-01, -6.1392e-01,
  2.4225e+00,  1.0142e+00, -1.7753e-01,  4.1470e-01, -1.2966e-01,
 -4.7064e-01,  3.8070e-01,  1.6309e-01, -3.2300e-01, -7.7899e-01,
 -4.2473e-01, -3.0826e-01, -4.2242e-01,  5.5069e-02,  3.8267e-01,
  3.7415e-02, -4.3020e-01, -3.9442e-01,  1.0511e-01,  8.7286e-01],
[ 7.1591e-01, -1.8230e-01,  5.9495e-01, -1.0324e-01, -1.7107e-01,
  2.8205e-01, -8.9865e-01,  5.7488e-01,  6.9234e-01, -7.9392e-01,
  8.3707e-01,  5.6658e-01, -2.3263e-01,  1.2802e+00,  7.8758e-02,
 -4.7402e-01, -7.6294e-01,  8.7332e-01, -1.3297e+00, -4.0187e-01,
 -2.0272e-01,  8.2063e-01, -2.9414e-01, -1.7837e-01,  5.6437e-01,
 -1.6316e+00, -2.7048e-01, -3.5347e-01,  1.1791e-01,  3.6560e-01,
  1.8555e+00, -3.7432e-01, -4.6504e-01, -1.3966e-01,  9.2002e-01,
  1.3795e-01,  7.6497e-01, -2.6930e-01, -2.1149e-01, -3.2370e-02,
 -4.3753e-02,  2.9203e-01,  4.2256e-01, -3.5090e-01, -4.1912e-01,
 -8.8538e-01,  1.9450e-01,  2.8401e-01, -2.7501e-01,  1.1425e+00],
[ 7.3908e-01,  4.9413e-01, -7.8118e-02, -3.5652e-01,  3.1676e-01,
  5.7715e-02, -1.4639e-01,  7.4961e-01,  3.4401e-01, -6.0416e-01,
  4.1808e-01,  4.8163e-01, -1.9152e-01,  6.6207e-01,  4.6404e-01,
 -1.9075e-01, -1.1314e+00,  6.9946e-01, -1.0374e-01, -7.5703e-01,
  1.8034e-01,  9.2689e-01, -9.8131e-01,  7.3790e-02,  3.1000e-01,
 -9.9073e-01, -4.4475e-02,  2.8798e-01,  5.5420e-01,  6.0115e-01,
  5.2614e-01,  8.9486e-01,  3.0385e-01,  7.0074e-02, -1.3731e-01,
  2.9898e-01,  3.0154e-01, -6.3233e-01,  8.9930e-01,  7.3238e-01,
  1.1348e-01,  3.9349e-01,  4.2235e-02,  9.2092e-01, -1.1699e-01,
 -7.1762e-01, -4.4671e-01, -2.9404e-01,  3.6754e-01,  1.9790e-01],
[ 9.6326e-01, -3.1685e-02,  6.7622e-01,  6.5282e-02,  3.4352e-01,
  7.0106e-02, -1.1083e+00,  1.1146e+00,  4.2360e-01, -1.2208e+00,
  5.3304e-01,  6.0783e-01, -3.1562e-01,  1.5906e+00,  2.2113e-01,
 -8.7097e-01, -5.6742e-01,  8.0841e-01, -1.5170e+00, -1.1391e+00,
 -4.6099e-02,  1.2752e+00,  3.1272e-01, -2.6758e-01,  4.6354e-01,
 -9.2472e-01, -4.3961e-03, -4.9565e-02,  3.0746e-01,  1.8146e-02,
  2.0982e+00,  2.9759e-01, -4.1905e-01, -2.4324e-01,  7.1959e-01,
  6.4791e-01,  8.4272e-01, -5.3182e-01, -1.2122e-01, -9.4960e-02,
 -8.5583e-02, -3.5631e-01,  8.2021e-01, -3.4058e-01, -2.1215e-01,
 -2.1247e-01, -4.3997e-01,  2.0940e-02, -1.8163e-01,  5.3004e-02],
[ 1.0930e+00,  2.5557e-01,  7.0036e-01, -7.9883e-01,  1.0741e+00,
  1.7331e-01, -4.2864e-01, -2.6062e-01,  8.6608e-01,  8.0682e-02,
 -2.8177e-02,  1.9031e-01,  8.3725e-02,  1.1534e+00,  1.1114e+00,
 -8.1118e-02, -8.6911e-01,  9.4254e-01, -1.7280e-01, -1.5783e-01,
  6.0530e-01,  4.4777e-02,  1.4699e-01,  2.0482e-01, -5.2930e-01,
 -1.4538e+00, -4.7470e-02,  1.8224e+00,  8.9063e-01, -1.0290e+00,
 -1.1012e-01, -8.1422e-03, -4.7567e-01,  6.5253e-01,  5.2025e-01,
```

```
         1.2790e+00,  4.8318e-01,  3.1067e-01,  3.7941e-01, -1.8365e-01,
         6.7126e-01,  1.6682e-01,  8.0067e-01, -1.4790e-01,  9.1602e-02,
        -1.2224e-01,  1.0385e-01,  1.6026e-01,  6.5254e-01, -1.1915e+00],
       [ 2.2638e-01,  1.9214e-01,  8.0589e-01, -8.2320e-01,  5.9969e-01,
         1.9381e-01, -3.5036e-01, -7.3503e-01,  1.1045e+00, -4.0402e-01,
        -2.9673e-01, -2.4630e-01, -3.7019e-03,  4.8380e-01,  2.8404e-01,
         8.2858e-02, -1.1703e+00,  1.1068e+00, -1.1664e-01, -4.2270e-01,
         9.2149e-02,  5.2830e-01,  4.0882e-02, -2.4695e-02, -1.8675e-01,
        -1.0715e+00, -3.3686e-01,  1.3798e+00,  1.0543e+00, -8.2971e-01,
         1.1473e-01,  4.8553e-01, -5.3291e-01,  5.6156e-01,  8.5246e-01,
         1.6123e+00,  4.7189e-01,  3.8906e-01,  1.5511e-02,  1.1167e-01,
         7.5259e-01,  8.3641e-02, -4.6846e-02,  3.7728e-01,  5.9144e-01,
        -7.0772e-01,  7.0361e-01, -3.6893e-02,  5.0924e-01, -7.4390e-01],
       [ 3.0507e-01,  4.7664e-01, -5.0303e-01, -6.4171e-01,  6.5123e-01,
         4.0926e-01,  1.5190e-02,  3.4744e-01,  2.1738e+00, -9.7942e-01,
        -2.4675e-01,  3.2346e-01,  5.8963e-01, -3.4128e-02,  4.8194e-01,
         2.5780e-01, -1.3294e+00, -1.0405e+00,  3.7715e-01, -1.2140e+00,
         5.7346e-01, -2.7389e-01, -3.0542e-01, -2.3720e-01, -8.4401e-01,
        -3.4359e-01, -2.8209e-01,  1.0537e+00,  1.4733e+00,  5.1045e-01,
        -1.2315e+00,  1.3436e+00,  7.4767e-03,  4.4080e-01,  8.1744e-01,
         1.7498e+00,  5.5063e-01,  1.2640e+00, -6.9242e-01,  3.9697e-01,
         1.1987e+00,  1.0660e+00,  1.1286e-01,  1.0536e+00, -1.4628e-01,
        -6.3429e-01, -7.2118e-02,  1.7852e-01,  9.3406e-01, -5.9165e-01]])
```

[181]:
```
def compare_words_with_colors(vecs, wds):
    wdsr = wds[:]
    wdsr.reverse()

    dim = len(vecs[0])

    fig = plt.figure(num=None, figsize=(16, 4), dpi=80, facecolor='w',␣
 ↪edgecolor='k')
    ax = fig.add_subplot(111)
    ax.set_facecolor('gray')

    for i,v in enumerate(vecs):
        ax.scatter(range(dim),[i]*dim, c=vecs[i], cmap='Spectral', s=200)

    #plt.xticks(range(n), [i+1 for i in range(n)])
    plt.xlabel('Dimension')
    plt.yticks(range(len(wds)), wds)

    plt.show()

compare_words_with_colors(embeddings, examples)
#examples.reverse()
```

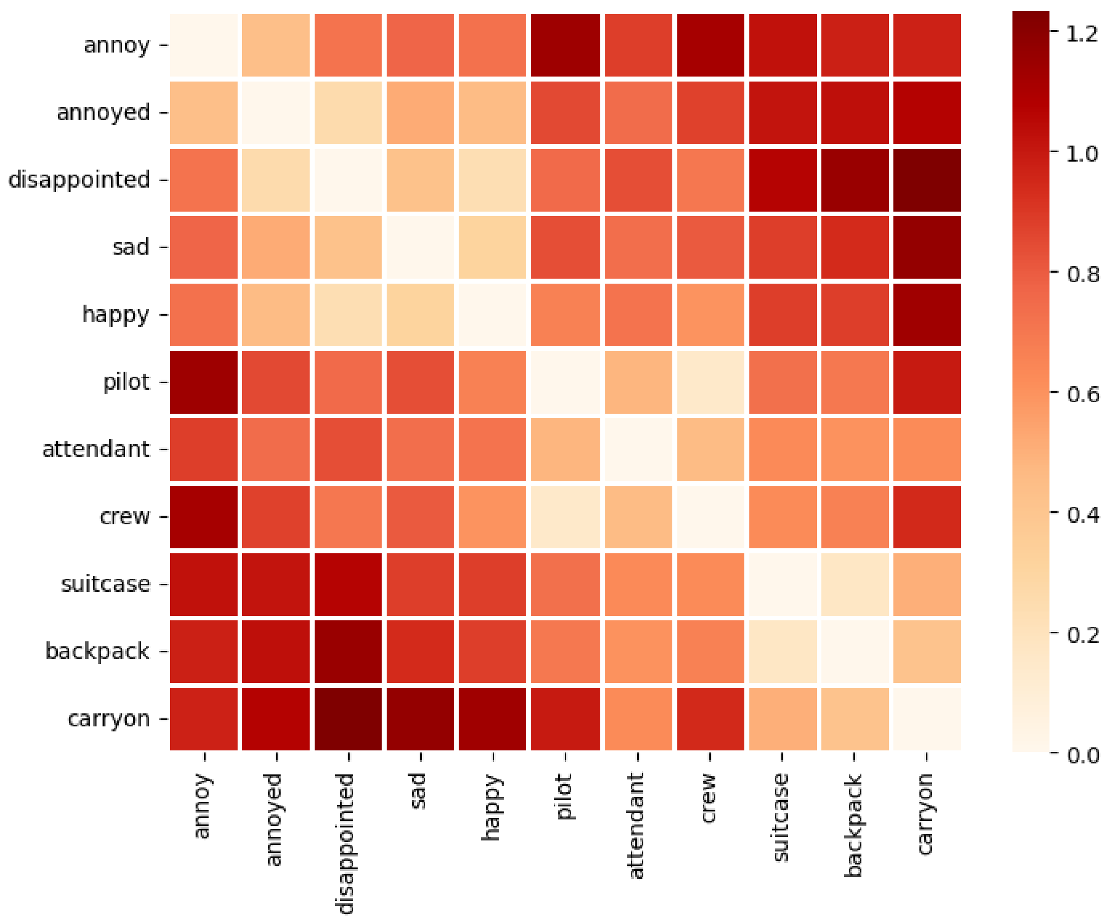



```
[182]: similarities = pairwise_cosine_similarity(embeddings, zero_diagonal=False)
distances = 1 - similarities
distances

pairwise_top = pd.DataFrame(
    distances,
    columns = examples,
    index = examples
)

distances
#similarities
```

```
[182]: tensor([[1.1921e-07, 4.3455e-01, 7.1622e-01, 7.6653e-01, 7.2499e-01, 1.1374e+00,
          8.8526e-01, 1.1141e+00, 1.0229e+00, 9.7518e-01, 9.6831e-01],
         [4.3455e-01, 5.9605e-08, 2.5124e-01, 5.1642e-01, 4.5671e-01, 8.4883e-01,
          7.4169e-01, 8.7239e-01, 1.0158e+00, 1.0266e+00, 1.0706e+00],
         [7.1622e-01, 2.5124e-01, 0.0000e+00, 4.2236e-01, 2.3252e-01, 7.4722e-01,
          8.3502e-01, 6.9937e-01, 1.0674e+00, 1.1509e+00, 1.2325e+00],
         [7.6653e-01, 5.1642e-01, 4.2236e-01, 0.0000e+00, 3.1094e-01, 8.3679e-01,
          7.3417e-01, 8.0400e-01, 8.8330e-01, 9.4015e-01, 1.1652e+00],
         [7.2499e-01, 4.5671e-01, 2.3252e-01, 3.1094e-01, 5.9605e-08, 6.6121e-01,
          7.1570e-01, 5.9392e-01, 8.8221e-01, 8.8511e-01, 1.1316e+00],
         [1.1374e+00, 8.4883e-01, 7.4722e-01, 8.3679e-01, 6.6121e-01, 0.0000e+00,
          4.7873e-01, 1.4877e-01, 7.2720e-01, 6.9736e-01, 9.9505e-01],
         [8.8526e-01, 7.4169e-01, 8.3502e-01, 7.3417e-01, 7.1570e-01, 4.7873e-01,
          0.0000e+00, 4.5259e-01, 6.2653e-01, 5.9906e-01, 6.2299e-01],
         [1.1141e+00, 8.7239e-01, 6.9937e-01, 8.0400e-01, 5.9392e-01, 1.4877e-01,
          4.5259e-01, 0.0000e+00, 6.2210e-01, 6.6070e-01, 9.4738e-01],
         [1.0229e+00, 1.0158e+00, 1.0674e+00, 8.8330e-01, 8.8221e-01, 7.2720e-01,
          6.2653e-01, 6.2210e-01, 0.0000e+00, 1.6754e-01, 5.0505e-01],
         [9.7518e-01, 1.0266e+00, 1.1509e+00, 9.4015e-01, 8.8511e-01, 6.9736e-01,
          5.9906e-01, 6.6070e-01, 1.6754e-01, 5.9605e-08, 4.0982e-01],
         [9.6831e-01, 1.0706e+00, 1.2325e+00, 1.1652e+00, 1.1316e+00, 9.9505e-01,
          6.2299e-01, 9.4738e-01, 5.0505e-01, 4.0982e-01, 0.0000e+00]])
```

[183]:
```
plt.figure(figsize=(8,6))
#sns.color_palette("viridis", as_cmap=True)
sns.color_palette("mako", as_cmap=True)
sns.heatmap(
    pairwise_top,
    #cmap='BuPu', #'OrRd',
    cmap='OrRd',
    linewidth=1
)
```

[183]: <Axes: >




[184]:
```
data_URL = 'https://raw.githubusercontent.com/sgeinitz/CS39AA/main/data/trainA.
↪csv'
df = pd.read_csv(data_URL)
print(f"df.shape: {df.shape}")
pd.set_option("display.max_colwidth", 240)
df.head(10)
```

```
df.shape: (10000, 2)
```

[184]:

```
  sentment  \
0  positive
1  positive
2  negative
3  negative
4  negative
5  negative
6   neutral
7  negative
8  negative
9  positive

                                                                                text
0                                                 @JetBlue @JayVig I like the
inflight snacks! I'm flying with you guys on 2/28! #JVMChat
1                                                                @VirginAmerica
thanks guys! Sweet route over the Rockies #airplanemodewason
2             @USAirways Your exchange/credit policies are worthless and shadier
than the White House. Dissatisfied to the nines right now.
3                                            @USAirways but in the meantime
I'll be sleeping on a park bench on dadeland st.  Thanks guys!
4
@VirginAmerica hold times at call center are a bit much
5                                         @USAirways not moving we are in the
tarmac delayed for some unknown reason. I'll keep you posted
6                                       @JetBlue What about if I booked it
through Orbitz? My email is correct, but there's a middle party.
7                                       @united 2nd flight also delayed no
pilots! But they boarded is so we can just sit here! #scheduling
8  .@AmericanAir after 50 minutes on hold, and another 30 minutes on the call
yes. Going to be pushing it to get to the airport on time now
9
@JetBlue flight 117. proud to fly Jet Blue!
```

Recall that about 2/3 of the data have negative labels, and that the remaining labels are roughly split between positive and neutral (slightly more neutral than positive).

[185]:

```python
random.seed(42)
indices = list(range(len(df)))
random.shuffle(indices)

df_test = df.iloc[indices[9000:],]
df = df.iloc[indices[:9000],]
```

[186]:

```python
df.sentiment.value_counts(normalize=False)
```

[186]:
```
negative    5868
neutral     1726
positive    1406
Name: sentiment, dtype: int64
```

Let's start with the nltk TweetTokenizer, which will split the text into separate words and characters based on common Twitter conventions.

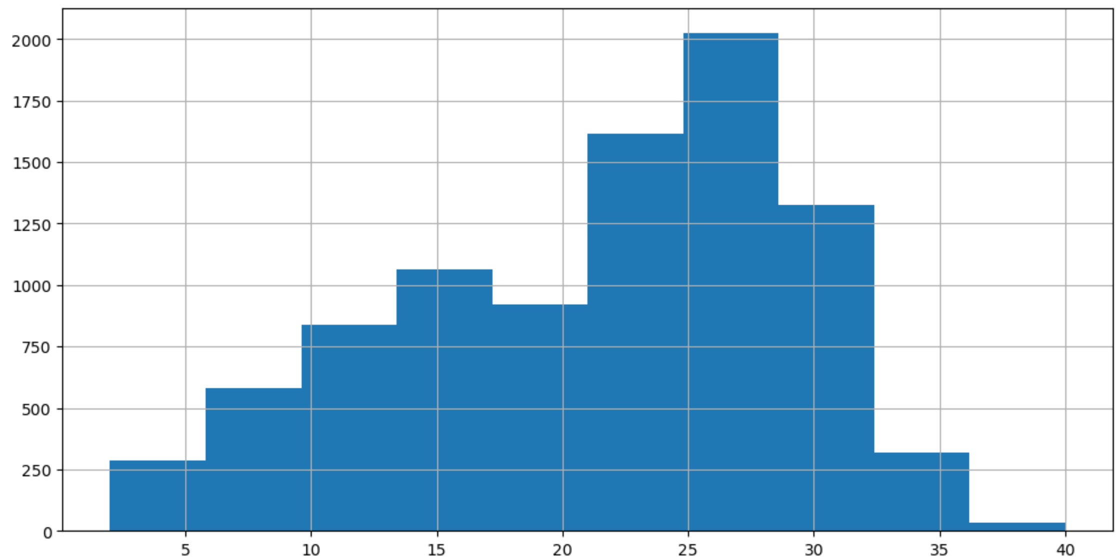
[187]:
```
import torchtext
from torchtext.data import get_tokenizer
tokenizer = get_tokenizer("basic_english") # "basic_english"   "subword" uses␣
↪revtok module (but does not work with GLoVE)
df['tokens_raw'] = df['text'].apply(lambda x: tokenizer(x.lower()))
df.head()
```

[187]:
```
     sentiment  \
3771   neutral
6672  positive
7261  negative
760    neutral
3779   neutral

  text  \
3771
@JetBlue Come on and provide service from Destin- Fort Walton Beach Airport
6672
@JetBlue u the real MVP http://t.co/jWL26G6lRw
7261  @SouthwestAir My brother &amp; his girlfriend's flight Cancelled Flightled
3 times, now leaving 72 hours Late Flight and dropping letter grades at school.
Help?
760                          @AmericanAir More or less - after a night in a
party hotel - no sleep and a 5:30 am rebook- on our way back to PHL
http://t.co/4GOKOz2rei
3779                                                                 @JetBlue marks
15th birthday with 'Blumanity' paint job - @Dallas_News (blog)
http://t.co/lFGR0Nifut

                          tokens_raw
3771
[@jetblue, come, on, and, provide, service, from, destin-, fort, walton, beach,
airport]
6672
[@jetblue, u, the, real, mvp, http, //t, ., co/jwl26g6lrw]
7261  [@southwestair, my, brother, &amp, his, girlfriend's, flight, cancelled,
flightled, 3, times, ,, now, leaving, 72, hours, late, flight, and, dropping,
letter, grades, at, school, ., help, ?]
760                       [@americanair, more, or, less, -, after, a, night, in,
```

```
a, party, hotel, -, no, sleep, and, a, 5, 30, am, rebook-, on, our, way, back,
to, phl, http, //t, ., co/4g0k0z2rei]
3779                                                                  [@jetblue,
marks, 15th, birthday, with, ', blumanity, ', paint, job, -, @dallas_news, (,
blog, ), http, //t, ., co/lfgr0nifut]
```

We can now look at the emebedding for each individual token in a single tweet. Notice that when a token does not exist in GloVE that it is assigned all zeros.

[188]:
```
df['tweet_length'] = df['tokens_raw'].apply(lambda x: len(x))
plt.figure(figsize=(12,6))
df['tweet_length'].hist() #bins=100, range=(0,45), width=0.9) #,␣
 ↪df['tweet_length'].mean(), df['tweet_length'].median()
plt.show()
```




[189]:
```
tweet_i= 65
tweet_embeddings = vec.get_vecs_by_tokens(df['tokens_raw'][tweet_i],␣
 ↪lower_case_backup=True)
print(f"sentiment of this tweet: {df['sentiment'][tweet_i]}")
print(f"tweet_embeddings.shape = {tweet_embeddings.shape}")
for i in range(df['tweet_length'][tweet_i]):
    print(f"    token, '{df['tokens_raw'][tweet_i][i]}' (at pos {i:2.0f}) has␣
 ↪tweet_embeddings[:5] = {tweet_embeddings[i][:5]}")
```

```
sentiment of this tweet: negative
tweet_embeddings.shape = torch.Size([15, 50])
    token, '@usairways' (at pos  0) has tweet_embeddings[:5] = tensor([0., 0.,
0., 0., 0.])
    token, 'no' (at pos  1) has tweet_embeddings[:5] = tensor([ 0.3496,  0.4015,
```

```
-0.0126,  0.1374,  0.4008])
    token, 'my' (at pos  2) has tweet_embeddings[:5] = tensor([-0.2728,  0.7752,
-0.1018, -0.9166,  0.9048])
    token, 'flight' (at pos  3) has tweet_embeddings[:5] = tensor([ 1.7306,
0.2840, -0.0406, -0.0874, -0.4819])
    token, 'plans' (at pos  4) has tweet_embeddings[:5] = tensor([ 1.3427,
-0.0681,  0.5598,  0.2806, -0.5117])
    token, 'have' (at pos  5) has tweet_embeddings[:5] = tensor([ 0.9491,
-0.3497,  0.4812, -0.1931, -0.0088])
    token, 'been' (at pos  6) has tweet_embeddings[:5] = tensor([ 0.9288,
-0.7246,  0.0681, -0.3816, -0.0387])
    token, 'delayed' (at pos  7) has tweet_embeddings[:5] = tensor([ 1.0818,
-0.3236,  0.0523,  0.2775, -1.1769])
    token, 'until' (at pos  8) has tweet_embeddings[:5] = tensor([ 0.2002,
-0.3282, -0.4086, -0.7944, -0.0162])
    token, 'tuesday' (at pos  9) has tweet_embeddings[:5] = tensor([ 0.3475,
-0.0761,  0.2134,  0.8125,  0.2234])
    token, 'due' (at pos 10) has tweet_embeddings[:5] = tensor([ 0.5253,
0.2501, -0.2989, -0.0039, -0.6855])
    token, 'to' (at pos 11) has tweet_embeddings[:5] = tensor([ 0.6805, -0.0393,
0.3019, -0.1779,  0.4296])
    token, 'your' (at pos 12) has tweet_embeddings[:5] = tensor([-0.0292,
0.8177,  0.3847, -0.7786,  1.1049])
    token, 'computer' (at pos 13) has tweet_embeddings[:5] = tensor([ 0.0791,
-0.8150,  1.7901,  0.9165,  0.1080])
    token, 'crash' (at pos 14) has tweet_embeddings[:5] = tensor([ 1.1944,
-0.0823,  0.7072,  0.5260, -0.7019])
```

[190]:
```
tweet_embeddings.shape
```

[190]:
```
torch.Size([15, 50])
```

Before we continue we must decide what a good length will be for a max-length of the number of tokens to keep. Let's look at a histogram of the lenghts of each tweet (where length equals the number of raw tokens).

[191]:
```
def meanTweetEmbeddings(raw_tokens):
    embeddings = vec.get_vecs_by_tokens(raw_tokens, lower_case_backup=True)
    n_embs = 0
    emb_sum = torch.zeros((embeddings.shape[1]))
    for i in range(min(embeddings.shape[0], 35)):
        if embeddings[i].abs().sum() > 0:
            n_embs += 1
            emb_sum += embeddings[i]
    if n_embs > 0:
        emb_avg = emb_sum / n_embs
    else:
        emb_avg = torch.zeros((embeddings.shape[1]))
```

```
    if np.any(np.isnan(emb_avg.numpy())):
        print(f"exists an nan: {emb_sum}")
    return emb_avg

#X_int = df['tokens_raw'].apply(lambda x: meanTweetEmbeddings(x)).values
#X_int[:5]
```

[192]:

```
# an alternative to the above is to stack the first k tokens in a tweet␣
 ↪together so that we keep the
# original embeddings, the disadvantage being that we are dropping all the␣
 ↪tokens after the kth one
def lineup_k_TweetEmbeddings(raw_tokens, k = 25):
    embeddings = vec.get_vecs_by_tokens(raw_tokens, lower_case_backup=True)
    #n_embs = 0
    emb_stacked = torch.zeros((k, embeddings.shape[1]))
    j = 0
    for i in range(min(embeddings.shape[0], k)):
        if embeddings[i].abs().sum() <= 1e-3:
            continue
        else:
            emb_stacked[j] = embeddings[i]
            j += 1
    if np.any(np.isnan(emb_stacked.numpy())):
        print(f"exists an nan: {emb_stacked}")
    return emb_stacked.flatten()

#X_int = df['tokens_raw'].apply(lambda x: lineup_k_TweetEmbeddings(x)).values
```

[193]:

```
# the embeddings will be in a tuple of tensors, so we need to stack them into a␣
 ↪single tensor
torch.stack(tuple(X_int)).size()
```

[193]: torch.Size([9000, 50])

[194]:

```
# alternatively, we could sum up the embeddings for each token in a tweet (then␣
 ↪we don't have worry about a max length)
def sum_Tweet_Embeddings(raw_tokens):
    embeddings = vec.get_vecs_by_tokens(raw_tokens, lower_case_backup=True)
    n_embs = 0
    emb_sum = torch.zeros((embeddings.shape[1]))
    for i in range(embeddings.shape[0]):
        if embeddings[i].abs().sum() > 0:
            n_embs += 1
            emb_sum += embeddings[i]
    if n_embs > 0:
        emb_avg = emb_sum / n_embs
    else:
```

```
        emb_avg = torch.zeros((embeddings.shape[1]))
    if np.any(np.isnan(emb_avg.numpy())):
        print(f"exists an nan: {emb_sum}")
    return emb_avg

X_int = df['tokens_raw'].apply(lambda x: sum_Tweet_Embeddings(x)).values
```

[195]:
```
X_int.shape # one tweet will now be represented by a single rank-one 50-element␣
↪tensor
```

[195]: (9000,)

[196]:
```
if len(X_int[0] > 50):
    avg_embedding = False
else:
    avg_embedding = True

X = torch.stack(tuple(X_int))
X.shape
#X[:2]
```

[196]: torch.Size([9000, 50])

There should be 9000 rows in X, since this is the number of tweets (i.e. observations) in the training data.

The number of columns is the *embedding size* times the number of max number of tokens we'll keep for each tweet. In this case, keeping 25 means that the number of columns will be: * 50 * 25 –> 1250.

[197]:
```
labels = df['sentiment'].unique()
enum_labels = enumerate(labels)
label_to_idx = dict((lab, i) for i,lab in enum_labels)
print(f"label dictionary: {label_to_idx}")
y = torch.tensor([label_to_idx[lab] for lab in df['sentiment']])
```

```
label dictionary: {'neutral': 0, 'positive': 1, 'negative': 2}
```

[198]:
```
y[:10]
```

[198]: tensor([0, 1, 2, 0, 0, 2, 2, 2, 0, 1])

[199]:
```
# Can be a good idea to occassionally check that the dims (or shapes) agree for␣
↪the inputs (X) and labels (y)
assert len(X) == len(y)
```

[200]:
```
class AirlineTweetDataset(Dataset):
    def __init__(self, observations, labels):
```

