lab3

September 29, 2024

Zachary Proom

EN.605.646.81: Natural Language Processing

1 Lab #3

1.1 a

First, I split the training data in train.tsv into two groups, based on the class.

[1]:
```
import pandas as pd
```

[2]:
```
# Load in train.tsv.
training_data = pd.read_csv('train.tsv', sep = '\t', header = None)

# Add column names.
training_data.columns = ["stars", "docid", "text"]

# Split into two groups based on class.
training_data_negative = training_data.loc[training_data["stars"] == 2]
training_data_positive = training_data.loc[training_data["stars"] == 4]
```

I found ten words that indicate positive or negative sentiment, and I show their relative frequencies in the table below (i.e. the percent of reviews they appear in in each class). The first five words indicate positive sentiment, and the last five indicate negative sentiment.

[3]:
```
relative_freqs = pd.DataFrame(columns = ['word', 'positive', 'negative'])

for word in ["amazing", "awesome", "great", "incredible", "fantastic",␣
 ↪"terrible", "horrible", "worst", "awful", "bad"]:
    negative_freq = sum(training_data_negative['text'].str.contains(word))/
 ↪len(training_data_negative) * 100
    positive_freq = sum(training_data_positive['text'].str.contains(word))/
 ↪len(training_data_positive) * 100
    relative_freqs = pd.concat([relative_freqs, pd.DataFrame({"word": [word],␣
 ↪"positive": [positive_freq], "negative": [negative_freq]})], ignore_index =␣
 ↪True)

relative_freqs = relative_freqs.reset_index(drop=True)
```

```
relative_freqs
```

[3]:

	word	positive	negative
0	amazing	5.2	1.8
1	awesome	5.7	2.2
2	great	35.9	17.3
3	incredible	0.4	0.1
4	fantastic	3.7	1.0
5	terrible	0.3	3.2
6	horrible	0.6	3.1
7	worst	0.7	4.7
8	awful	0.6	2.5
9	bad	7.1	16.7

After reading a few of the reviews, I noticed that there's a lot of mixed language. Take this review as an example:

2 YT9tezwopYagEjTxIzN2dg i am just not a fan of this kind of pizza. i hate the sweet sauce, i hate that the ingredients are under the cheese so they don't get crunchy and crispy, the pepperoni is floppy. just not for me. the crust was kinda gooey like. my delivery was 30 minutes late but its ok bc i wasn't in a hurry. they were really nice i just can't stand this kind of pizza. if you like the sweet sauce and toppings under the cheese then go for it bc you'll probably love it!

The review is part of the negative class, but it contains several positive words such as "love", "nice", and "like". This is true for a lot of the reviews I read. A lot of the reviews are balanced and not polarized. This makes classifying sentiment more challenging.

1.2 b

Below I train a Multinomial Naive Bayes model ("bag of words model") on the training set. To do this, I use sklearn, a popular open source machine learning package. I use sklearn's make_pipeline() method to create the model. The pipeline applies the functions passed in as arguments to the training data. In this case, it first applies CountVectorizer() to the training data to convert it into a matrix of counts. Finally, it applies MultinomialNB() to train the Multinomial NB model on the matrix of counts. The only parameters I provided are the reviews and ratings in the training data. I didn't modify the behavior of MultinomialNB(). By default, it sets alpha=1.0, which means it uses Laplace smoothing.

[4]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn import metrics # Use this in part c to evaluate predictions.

X_train = training_data["text"]
y_train = training_data["stars"]
multinomialnb_model = make_pipeline(CountVectorizer(), MultinomialNB())

# Train the model.
multinomialnb_model.fit(X_train, y_train)
```

```
multinomialnb_model[1].n_features_in_
```

[4]: 11468

The total number of features is shown above: 11,468. It's all the unique words in the training data.

Next, I print a feature representation for the first document in the dev set.

[5]:
```
# Load in dev data.
dev_data = pd.read_csv('dev.tsv', sep = '\t', header = None)

# Add column names.
dev_data.columns = ["stars", "docid", "text"]

# Create CountVectorizer instance.
vectorizer = CountVectorizer()

# Fit and transform the first document in the dev set.
X = vectorizer.fit_transform([dev_data["text"][0]])

feature_rep = pd.DataFrame({"feature": vectorizer.get_feature_names_out(),␣
 ↪"frequency": X.toarray()[0].tolist()})
feature_rep = feature_rep.sort_values(by = ['frequency'], ascending = False)
feature_rep = feature_rep.reset_index(drop=True)

# Print all rows.
print(feature_rep.to_string())
```

```
       feature  frequency
0          the          6
1          and          6
2         menu          3
3          not          3
4        their          2
5         good          2
6           to          2
7           it          2
8         have          2
9         were          2
10       china          2
11          on          2
12    original          2
13          is          2
14      really          1
15      recent          1
16  restaurant          1
17      return          1
```

18	she	1
19	price	1
20	portion	1
21	site	1
22	size	1
23	so	1
24	again	1
25	steak	1
26	tasty	1
27	phone	1
28	this	1
29	told	1
30	trying	1
31	tso	1
32	under	1
33	wanted	1
34	web	1
35	what	1
36	when	1
37	years	1
38	yet	1
39	you	1
40	place	1
41	once	1
42	pepper	1
43	general	1
44	average	1
45	both	1
46	business	1
47	chef	1
48	chicken	1
49	deserves	1
50	does	1
51	enjoyed	1
52	fan	1
53	first	1
54	food	1
55	forward	1
56	friendly	1
57	grand	1
58	other	1
59	great	1
60	if	1
61	in	1
62	items	1
63	just	1
64	lady	1
65	look	1

```
66  management          1
67     matched          1
68         new          1
69          of          1
70      online          1
71      opened          1
72        your          1
```

Next, I print the docid and prediction (separated by a tab) for the first 10 documents in the dev file.

[6]:
```
X_test = dev_data["text"][0:10]

predictions = multinomialnb_model.predict(X_test)

for i in range(0, 10):
    print(dev_data["docid"][i] + "\t" + str(predictions[i]))
```

```
ZSJnW6faaNFQoqq4ALqYg   4
Rcbv11hm5AYEwZyqYwAvg   2
rkRTjhu5szaBggeFVcVJlA  4
dhmeDsQGUS1FXMLs49SWjQ  4
z9zfIMYmRRCE4ggfOIieEw  4
Xtb3pGSh39bqcozkBECw    2
DOUflAGzxLsXG6xOmR1w    2
ORxCEWURe08CTcZt95F4AQ  2
MzUg5twEcCyd0X6lBMP2Lg  2
uNlw2D5CYKk0wjNxLtYw    4
```

Finally, I make predictions for the dev and test partitions and write those to two files. The file names are dev_predictions_multinomialnb.tsv and test_predictions_multinomialnb.tsv.

[7]:
```
# Make predictions for the full dev data.
X_test = dev_data["text"]
predictions = multinomialnb_model.predict(X_test)

# Write predictions to a file.
predictions_df = pd.DataFrame({"docid": dev_data["docid"], "prediction":␣
 ↪predictions})
predictions_df.to_csv("dev_predictions_multinomialnb.tsv", sep = "\t", index =␣
 ↪False)

# Make predictions for the test data.
# Load in test data and add column names.
test_data = pd.read_csv('test.tsv', sep = '\t', header = None)
test_data.columns = ["stars", "docid", "text"]
X_test = test_data["text"]
predictions = multinomialnb_model.predict(X_test)
```

```
# Write predictions to a file.
predictions_df = pd.DataFrame({"docid": test_data["docid"], "prediction":␣
 ↪predictions})
predictions_df.to_csv("test_predictions_multinomialnb.tsv", sep = "\t", index =␣
 ↪False)
```

1.3 c

Next I evaluate the predictions of my Multinomial NB model from part b. I read in the predictions file (dev_predictions_multinomialnb.tsv) and calculate precision, recall, and F1 scores for the positive class (4 stars).

[8]:
```
# Read in the dev set predictions.
predictions_dev = pd.read_csv('dev_predictions_multinomialnb.tsv', sep = '\t')

# Join prediction DF with dev data.
combined_dev = pd.merge(dev_data, predictions_dev, on = "docid", how = "inner")

# Rename "stars" column as "actual".
combined_dev = combined_dev.rename(columns={"stars": "actual"})

# Filter data so it only includes observations from the positive class.
combined_dev_positive = combined_dev.loc[combined_dev["actual"] == 4]
combined_dev_negative = combined_dev.loc[combined_dev["actual"] == 2]

# Calculate precision.
# precision = true positives/(true positives + false positives)
n_true_positives = (combined_dev_positive["actual"] ==␣
 ↪combined_dev_positive["prediction"]).sum()
n_false_positives = (combined_dev_negative["actual"] !=␣
 ↪combined_dev_negative["prediction"]).sum()
precision = n_true_positives/(n_true_positives + n_false_positives)

# Calculate recall.
# recall = true positives/(true positives + false negatives)
n_false_negatives = (combined_dev_positive["actual"] !=␣
 ↪combined_dev_positive["prediction"]).sum()
recall = n_true_positives/(n_true_positives + n_false_negatives)

# Calculate F1.
f1 = (2 * precision * recall)/(precision + recall)

# Report results.
summary_stats_multinomialnb = pd.DataFrame({"statistic": ["precision",␣
 ↪"recall", "f1"], "value": [precision, recall, f1]})
summary_stats_multinomialnb
```

[8]:
```
   statistic     value
0  precision  0.835095
1     recall  0.790000
2         f1  0.811922
```

Below I share some interesting mistakes my classifier made.

[9]:
```
mistakes = combined_dev.loc[combined_dev["actual"] !=␣
 ↪combined_dev["prediction"]]
mistakes
```

[9]:
```
      actual                    docid  \
1          4   Rcbv11hm5AYEwZyqYwAvg
2          2  rkRTjhu5szaBggeFVcVJlA
7          4  0RxCEWURe08CTcZt95F4AQ
9          2    uNlw2D5CYKk0wjNxLtYw
14         2  HqtEtHHDgSM0ctJHehaWaw
…      …                       …
1988       4  T1Lu6UeSHH6AZmzl9pnz7w
1992       2  ggr9TDU0bVnZBEsiObxYEQ
1993       2   b1kLsgl6nc2blZXRAGjXw
1998       4   jV8aDYrQ4LFHb0b0hC3Iw
1999       4  v6pqzYfd7XFM4t6fWI1wQA

                                                   text  prediction
1     Meeting a friend for lunch, we had a little mi…           2
2     Olive Garden used to be a favorite of the fami…           4
7     Our Las Vegas friend suggested duck tacos afte…           2
9     Stopped in here for lunch the other day. Quiet…           4
14    I've never been a big fan of BK, I have held p…           4
…                                                   …         …
1988  I had great service here today. I stopped in f…           2
1992  Firstly, letme agree with Stephanie T. This pl…           4
1993  Been there on several Friday nights and the fo…           4
1998  As soon as I walked in I was greeted by the sw…           2
1999  This place is gonna cost me about $2,000.  ---…           2

[366 rows x 4 columns]
```

My model misclassified 366 reviews, including following doc IDs: 0RxCEWURe08CTcZt95F4AQ, gjWVccNw6kB2UycZAEzyQg, rkRTjhu5szaBggeFVcVJlA, and ggr9TDU0bVnZBEsiObxYEQ. The first two reviews are actually positive, but my model classifies them as negative. The latter two reviews are actually negative, but my model classifies them as positive. Below are the reviews:

Actual Class: Positive, Predicted Class: Negative

0RxCEWURe08CTcZt95F4AQ: Our Las Vegas friend suggested duck tacos after hanging out and having some drinks for a late night/ early morning (around 4am) snack. All we needed to hear was

tacos and we were in. The small bar area of this place is open 24/7 so we stopped in for some half priced tapas (happy hour from midnight to 8am) and to chat with Dave, our friend's friend and an incredibly cool guy that was working the overnight shift. It was 4am and I didn't want to be stuffed before going to bed, so I only ordered a few dishes. I had the steak skewers with teriyaki glaze and the garlic cheese bread. Both were good, but the garlic cheese bread was really good in its garlicy and cheesy-ness. I also had one of the duck tacos from one of the others' plates and it was decent, but if I got it again I'd get it without the creamy tomato cilantro sauce. Overall, I'd say the food is worth 3 or 3.5 stars, but Dave's coolness bumps this location up to a solid 4 stars.

gjWVccNw6kB2UycZAEzyQg: I hate to find out that there was another location closer to where I was. It doesn't matter because it was worth the trip. I ordered pick up because I was by myself and wanted to a bit of exploring. The place has a nice casual ambiance. They also have an outdoor seating area. I ordered the pad thai that comes with chicken and shrimp as a standard. Since it was lunch time, it was part of the lunch menu that came with a choice of soup or salad. I chose the soup, but was too stuffed to have it because I jumped right into the pad thai. I ordered it medium spicy, and it was pretty spicy. I can't imagine what hot would have tasted like. Actually, I'm more curious of what it would have tasted like in a mild flavor. The noodles were slightly overcooked, but still flavorful. I also had the Thai Iced Tea here, and it was perfect for the weather and a great combination with the tangy pad thai flavor. The prices are expensive compared to Chicago, but I hear it's typical for Phoenix/Scottsdale area. My lunch speciai and drink costed me a little over $15. I guess if you are craving it that bad like I was, you'll pay the price…

Actual Class: Negative, Predicted Class: Positive

rkRTjhu5szaBggeFVcVJlA: Olive Garden used to be a favorite of the family, recently they cut back the menu extensively and many of our favorites are gone. I suggest checking the menu online before coming to see what's left.

ggr9TDU0bVnZBEsiObxYEQ: Firstly, letme agree with Stephanie T. This place is far from 'rapid' at any time of the day. Whether there is a huge queue forming out of the door or just you in the whole shop, you will wait for what seems like an eternity before you are eventually served. Although to give credit where credit is due, the actual preperation time of the sandwiches is fairly quick - which may explain why they always look a bit messy and sloppy. However, once you stop being shallow about your food, the sandwiches here are actually pretty damn tasty and good value for what you pay for them.

The first two mistakes (actual class is positive) seem positive when considering the entire text, but there are a few negative words that may have led the model to misclassify them as negative. The first review (0RxCEWURe08CTcZt95F4AQ) includes the words "late" and "but", and the second review (gjWVccNw6kB2UycZAEzyQg) includes even stronger negative words like "hate" and "overcooked". Similarly, the second two mistakes (actual class is negative) include a few positive words that probably led the model to misclassify them. The first of these reviews (rkRTjhu5szaBggeFVcVJlA) uses the word "favorite", and the second review (ggr9TDU0bVnZBEsiObxYEQ) uses the words "quick", "tasty", and "good".

1.4 d

Next, I train an SVM model as an alternative to the Multinomial NB model above. I use the SVC class from sklearn to train the model. I first apply a tokenizer to the dev data, TfidfVectorizer(), to convert it into a matrix of TF-IDF scores. Next, I use the SVC class to train the SVM model on the

matrix of TF-IDF scores. I use the default parameters during training, except for one modification. I set the kernel argument to "linear". This means the model uses a "line" (hyperplane in higher dimensions) as the decision boundary between classes. I chose a linear kernel because I saw it's widely recommended online for sentiment classification. Some of the reasons a linear kernel is recommended for sentiment classification are that: (1) text data have high dimensionality and can be separated with a line/hyperplane; (2) it avoids overfitting the training data; and (3) it's faster to train than other kernels (e.g. polynomial, sigmoid, etc.).

[10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

X_train = training_data['text']
y_train = training_data["stars"]

# Use the vectorizer below to find feature set for first document in dev set.
vectorizer = TfidfVectorizer()

svm = make_pipeline(TfidfVectorizer(), SVC(kernel = "linear"))
svm.fit(X_train, y_train)
svm[1].n_features_in_
```

[10]:
```
11468
```

The total number of features is shown above: 11,468. It's all the unique words in the training data.

Next, I print a feature representation for the first document in the dev set.

[11]:
```
# Fit and transform the first document in the dev set.
X = vectorizer.fit_transform([dev_data["text"][0]])

feature_rep = pd.DataFrame({"feature": vectorizer.get_feature_names_out(),␣
 ↪"tf-idf": X.toarray()[0].tolist()})
feature_rep = feature_rep.sort_values(by = ['tf-idf'], ascending = False)
feature_rep = feature_rep.reset_index(drop=True)

# Print all rows.
print(feature_rep.to_string())
```

```
      feature    tf-idf
0         the  0.436436
1         and  0.436436
2        menu  0.218218
3         not  0.218218
4       their  0.145479
5        good  0.145479
6          to  0.145479
7          it  0.145479
8        have  0.145479
```

9	were	0.145479
10	china	0.145479
11	on	0.145479
12	original	0.145479
13	is	0.145479
14	really	0.072739
15	recent	0.072739
16	restaurant	0.072739
17	return	0.072739
18	she	0.072739
19	price	0.072739
20	portion	0.072739
21	site	0.072739
22	size	0.072739
23	so	0.072739
24	again	0.072739
25	steak	0.072739
26	tasty	0.072739
27	phone	0.072739
28	this	0.072739
29	told	0.072739
30	trying	0.072739
31	tso	0.072739
32	under	0.072739
33	wanted	0.072739
34	web	0.072739
35	what	0.072739
36	when	0.072739
37	years	0.072739
38	yet	0.072739
39	you	0.072739
40	place	0.072739
41	once	0.072739
42	pepper	0.072739
43	general	0.072739
44	average	0.072739
45	both	0.072739
46	business	0.072739
47	chef	0.072739
48	chicken	0.072739
49	deserves	0.072739
50	does	0.072739
51	enjoyed	0.072739
52	fan	0.072739
53	first	0.072739
54	food	0.072739
55	forward	0.072739
56	friendly	0.072739

```
57       grand  0.072739
58       other  0.072739
59       great  0.072739
60          if  0.072739
61          in  0.072739
62       items  0.072739
63        just  0.072739
64        lady  0.072739
65        look  0.072739
66  management  0.072739
67     matched  0.072739
68         new  0.072739
69          of  0.072739
70      online  0.072739
71      opened  0.072739
72        your  0.072739
```

Next, I print the docid and prediction (separted by a tab) for the first 10 documents in the dev file.

[12]:
```
X_test = dev_data["text"][0:10]

predictions = svm.predict(X_test)

for i in range(0, 10):
    print(dev_data["docid"][i] + "\t" + str(predictions[i]))
```

```
ZSJnW6faaNFQoqq4ALqYg   4
Rcbv11hm5AYEwZyqYwAvg   4
rkRTjhu5szaBggeFVcVJlA  4
dhmeDsQGUS1FXMLs49SWjQ  4
z9zfIMYmRRCE4ggfOIieEw  4
Xtb3pGSh39bqcozkBECw    2
DOUflAGzxLsXG6xOmR1w    2
ORxCEWURe08CTcZt95F4AQ  2
MzUg5twEcCyd0X6lBMP2Lg  2
uNlw2D5CYKk0wjNxLtYw    2
```

Finally, I make predictions for the dev and test partitions and write those to two files. The file names are dev_predictions_svm.tsv and test_predictions_svm.tsv.

[13]:
```
# Make predictions for the full dev data.
X_test = dev_data["text"]
predictions = svm.predict(X_test)

# Write predictions to a file.
predictions_df = pd.DataFrame({"docid": dev_data["docid"], "prediction":␣
 ↪predictions})
predictions_df.to_csv("dev_predictions_svm.tsv", sep = "\t", index = False)
```

```
# Make predictions for the test data.
X_test = test_data["text"]
predictions = svm.predict(X_test)

# Write predictions to a file.
predictions_df = pd.DataFrame({"docid": test_data["docid"], "prediction":␣
 ↪predictions})
predictions_df.to_csv("test_predictions_svm.tsv", sep = "\t", index = False)
```

Next, I evaluate the predictions, calculating precision, recall, and F1 for the positive class.

[14]:
```
# Read in the dev set predictions.
predictions_dev = pd.read_csv('dev_predictions_svm.tsv', sep = '\t')

# Join prediction DF with dev data.
combined_dev = pd.merge(dev_data, predictions_dev, on = "docid", how = "inner")

# Rename "stars" column as "actual".
combined_dev = combined_dev.rename(columns={"stars": "actual"})

# Filter data so it only includes observations from the positive class.
combined_dev_positive = combined_dev.loc[combined_dev["actual"] == 4]
combined_dev_negative = combined_dev.loc[combined_dev["actual"] == 2]

# Calculate precision.
# precision = true positives/(true positives + false positives)
n_true_positives = (combined_dev_positive["actual"] ==␣
 ↪combined_dev_positive["prediction"]).sum()
n_false_positives = (combined_dev_negative["actual"] !=␣
 ↪combined_dev_negative["prediction"]).sum()
precision = n_true_positives/(n_true_positives + n_false_positives)

# Calculate recall.
# recall = true positives/(true positives + false negatives)
n_false_negatives = (combined_dev_positive["actual"] !=␣
 ↪combined_dev_positive["prediction"]).sum()
recall = n_true_positives/(n_true_positives + n_false_negatives)

# Calculate F1.
f1 = (2 * precision * recall)/(precision + recall)

# Report results.
summary_stats_svm = pd.DataFrame({"statistic": ["precision", "recall", "f1"],␣
 ↪"value": [precision, recall, f1]})
summary_stats_svm
```

[14]:
```
   statistic     value
0  precision  0.847336
1     recall  0.827000
2         f1  0.837045
```

Next, I show some interesting mistakes the SVM model made.

[15]:
```
mistakes = combined_dev.loc[combined_dev["actual"] !=␣
 ↪combined_dev["prediction"]]
mistakes
```

[15]:
```
      actual                   docid  \
2          2  rkRTjhu5szaBggeFVcVJlA
7          4  ORxCEWURe08CTcZt95F4AQ
14         2  HqtEtHHDgSM0ctJHehaWaw
22         4  gjWVccNw6kB2UycZAEzyQg
23         4  JemvAfRVgCASkI3uG3Vffg
…        …                        …
1988       4  T1Lu6UeSHH6AZmzl9pnz7w
1992       2  ggr9TDU0bVnZBEsiObxYEQ
1993       2   b1kLsgl6nc2blZXRAGjXw
1998       4   jV8aDYrQ4LFHb0b0hC3Iw
1999       4  v6pqzYfd7XFM4t6fWI1wQA

                                                   text  prediction
2     Olive Garden used to be a favorite of the fami…           4
7     Our Las Vegas friend suggested duck tacos afte…           2
14    I've never been a big fan of BK, I have held p…           4
22    I hate to find out that there was another loca…           2
23    Been here several times. Hasn't let me down ye…           2
…                                                   …         …
1988  I had great service here today. I stopped in f…           2
1992  Firstly, letme agree with Stephanie T. This pl…           4
1993  Been there on several Friday nights and the fo…           4
1998  As soon as I walked in I was greeted by the sw…           2
1999  This place is gonna cost me about $2,000.  ---…           2

[322 rows x 4 columns]
```

Actual Class: Positive, Predicted Class: Negative

JemvAfRVgCASkI3uG3Vffg: Been here several times. Hasn't let me down yet. It's easily accessible off Boulder Highway and is open 24/7. This is one of my preferred spots to redeem the BOGO coupon for the Double Del Cheeseburger, $3.00 each, and I don't recall having any issues. They also aren't upselling jerks who pester you about trying a new item or asking 3 times whether you're sure that you've concluded your order.

v6pqzYfd7XFM4t6fWI1wQA: This place is gonna cost me about $2,000. —- Back story, I was backpacking around South America a few years ago and spent a few weeks in Ecuador. Loved it.

Great place. Of course I ate Ecuadorian food but don't remember much about it. Now after eating at MCMP I figure it's gonna be about $2,000 for me to go back to Ecuador and try the real stuff again so I can compare the two. Anyways, loved the oatmeal drink. It tasted like pineapple juice with a hint of spice or cinnamon. Chips and salsa were standard issue. The two dinners we had were very good. One was like a spicy carne asada, the other was sort of a combo plate with beef, beans, two types of plantains, sausage, egg, rice and chichron. Our entrees were fine, but I'm really excited to try the other items on the menu. Empanadas, tamales, and the chiviche looked really good. This place has about 6-8 tables that's it. Only one person to serve everyone so if the place is packed expect a few more minutes for everything.

Actual Class: Negative, Predicted Class: Positive

rkRTjhu5szaBggeFVcVJlA: Olive Garden used to be a favorite of the family, recently they cut back the menu extensively and many of our favorites are gone. I suggest checking the menu online before coming to see what's left.

ggr9TDU0bVnZBEsiObxYEQ: Firstly, letme agree with Stephanie T. This place is far from 'rapid' at any time of the day. Whether there is a huge queue forming out of the door or just you in the whole shop, you will wait for what seems like an eternity before you are eventually served. Although to give credit where credit is due, the actual preperation time of the sandwiches is fairly quick - which may explain why they always look a bit messy and sloppy. However, once you stop being shallow about your food, the sandwiches here are actually pretty damn tasty and good value for what you pay for them.

Like the earlier mistakes I showed for the first model (Multinomial NB), some of these examples have mixed language. The first review has the words "issues" and "upselling". It's not clear why the second example was mistakenly classified as negative. The word "but" appears in the review. Otherwise, it seems positive. The third and fourth reviews have positive words like "favorite", "quick", and "tasty", so it's easier to see why the SVM model misclassified them as positive.

I show the performance of both models below (summary stats).

[16]:
```
print("Multinomial NB")
print(summary_stats_multinomialnb)
print("")
print("SVM")
print(summary_stats_svm)
```

```
Multinomial NB
   statistic     value
0  precision  0.835095
1     recall  0.790000
2         f1  0.811922

SVM
   statistic     value
0  precision  0.847336
1     recall  0.827000
2         f1  0.837045
```

The SVM model performs better across all performance metrics than the Multinomial NB model.

1.5 e

I improve on the SVM model from part d by including word bigrams in the set of features. I do this by modifying the ngram_range argument in the TfidfVectorizer constructor. By default, the tokenizer only uses word unigrams. I set it to the tuple (1, 2), which means the minimum ngram the tokenizer extracts from the training data is a unigram, and the maximum ngram it extracts from the training data is a bigram. I train the model, make predictions on the dev set, and evaluate the predictions on the dev set (positive class only) alongside the two previous models.

[17]:
```
X_train = training_data['text']
y_train = training_data["stars"]

svm_bigram = make_pipeline(TfidfVectorizer(ngram_range = (1, 2)), SVC(kernel =␣
↪"linear"))
svm_bigram.fit(X_train, y_train)
svm_bigram[1].n_features_in_
```

[17]:
```
112436
```

[18]:
```
# Make predictions for the full dev data.
X_test = dev_data["text"]
predictions = svm_bigram.predict(X_test)

# Write predictions to a file.
predictions_df = pd.DataFrame({"docid": dev_data["docid"], "prediction":␣
↪predictions})
predictions_df.to_csv("dev_predictions_svm_bigram.tsv", sep = "\t", index =␣
↪False)

# Make predictions for the test data.
X_test = test_data["text"]
predictions = svm_bigram.predict(X_test)

# Write predictions to a file.
predictions_df = pd.DataFrame({"docid": test_data["docid"], "prediction":␣
↪predictions})
predictions_df.to_csv("test_predictions_svm_bigram.tsv", sep = "\t", index =␣
↪False)
```

[19]:
```
# Read in the dev set predictions.
predictions_dev = pd.read_csv('dev_predictions_svm_bigram.tsv', sep = '\t')

# Join prediction DF with dev data.
combined_dev = pd.merge(dev_data, predictions_dev, on = "docid", how = "inner")

# Rename "stars" column as "actual".
combined_dev = combined_dev.rename(columns={"stars": "actual"})
```

```
# Filter data so it only includes observations from the positive class.
combined_dev_positive = combined_dev.loc[combined_dev["actual"] == 4]
combined_dev_negative = combined_dev.loc[combined_dev["actual"] == 2]

# Calculate precision.
# precision = true positives/(true positives + false positives)
n_true_positives = (combined_dev_positive["actual"] ==␣
 ↪combined_dev_positive["prediction"]).sum()
n_false_positives = (combined_dev_negative["actual"] !=␣
 ↪combined_dev_negative["prediction"]).sum()
precision = n_true_positives/(n_true_positives + n_false_positives)

# Calculate recall.
# recall = true positives/(true positives + false negatives)
n_false_negatives = (combined_dev_positive["actual"] !=␣
 ↪combined_dev_positive["prediction"]).sum()
recall = n_true_positives/(n_true_positives + n_false_negatives)

# Calculate F1.
f1 = (2 * precision * recall)/(precision + recall)

# Report results.
summary_stats_svm_bigram = pd.DataFrame({"statistic": ["precision", "recall",␣
 ↪"f1"], "value": [precision, recall, f1]})
print("Multinomial NB")
print(summary_stats_multinomialnb)
print("")
print("SVM")
print(summary_stats_svm)
print("")
print("SVM Bigram")
print(summary_stats_svm_bigram)
```

```
Multinomial NB
   statistic     value
0  precision  0.835095
1     recall  0.790000
2         f1  0.811922

SVM
   statistic     value
0  precision  0.847336
1     recall  0.827000
2         f1  0.837045

SVM Bigram
   statistic     value
```

```
0  precision  0.870540
1     recall  0.854000
2         f1  0.862191
```

The modified SVM model that uses bigrams performs better than both of the previous models on all the evaluation metrics. It has higher precision, recall, and F1 scores when considering observations from the positive class.

Finally, I use the modified SVM model to make predictions on the test set, and I write the predictions to zproom1.tsv.

[20]:
```
# Make predictions for the full test data.
X_test = test_data["text"]
predictions = svm_bigram.predict(X_test)

# Write predictions to a file.
predictions_df = pd.DataFrame({"docid": dev_data["docid"], "prediction":␣
 ↪predictions})
predictions_df.to_csv("zproom1.tsv", sep = "\t", index = False)
```