Andrew Moshkovich
James Gu
Aaron Steinberg

Assignment 1

1 Problem: Naive Bayes

a)

P(terrible|y=-) = 4/22, P(epic|y=+) = 5/23, P(epic|y=-) = 4/22, P(boring|y=+) = 2/23, P(boring|y=-) = 5/22, P(terrible|y=+) = 1/23, P(terrible|y=-) = 4/22, P(disappointing|y=+) = 1/23, P(disappointing|y=+) = 6/22, P(great|y=+) = 6/23, P(amazing|y=+) = 8/23, P(great|y=-) = 2/22, P(amazing|y=-) = 1/22

$$log(\frac{0.5(\frac{2}{23}*\frac{8}{23}*\frac{1}{23}*\frac{1}{23})}{0.5(\frac{2}{23}*\frac{8}{23}*\frac{1}{23}*\frac{1}{23})+0.5(\frac{2}{22}*\frac{1}{22}*\frac{4}{22}*\frac{6}{22})})$$

$$P(+ \mid S) = -0.341$$

$$log(\frac{0.5(\frac{2}{22}*\frac{1}{22}*\frac{4}{22}*\frac{6}{22})}{0.5(\frac{2}{23}*\frac{8}{23}*\frac{1}{23}*\frac{1}{23})+0.5(\frac{2}{22}*\frac{1}{22}*\frac{4}{22}*\frac{6}{22})})$$

$$P(- \mid S) = -0.264$$

Result: Negative label

b)

1-smoothing

P(epic|y=+) = 6/29, P(epic|y=-) = 5/28, P(boring|y=+) = 3/29, P(boring|y=-) = 6/28, P(terrible|y=+) = 2/29, P(disappointing|y=+) = 2/29, P(terrible|y=-) = 5/28, P(disappointing|y=+) = 7/28, P(great|y=+) = 7/29, P(amazing|y=+) = 9/29, P(great|y=-) = 3/28, P(amazing|y=-) = 2/28

$$log(\frac{0.5(\frac{7}{29}*\frac{9}{29}*\frac{2}{29}*\frac{2}{29})}{0.5(\frac{7}{29}*\frac{9}{29}*\frac{2}{29}*\frac{2}{29})+0.5(\frac{3}{28}*\frac{2}{28}*\frac{5}{28}*\frac{7}{28})})$$

$$P(+ \mid S) = -0.292$$

$$log(\frac{0.5(\frac{3}{28}*\frac{2}{28}*\frac{5}{28}*\frac{7}{28})}{0.5(\frac{7}{29}*\frac{9}{29}*\frac{2}{29}*\frac{2}{29})+0.5(\frac{3}{28}*\frac{2}{28}*\frac{5}{28}*\frac{7}{28})})$$

$$P(- \mid S) = -0.310$$

Label is now Positive

c)

You could extract words like ‘not’ from the text, and if you used a bigram model, then words like ‘not disappointing’ would be considered positive

The word after not would be considered flipped

2 Programming: Hate speech detection

2.1 Naive Bayes

For our Naive Bayes model we did it step by step in a similar fashion to how we did it by hand. First, we calculated the probability of a word being positive and also a word being negative. Alongside this calculation, we made an array of tuples that held the probabilities of each word. Index 0 of each tuple being positive and index 1 of each tuple being negative. As we counted the number of negative and positive words we counted each frequency for each corresponding word. From these frequencies, we divided it by total positive or total negative words while also applying Laplace add-1 smoothing with alpha being 1 and V being len(X[0]).

Counting up the frequencies visualization example:

Sentence	1	2	…	n-1	n	Y
1	2	2	1	1	0	+
2	1	4	3	2	1	+
3	3	1	0	2	0	-
4	0	2	2	1	3	-

The next step to our implementation is actually inserting these values into the Naive Bayes formula. Usually the Naive Bayes formula has a denominator, but the goal of our implementation is to find which out of the positive and negative probabilities is greater, so if their denominators are the same, then we don’t need to use the denominator. So our calculation process only involved summing every log probability of each word along with adding the log probability of the overall positive or negative chance. So just by comparing the numerators of the negative and positive, our model predicts whether a tweet is hate speech or not based on which log probability is greater.

1)

Data Set	Accuracy
Train	0.9483
Test	0.7640
Dev	0.7200

2)

Number	Text	Actual Value	Dev Result	Explanation
5	It's good to see an educational institution that maintains its values and refuses to bow down in the name of `` progress . ''	0	1	Words like “refuses” and “institution” likely caused the classifier to wrongly mark it as hate speech.
10	I think I will post one , I placed my profile on Eurodate a few days ago .	0	0	There weren’t really any words that were deemed “hateful”, so the classifier marked it as non-hate speech.
15	I love my beer ! maybe you were drinking liquor I always get crazy on that so I gave it up and now everything is fine .	0	0	Words like “love” and “fine” likely caused the classifier to mark it as non-hate speech

3)

The 10 words that had the most distinctly hate speech words were:

‘shrinking’, ‘the’, ‘to’, ‘skyrocketing’, ‘,’ , ‘a’, ‘is’, ‘ground’, ‘town’, ‘)’

And the 10 words that had the least distinctly hate speech words were:

‘crime’, ‘hell’, ‘.’, ‘of’, ‘good,’ ‘it’, ‘21325’, ‘for’, ‘us’, ‘do’

We noticed that words that didn't appear often had a very skewed ratio, so words like '21325' would have a ratio of (nan:1), giving it a very low hate speech value. The same could be said for the words with high hate speech, with words like 'shrinking' and 'town' having a ratio of (1:nan).

2.2 Logistic Regression

1)

Data Set	Accuracy
Train	0.9200
Test	0.7760
Dev	0.7120

To implement logistic regression, we initialized the weights and bias. Then, over all the indices, until our gradient is smaller than our defined epsilon value (until we've determined that the model has converged), then we will keep updating the weights based on the (stochastic gradient descent formula, where the new weight = (previous weight + learning rate * target value - predicted value) * gradient (gradient is the change on the curve approaching the minimum change). When we have a low change that's less than epsilon then we know we've converged.

Compared to our Naive Bayes implementation, our Logistic Regression model had higher accuracy in the test data set while train and dev were slightly lower. These numbers are dependent on both our learning rate (alpha) and our epsilon value.

2)

Lambda	10	1	0.1	0.01	0.001	0.0001	0.00001
Train	0.5131	0.5117	0.5768	0.7006	0.7516	0.8719	0.9151
Test	0.5400	0.5400	0.5680	0.6720	0.6600	0.7880	0.7800
Dev	0.5200	0.5200	0.5600	0.6120	0.6160	0.6840	0.7200

From what we observed, smaller lambda values increased the training accuracy and dev accuracy, but as lambda got smaller and smaller, so did the amount the test accuracies changed by. The bigger the lambda value was, the closer our model was to becoming a coin flip.