```
        self.obs = observations
        self.labs = labels
        self.create_split(len(observations))

    def create_split(self, n, seed=2, train_perc=0.7):
        random.seed(seed)
        indices = list(range(n))
        random.shuffle(indices)
        self._train_ids = list(indices[:int(n * train_perc)])
        self._test_ids = list(indices[int(n * train_perc):])
        self._split_X = self.obs[self._train_ids]
        self._split_y = self.labs[self._train_ids]

    def set_split(self, split='train'):
        if split == 'train':
            self._split_X = self.obs[self._train_ids]
            self._split_y = self.labs[self._train_ids]
        else:
            self._split_X = self.obs[self._test_ids]
            self._split_y = self.labs[self._test_ids]

    def __len__(self):
        return len(self._split_y)

    def __getitem__(self, idx):
        return {'x':self._split_X[idx], 'y':self._split_y[idx]}

    def get_num_batches(self, batch_size):
        return len(self) // batch_size

dataset = AirlineTweetDataset(X, y)
dataset.create_split(len(X), seed=42, train_perc=0.85)
```

[201]:
```
dataset.set_split('train')
print(f"len(dataset) = {len(dataset)}")
#len(dataset[:]['x'])
dataset[0]['x']
```

```
len(dataset) = 7650
```

[201]:
```
tensor([ 0.2755,  0.3532,  0.1414, -0.0719,  0.2755,  0.0436, -0.4410, -0.0756,
        -0.1834, -0.1497, -0.0589,  0.1035, -0.5399, -0.0845,  0.6077,  0.1494,
        -0.1089,  0.0532, -0.5013, -0.3242, -0.0101,  0.3278,  0.2874,  0.0467,
         0.3526, -1.5300, -0.3083,  0.2123,  0.4151, -0.3225,  3.1063,  0.2745,
        -0.2358, -0.1187,  0.0452, -0.1378,  0.3318,  0.0707,  0.1511,  0.0235,
        -0.1112,  0.0702,  0.0894,  0.2073, -0.2028, -0.0087, -0.0561, -0.1535,
         0.0571,  0.1092])
```

[202]:
```
assert not np.any(np.isnan(dataset[:]['x'].numpy()))
assert np.all(np.isfinite(dataset[:]['x'].numpy()))
```

[203]:
```
class AirlineTweetClassifier(nn.Module):
    """ A 2-layer Multilayer Perceptron for classifying surnames """
    def __init__(self, input_dim, hidden_dim, output_dim):
        """
        Args:
            input_dim (int): the size of the input embeddings
            hidden_dim (int): the output size of the first Linear layer
            output_dim (int): the output size of the second Linear layer
        """
        super(AirlineTweetClassifier, self).__init__()

        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, 32)
        self.fc3 = nn.Linear(32, output_dim)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x_in, apply_softmax=False):
        """The forward pass of the classifier

        Args:
            x_in (torch.Tensor): an input data tensor.
                x_in.shape should be (batch, input_dim)
            apply_softmax (bool): a flag for the softmax activation
                should be false if used with the Cross Entropy losses
        Returns:
            the resulting tensor. tensor.shape should be (batch, output_dim)
        """
        intermediate_vector = F.relu(self.fc1(x_in))

        intermediate_vector = F.relu(self.fc2(intermediate_vector))
        intermediate_vector = self.dropout(intermediate_vector)

        prediction_vector = self.fc3(intermediate_vector)

        if apply_softmax:
            prediction_vector = F.softmax(prediction_vector, dim=1)

        return prediction_vector
```

[204]:
```
batch_size = 64
learning_rate = 0.0005 # 0.005
num_epochs = 30

device = 'cpu'
```

```
#device = torch.device('cuda' if torch.backend.mps.is_available() else 'cpu')

dataloader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=True)
```

[205]:
```
dataset.set_split('train')
print(len(dataloader) * batch_size)
dataset.set_split('val')
print(len(dataloader) * batch_size)
```

```
7680
1408
```

[206]:
```
model = AirlineTweetClassifier(len(dataset[0]['x']), 128, 3)

# define loss function and optimizer
#weights = 1 / torch.tensor([15.0, 65.0, 20.0])
loss_fun = nn.CrossEntropyLoss()#weights)

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

[207]:
```
seed = 2
np.random.seed(seed)
torch.manual_seed(seed)
random.seed(seed)
```

[208]:
```
import tqdm.auto

epoch_bar = tqdm.notebook.tqdm(desc='training routine', total=num_epochs,␣
 ↪position=0)

dataset.set_split('train')
train_bar = tqdm.notebook.tqdm(desc='split=train', total=dataset.
 ↪get_num_batches(batch_size), position=1, leave=True)

dataset.set_split('val')
val_bar = tqdm.notebook.tqdm(desc='split=val', total=dataset.
 ↪get_num_batches(batch_size), position=1, leave=True)

losses = {'train':[], 'val':[]}

for epoch in range(num_epochs):

    dataset.set_split('train')
    model.train()
    running_loss_train = 0.0

    for batch_i, batch_data in enumerate(dataloader):
```

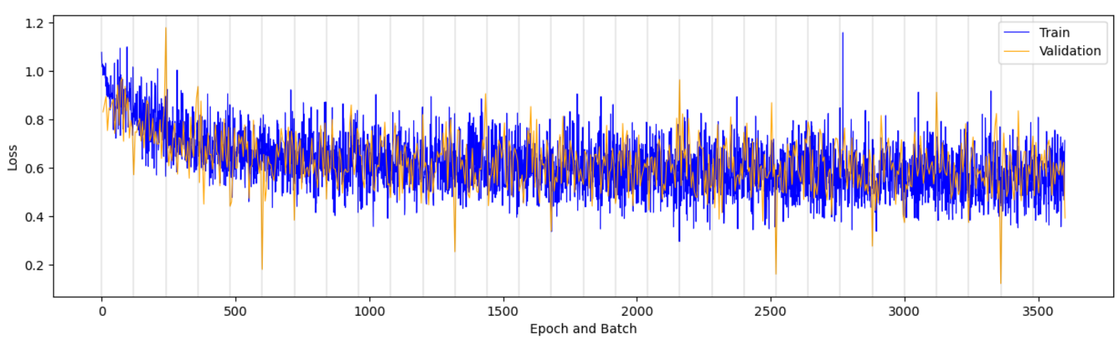
```
        tweets = batch_data['x'].to(device)
        labels = batch_data['y'].to(device)

        # forward
        outputs = model(tweets)
        loss = loss_fun(outputs, labels)
        losses['train'].append(loss.item())
        running_loss_train += loss.item() #/ batch_size

        # backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        #if (batch_i+1) % 10 == 0:
        #    print(f"    train batch {batch_i+1:3.0f} (of {len(dataloader):3.
↪0f}) loss: {loss.item():.4f}")
            # update bar
        train_bar.set_postfix(loss=running_loss_train, epoch=epoch)
        train_bar.update()

    train_bar.set_postfix(loss=running_loss_train/dataset.
↪get_num_batches(batch_size), epoch=epoch)
    train_bar.update()

    #running_loss_train = running_loss_train / len(dataset)

    dataset.set_split('val')
    model.eval() # turn off the automatic differentiation
    running_loss_val = 0.0

    for batch_i, batch_data in enumerate(dataloader):
        tweets = batch_data['x'].to(device)
        labels = batch_data['y'].to(device)

        # forward (no backward step for validation data)
        outputs = model(tweets)
        loss = loss_fun(outputs, labels)
        losses['val'].append(loss.item())
        running_loss_val += loss.item() #/ batch_size
        #if (batch_i+1) % 20 == 0:
        #    print(f"    valid batch {i+1:3.0f} (of {len(dataloader):3.0f})␣
↪loss: {loss.item():.4f}")
        val_bar.set_postfix(loss=running_loss_val, epoch=epoch)
        val_bar.update()
```

```
    val_bar.set_postfix(loss=running_loss_val/dataset.
↪get_num_batches(batch_size), epoch=epoch)
    val_bar.update()


    train_bar.n = 0
    val_bar.n = 0
    epoch_bar.update()

    #running_loss_val = running_loss_val / len(dataset)

    #print(f"Epoch {epoch+1} (of {num_epochs}): mean train loss =␣
↪{running_loss_train:.4f}, mean val loss = {running_loss_val:.4f}")
```

```
training routine:   0%|          | 0/30 [00:00<?, ?it/s]

split=train:   0%|          | 0/119 [00:00<?, ?it/s]

split=val:   0%|          | 0/21 [00:00<?, ?it/s]
```

[209]:
```
matplotlib.rc('figure', figsize=(15,4))
val_ticks = [(i+1)*len(losses['train'])/len(losses['val']) for i in␣
↪range(len(losses['val']))]
plt.plot(range(len(losses['train'])), losses['train'], c='blue', lw=0.75)
plt.plot(val_ticks, losses['val'], c='orange', lw=0.75)
for i in range(num_epochs):
    plt.axvline(x=i*len(losses['train'])/num_epochs, c='black', lw=0.25,␣
↪alpha=0.5)
plt.ylabel('Loss')
plt.xlabel('Epoch and Batch')
plt.legend(('Train','Validation'))
```

[209]:
```
<matplotlib.legend.Legend at 0x2c2940050>
```



[210]:
```
dataset.set_split('val')
len(dataset)
```

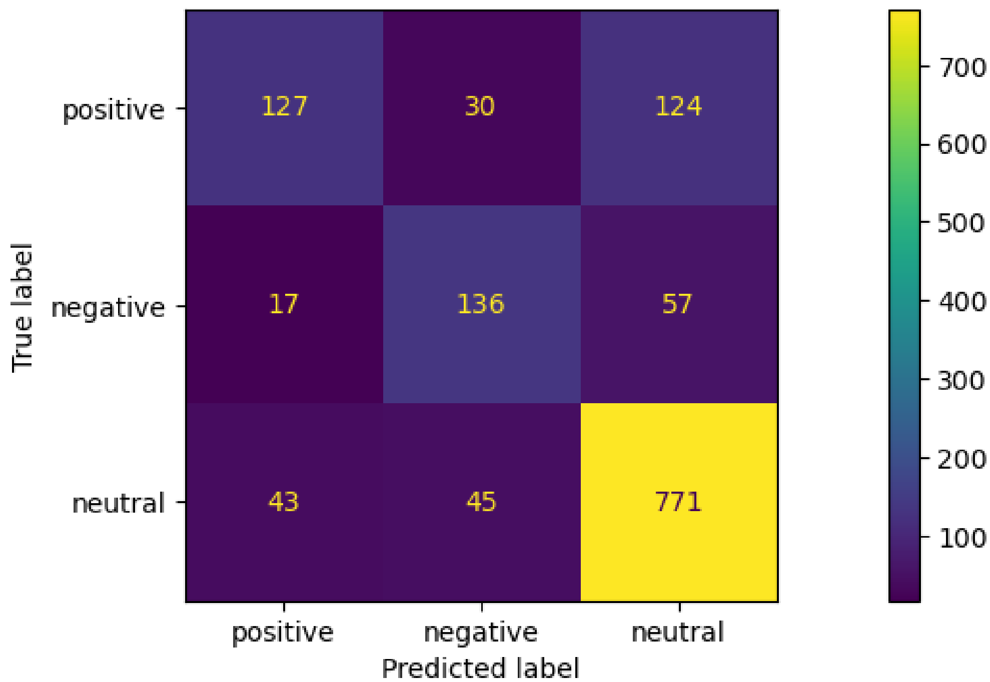
[210]: 1350

[211]:
```
# Test the model
# In test phase, we don't need to compute gradients (for memory efficiency)
y_true = []
y_pred = []

dataset.set_split('val')
with torch.no_grad():
    correct = 0
    total = 0
    for batch_data in dataloader:
        tweets = batch_data['x'].to(device)
        labels = batch_data['y'].to(device)
        outputs = model(tweets)
        _, predicted = torch.max(outputs.data, 1)
        y_true += labels.tolist()
        y_pred += predicted.tolist()
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print(f"Accuracy (on {len(dataloader)*batch_size} validation tweets): {100␣
↪* correct / total:.2f}%")
```

```
Accuracy (on 1408 validation tweets): 76.59%
```

[212]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(cm,␣
↪display_labels=['positive','negative','neutral'])
disp.plot()
```

[212]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c4f50fd0>




[213]:
```python
# length of an input is
len(dataset[0]['x'])
```

[213]: 50

[214]:
```python
import torchsummary
torchsummary.summary(model, tuple(dataset[0]['x'].size()))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 128]           6,528
            Linear-2                   [-1, 32]           4,128
           Dropout-3                   [-1, 32]               0
            Linear-4                    [-1, 3]              99
================================================================
Total params: 10,755
Trainable params: 10,755
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.04
Estimated Total Size (MB): 0.04
----------------------------------------------------------------
```

[215]:
```
df_test['tokens_raw'] = df_test['text'].apply(lambda x: tokenizer(x.lower()))
X_test_int = df_test['tokens_raw'].apply(lambda x: sum_Tweet_Embeddings(x)).
 ↪values
X_test = torch.stack(tuple(X_test_int))
X_test.shape
```

[215]:
```
torch.Size([1000, 50])
```

[216]:
```
y_test = torch.tensor([label_to_idx[lab] for lab in df_test['sentiment']])
```

[217]:
```
test_dataset = AirlineTweetDataset(X_test, y_test)
test_dataset.create_split(len(X_test), seed=42, train_perc=1.0)
```

[218]:
```
len(test_dataset)
```

[218]:
```
1000
```

[219]:
```
test_dataset[999]
```

[219]:
```
{'x': tensor([ 4.2224e-01,  1.4241e-01,  1.2053e-01, -1.7798e-03,  4.0264e-01,
         -1.3060e-01, -5.1082e-01, -1.8689e-01, -1.1695e-01, -8.3994e-02,
         -9.4942e-02,  4.1504e-01, -2.4253e-01, -1.4244e-01,  5.1127e-01,
          4.0539e-01, -5.6116e-02,  1.4513e-02,  9.3961e-02, -6.1356e-01,
          3.3893e-02,  2.3875e-01,  1.5466e-01,  1.9822e-01,  2.5455e-01,
         -1.5758e+00, -3.5155e-01,  1.2649e-01,  5.5148e-01, -5.4988e-01,
          3.1252e+00,  3.4321e-01, -4.2093e-01, -3.0634e-01, -1.2966e-01,
         -1.2857e-01,  8.4230e-04,  5.0270e-02,  1.3142e-01, -2.5790e-01,
          1.8883e-01,  1.0731e-01,  4.9320e-02,  3.1827e-01, -5.0537e-02,
          1.0613e-01,  1.8867e-02,  1.0771e-01, -6.1917e-02,  4.5576e-01]),
 'y': tensor(2)}
```

[220]:
```
bs = 100
test_loader = DataLoader(dataset=test_dataset, batch_size=bs, shuffle=False)
```

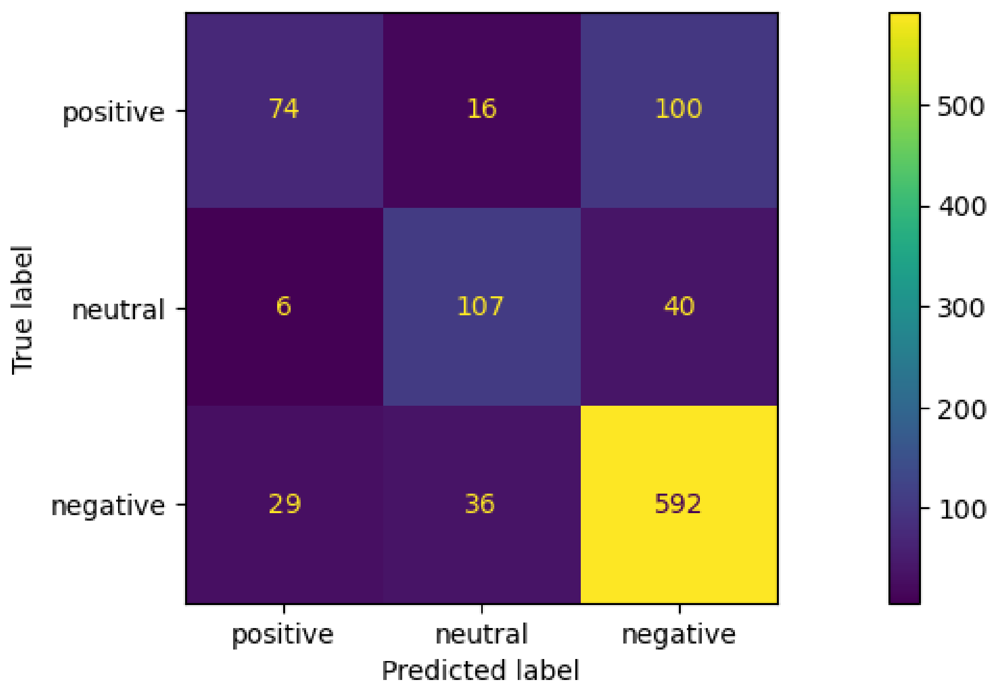
[221]:
```
y_true = []
y_pred = []
with torch.no_grad():
    correct = 0
    total = 0
    for batch_data in test_loader:
        tweets = batch_data['x'].to(device)
        labels = batch_data['y'].to(device)
        outputs = model(tweets)
        _, predicted = torch.max(outputs.data, 1)
        y_true += labels.tolist()
        y_pred += predicted.tolist()
        total += labels.size(0)
```

```
        correct += (predicted == labels).sum().item()

    print(f"Accuracy (on {len(test_loader)*bs} test tweets): {100 * correct /␣
↪total:.2f}%")
```

```
Accuracy (on 1000 test tweets): 77.30%
```

[222]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(cm,␣
↪display_labels=['positive','neutral','negative'])
disp.plot()
```

[222]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c2d97a10>
```




[ ]:
```

```