Assignment 1
CSE 143: Intro to Natural Language Processing

Kristian Murray, William Siegmund, Jaisuraj Kaleeswaran, Vishwa Vijayasankar

April 28, 2024

1 Problem: Naive Bayes

(a) Compute the model's predicted scores for both positive and negative classes for sentence S (i.e. $P(+|S)$ and $P(-|S)$), and determine which label the model will apply to S.

S: *The film was great, the plot was simply amazing! Makes other superhero movies look terrible, this was not disappointing.*

Following the Formula:

$$P(w_i|c) = \frac{count(w_i, c)}{\sum_w \in V count(w, c)}$$

The predicted score for the positive class given sentence S is given by:

$$\log(P(+|S)) =$$

$$= \log(\frac{1/2 * 6/23 * 8/23 * 1/23 * 1/23}{1/2 * 6/23 * 8/23 * 1/23 * 1/23 + 1/2 * 2/22 * 1/22 * 4/22 * 6/22}) = -0.341$$

The predicted score for the negative class given sentence S is given by:

$$\log((P(-|S))) =$$

$$= \log(\frac{1/2 * 2/22 * 1/22 * 4/22 * 6/22}{1/2 * 2/22 * 1/22 * 4/22 * 6/22 + 1/2 * 6/23 * 8/23 * 1/23 * 1/23}) = -0.264$$

$P(-|S) > P(+|S)$; thus, the model would apply the **negative** label for sentence S

(b) