nb11b_airline_tweets_MLP_w_1hot_encoding

November 1, 2023

1 CS 39AA - Notebook 11b: Airline Tweets MLP w/ 1-hot Encoding

This notebook exists only so that we have a fair point of comparison with our Airline Tweets MLP using Word Embeddings. That notebook has more documentation and explanation of the steps taken. This notebook is nearly identical but uses one-hot word representations instead.

[85]:
```
import torch
import random
import matplotlib

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tqdm.auto
from torch.utils.data import Dataset, DataLoader
```

[86]:
```
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)
```

[86]:
```
  sentiment  \
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!
```

[87]:
```python
random.seed(2)
indices = list(range(len(df)))
random.shuffle(indices)

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

[88]:
```python
df_test.shape
df.shape
```

[88]:
```
(9000, 2)
```

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).

[89]:
```python
df.sentiment.value_counts(normalize=True)
```

[89]:
```
negative    0.653556
neutral     0.191111
positive    0.155333
Name: sentiment, dtype: float64
```

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

```
[90]: from nltk.tokenize import TweetTokenizer
      tk = TweetTokenizer()
      df['tokens_raw'] = df['text'].apply(lambda x: tk.tokenize(x.lower()))
      df.head()
```

```
[90]:      sentiment  \
      7010  positive
      9477   neutral
      4584   neutral
      3460  negative
      9065  negative

                                                                      text  \
      7010                                          @AmericanAir I ended up on a
      flight to LA my fourth time on standby. Thanks! http://t.co/NA5G5EAKPA
      9477   @JetBlue thanks! I only loose 'em at airports…1st time we found it. I
      think @fitbit needs to make flexes that stay on when carrying bags!
      4584  @SouthwestAir can take u to Midway-Chicago March 8th-April 6th. Can't make
      it then? @AmericanAir can get u to @fly2ohare year round. #FlyPBI
      3460                           @AmericanAir is this how you let your employees
      treat your loyal customers? #attackingbabymomma #crazinessintherockies
      9065
      @USAirways so I still need to stay on hold? http://t.co/04SDytT7zd

                   tokens_raw
      7010                                                       [@americanair, i,
      ended, up, on, a, flight, to, la, my, fourth, time, on, standby, ., thanks, !,
      http://t.co/na5g5eakpa]
      9477  [@jetblue, thanks, !, i, only, loose, ', em, at, airports, …, 1st, time,
      we, found, it, ., i, think, @fitbit, needs, to, make, flexes, that, stay, on,
      when, carrying, bags, !]
      4584      [@southwestair, can, take, u, to, midway-chicago, march, 8th, -,
      april, 6th, ., can't, make, it, then, ?, @americanair, can, get, u, to,
      @fly2ohare, year, round, ., #flypbi]
      3460                                            [@americanair, is, this, how,
      you, let, your, employees, treat, your, loyal, customers, ?,
      #attackingbabymomma, #crazinessintherockies]
      9065
      [@usairways, so, i, still, need, to, stay, on, hold, ?, http://t.co/04sdytt7zd]
```

Next, let's remove common stop words (e.g. "*the*", "*in*", etc.). In this next cell we will also remove some characters/punctuation, as well as hashtag tokens.

```
[91]: import re
      from nltk.corpus import stopwords
      stops = set(stopwords.words('english'))
```

```
chars2remove = set(['.','!','/', '?'])
df['tokens_raw'] = df['tokens_raw'].apply(lambda x: [w for w in x if w not in␣
 ↪stops])
df['tokens_raw'] = df['tokens_raw'].apply(lambda x: [w for w in x if w not in␣
 ↪chars2remove])
df['tokens_raw'] = df['tokens_raw'].apply(lambda x: [w for w in x if not re.
 ↪match('^#', w)]) # remove hashtags
#df['tokens_raw'] = df['tokens_raw'].apply(lambda x: [w for w in x if not re.
 ↪match('^http', w)]) # remove web links
#df['tokens_raw'] = df['tokens_raw'].apply(lambda x: [w for w in x if not re.
 ↪match('^@', w)]) # remove web links

df.head()
```

[91]:
```
     sentiment  \
7010  positive
9477   neutral
4584   neutral
3460  negative
9065  negative


                                                                 text  \
7010                                            @AmericanAir I ended up on a
flight to LA my fourth time on standby. Thanks! http://t.co/NA5G5EAKPA
9477   @JetBlue thanks! I only loose 'em at airports…1st time we found it. I
think @fitbit needs to make flexes that stay on when carrying bags!
4584  @SouthwestAir can take u to Midway-Chicago March 8th-April 6th. Can't make
it then? @AmericanAir can get u to @fly2ohare year round. #FlyPBI
3460                            @AmericanAir is this how you let your employees
treat your loyal customers? #attackingbabymomma #crazinessintherockies
9065
@USAirways so I still need to stay on hold? http://t.co/04SDytT7zd


                                                      tokens_raw
7010                                           [@americanair, ended, flight, la,
fourth, time, standby, thanks, http://t.co/na5g5eakpa]
9477     [@jetblue, thanks, loose, ', em, airports, …, 1st, time, found,
think, @fitbit, needs, make, flexes, stay, carrying, bags]
4584  [@southwestair, take, u, midway-chicago, march, 8th, -, april, 6th, can't,
make, @americanair, get, u, @fly2ohare, year, round]
3460
[@americanair, let, employees, treat, loyal, customers]
9065
[@usairways, still, need, stay, hold, http://t.co/04sdytt7zd]
```

For the final step of text pre-processing we will lemmatize the tokens. Note that there are much better ways to do this but that we want to use a simple lemmatizer. For example, some lemmatizers also utilize a model internally to predict the part-of-speech for each word, since whether the word is a noun, adjective, verb, etc. will affect how lemmatization is done. Since we want to keep things simple here, and focus only on the lemmatization step, we'll assume every word is the same part of speech. Note that this is not by any means ideal (try to identify the incorrectly lemmatized token in the five tweets printed out below). In practice we would certainly utilize a 'smarter' lemmatizer .

[92]:
```
from nltk.stem import WordNetLemmatizer
# also need to run following one time on your system (can be done outside of␣
↪this notebook)
# import nltk
# nltk.download('wordnet')
# nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()
df['tokens'] = df['tokens_raw'].apply(lambda x: [lemmatizer.lemmatize(w,␣
↪pos="v") for w in x])
#df['tokens'] = df['tokens_raw'].apply(lambda x: [lemmatizer.lemmatize(w) for w␣
↪in x])
df.head()
```

[92]:
```
     sentiment  \
7010  positive
9477   neutral
4584   neutral
3460  negative
9065  negative


                                                             text  \
7010                                           @AmericanAir I ended up on a
flight to LA my fourth time on standby. Thanks! http://t.co/NA5G5EAKPA
9477   @JetBlue thanks! I only loose 'em at airports…1st time we found it. I
think @fitbit needs to make flexes that stay on when carrying bags!
4584  @SouthwestAir can take u to Midway-Chicago March 8th-April 6th. Can't make
it then? @AmericanAir can get u to @fly2ohare year round. #FlyPBI
3460                        @AmericanAir is this how you let your employees
treat your loyal customers? #attackingbabymomma #crazinessintherockies
9065
@USAirways so I still need to stay on hold? http://t.co/04SDytT7zd

                                                   tokens_raw  \
7010                                            [@americanair, ended, flight, la,
fourth, time, standby, thanks, http://t.co/na5g5eakpa]
9477     [@jetblue, thanks, loose, ', em, airports, …, 1st, time, found,
think, @fitbit, needs, make, flexes, stay, carrying, bags]
4584  [@southwestair, take, u, midway-chicago, march, 8th, -, april, 6th, can't,
make, @americanair, get, u, @fly2ohare, year, round]
3460
```

```
[@americanair, let, employees, treat, loyal, customers]
9065
[@usairways, still, need, stay, hold, http://t.co/04sdytt7zd]

                                                tokens
7010                                             [@americanair, end, flight, la,
fourth, time, standby, thank, http://t.co/na5g5eakpa]
9477             [@jetblue, thank, loose, ', em, airports, …, 1st, time,
find, think, @fitbit, need, make, flex, stay, carry, bag]
4584  [@southwestair, take, u, midway-chicago, march, 8th, -, april, 6th, can't,
make, @americanair, get, u, @fly2ohare, year, round]
3460
[@americanair, let, employees, treat, loyal, customers]
9065
[@usairways, still, need, stay, hold, http://t.co/04sdytt7zd]
```

Since each tweet is currently stored as a string we then created a new column that was a list of each of the words in the tweet (since the default delimiter is a space character). Next, we created a vocabularly sorted by frequency for the full dataset, the subset of positive tweets, negative tweets, and neutral tweets.

The input to the sklearn vectorizer function requires that each observation (i.e. tweet) is in the form of a string, rather than a list of tokens. So we first need to combine the individual tokens for each tweet back into a string, which we do here:

[93]:
```
df['textclean'] = df['tokens'].apply(lambda x: ' '.join(x))
df.head()
```

[93]:
```
     sentiment  \
7010  positive
9477   neutral
4584   neutral
3460  negative
9065  negative


                                                               text  \
7010                                            @AmericanAir I ended up on a
flight to LA my fourth time on standby. Thanks! http://t.co/NA5G5EAKPA
9477   @JetBlue thanks! I only loose 'em at airports…1st time we found it. I
think @fitbit needs to make flexes that stay on when carrying bags!
4584  @SouthwestAir can take u to Midway-Chicago March 8th-April 6th. Can't make
it then? @AmericanAir can get u to @fly2ohare year round. #FlyPBI
3460                       @AmericanAir is this how you let your employees
treat your loyal customers? #attackingbabymomma #crazinessintherockies
9065
@USAirways so I still need to stay on hold? http://t.co/04SDytT7zd

                                          tokens_raw  \
7010                                          [@americanair, ended, flight, la,
fourth, time, standby, thanks, http://t.co/na5g5eakpa]
9477     [@jetblue, thanks, loose, ', em, airports, …, 1st, time, found,
think, @fitbit, needs, make, flexes, stay, carrying, bags]
4584  [@southwestair, take, u, midway-chicago, march, 8th, -, april, 6th, can't,
make, @americanair, get, u, @fly2ohare, year, round]
3460
[@americanair, let, employees, treat, loyal, customers]
9065
[@usairways, still, need, stay, hold, http://t.co/04sdytt7zd]

                                                tokens  \
7010                                            [@americanair, end, flight, la,
fourth, time, standby, thank, http://t.co/na5g5eakpa]
9477              [@jetblue, thank, loose, ', em, airports, …, 1st, time,
find, think, @fitbit, need, make, flex, stay, carry, bag]
4584  [@southwestair, take, u, midway-chicago, march, 8th, -, april, 6th, can't,
make, @americanair, get, u, @fly2ohare, year, round]
3460
[@americanair, let, employees, treat, loyal, customers]
9065
[@usairways, still, need, stay, hold, http://t.co/04sdytt7zd]

                          textclean
7010                                    @americanair end flight la fourth time
standby thank http://t.co/na5g5eakpa
9477               @jetblue thank loose ' em airports … 1st time find think
@fitbit need make flex stay carry bag
4584  @southwestair take u midway-chicago march 8th - april 6th can't make
@americanair get u @fly2ohare year round
3460                                                             @americanair
let employees treat loyal customers
9065                                                       @usairways still
need stay hold http://t.co/04sdytt7zd
```

Now we will load the term-frequency inverse-document-frequency vectorizer from sklearn, `TfidfVectorizer`, to convert each tweet into a vector. We'll go ahead and call the resulting vectorized data, `X`, or `X_train` since it is only the training dataset. As with conventional statistical models, "X" represents the set of predictors, or independent variables.

Also, note that `TfidfVectorizer` is a powerful text processing object. It has the ability to remove stop words, strip symbols, and do much of the work that our manual tokenization did. As such, we could easily use the original tweet text here, but we'll go ahead and continue with our manually tokenized data in the column, `textclean`.

[94]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()
```

```
X_np = tfidf_vectorizer.fit_transform(df['textclean']).toarray()

print(f"X_np.shape = {X_np.shape}")
type(X_np)
```

```
X_np.shape = (9000, 9008)
```

[94]:
```
numpy.ndarray
```

[95]:
```
X_np[:3,:5]
```

[95]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[96]:
```
X = torch.tensor(X_np).float()
X.size()
```

[96]:
```
torch.Size([9000, 9008])
```

[97]:
```
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: {'positive': 0, 'neutral': 1, 'negative': 2}
```

[98]:
```
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)
```

[99]:
```
dataset.set_split('train')
print(f"len(dataset) = {len(dataset)}")
#len(dataset[:]['x'])
dataset[0]['x']
```

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

[99]:
```
tensor([0., 0., 0.,  …, 0., 0., 0.])
```

[100]:
```
assert not np.any(np.isnan(dataset[:]['x'].numpy()))
assert np.all(np.isfinite(dataset[:]['x'].numpy()))
```

[101]:
```
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
```

[102]:
```
batch_size = 32
learning_rate = 0.0005 # 0.005
num_epochs = 30

#device = torch.device('mps' if torch.backends.mps.is_available() else 'cpu')
device = 'cpu'

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

[103]:
```
dataset.set_split('train')
print(len(dataloader) * batch_size)
dataset.set_split('val')
print(len(dataloader) * batch_size)
```

```
7680
1376
```

[104]:
```
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)
```

[105]:
```
seed = 2
np.random.seed(seed)
```

```
torch.manual_seed(seed)
random.seed(seed)
```

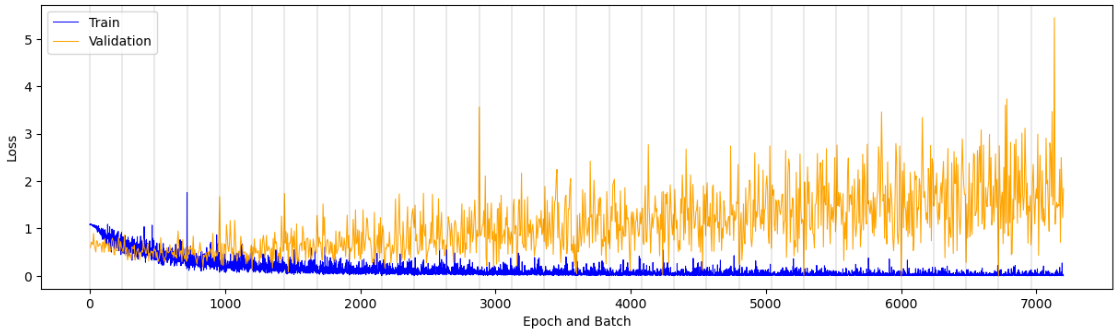
[106]:
```
epoch_bar = tqdm.notebook.tqdm(desc='training routine', total=num_epochs,␣
 ↪position=0)

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()

        # 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()
        #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)
```

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

split=train:   0%|          | 0/239 [00:00<?, ?it/s]

split=val:   0%|          | 0/42 [00:00<?, ?it/s]
```

[107]:
```
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'))
```

[107]:
```
<matplotlib.legend.Legend at 0x2ea314950>
```

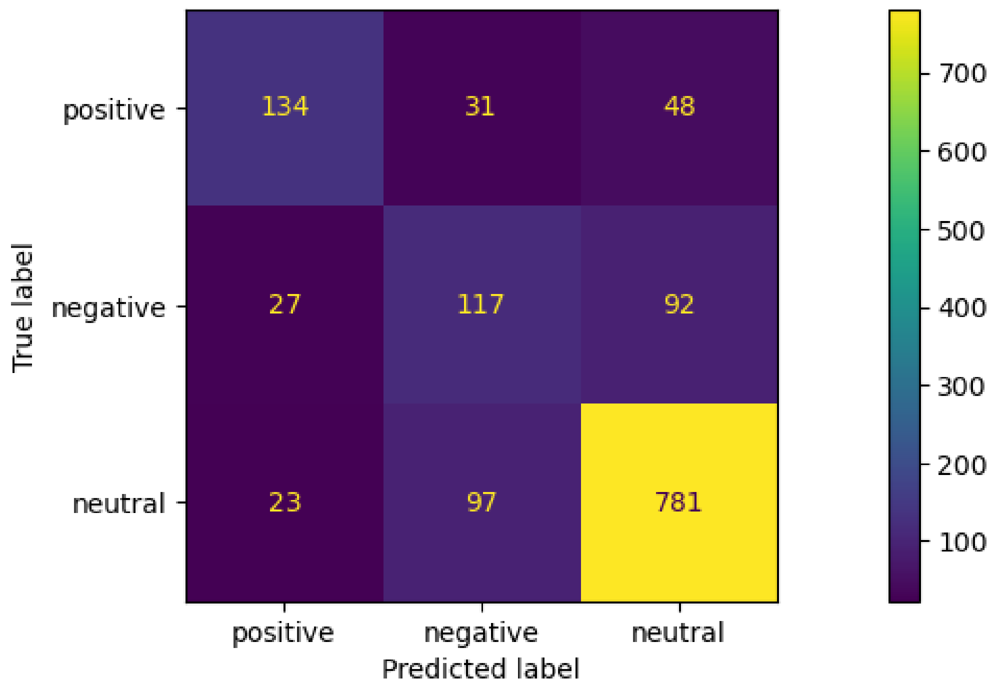


[108]:
```
# Test the model
# In test phase, we don't need to compute gradients (for memory efficiency)
y_true = []
y_pred = []
with torch.no_grad():
    correct = 0
    total = 0
    for batch_data in dataloader:
        tweets = batch_data['x'].to(device)
        labels = batch_data['y'].to(device)
        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 1376 validation tweets): 76.44%
```

[109]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(cm,␣
 ↪display_labels=['positive','negative','neutral'])
disp.plot()
```

```
[109]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2ea375c90>
```



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

```
[110]: 9008
```

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

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 128]       1,153,152
            Linear-2                   [-1, 32]           4,128
           Dropout-3                   [-1, 32]               0
            Linear-4                    [-1, 3]              99
================================================================
Total params: 1,157,379
Trainable params: 1,157,379
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.03
Forward/backward pass size (MB): 0.00
Params size (MB): 4.42
```

```
Estimated Total Size (MB): 4.45
----------------------------------------------------------------
```

[112]:
```
tk = TweetTokenizer()
df_test['tokens_raw'] = df_test['text'].apply(lambda x: tk.tokenize(x.lower()))
df_test['tokens_raw'] = df_test['tokens_raw'].apply(lambda x: [w for w in x if␣
 ↪w not in stops])
df_test['tokens_raw'] = df_test['tokens_raw'].apply(lambda x: [w for w in x if␣
 ↪w not in chars2remove])
df_test['tokens_raw'] = df_test['tokens_raw'].apply(lambda x: [w for w in x if␣
 ↪not re.match('^#', w)]) # remove hashtags
df_test.head()
```

[112]:
```
     sentiment  \
7391  negative
6692  negative
2849  negative
3824  negative
9386  negative

                                                              text  \
7391  Why even ask me to DM you and offer help if you "can't do anything"
@united #terriblecustomerservice #unitedairlines http://t.co/feC4i3Vwq7
6692   @USAirways Do you have any pride in your service? Any concerns for my
wife and everyone else on that flight? Or you just don't care????!!!
2849     @USAirways Would you guys please send service agents to gate B15 in
Philly?  All the people missed there connections and there's only 2.
3824                                                         @JetBlue apparently
the plane was delayed coming up from San Juan. Monsoon there today?
9386                  @JetBlue This is the error message: Paper tickets cannot be
serviced on-line.\nPlease see a JetBlue Crewmember for assistance.

                                  tokens_raw
7391                                    [even, ask, dm, offer, help, ", can't,
anything, ", @united, http://t.co/fec4i3vwq7]
6692                                                  [@usairways, pride, service,
concerns, wife, everyone, else, flight, care]
2849    [@usairways, would, guys, please, send, service, agents, gate, b15,
philly, people, missed, connections, there's, 2]
3824                                                   [@jetblue, apparently, plane,
delayed, coming, san, juan, monsoon, today]
9386  [@jetblue, error, message, :, paper, tickets, cannot, serviced, on-line,
please, see, jetblue, crewmember, assistance]
```

[113]:
```
df_test['tokens'] = df_test['tokens_raw'].apply(lambda x: [lemmatizer.
 ↪lemmatize(w, pos="v") for w in x])
df_test['textclean'] = df_test['tokens'].apply(lambda x: ' '.join(x))
```

```
X_test_int = tfidf_vectorizer.transform(df_test['textclean']).toarray() # be␣
 ↪sure that we are using .transform() here, and not .fit_transform()

print(f"X_test_int.shape = {X_test_int.shape}")
type(X_test_int)
```

```
X_test_int.shape = (1000, 9008)
```

[113]:
```
numpy.ndarray
```

[114]:
```
X_test = torch.tensor(X_test_int).float()
X_test.size()
```

[114]:
```
torch.Size([1000, 9008])
```

[115]:
```
y_test = torch.tensor([label_to_idx[lab] for lab in df_test['sentiment']])
```

[116]:
```
test_dataset = AirlineTweetDataset(X_test, y_test)
test_dataset.create_split(len(X_test), seed=42, train_perc=1.0)
```

[117]:
```
len(test_dataset)
```

[117]:
```
1000
```

[118]:
```
test_dataset[999]
```

[118]:
```
{'x': tensor([0., 0., 0.,  …, 0., 0., 0.]), 'y': tensor(2)}
```

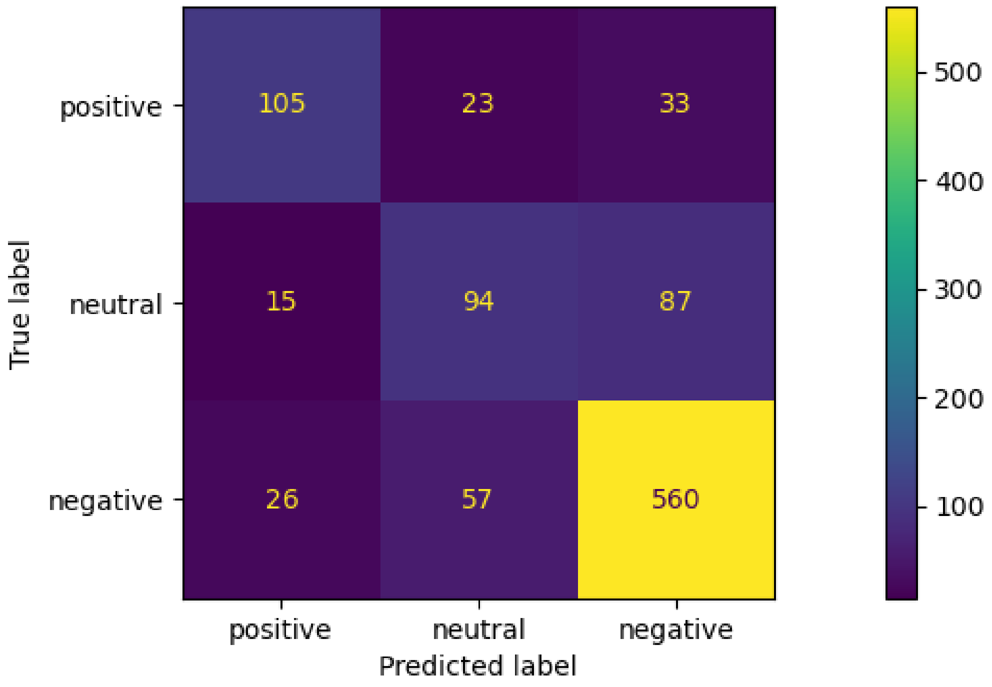
[119]:
```
bs = 500
test_loader = DataLoader(dataset=test_dataset, batch_size=bs, shuffle=False)
```

[120]:
```
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): 75.90%
```

[121]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(cm,␣
  ↪display_labels=['positive','neutral','negative'])
disp.plot()
```

[121]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x33236db90>
```



[ ]:

[ ]: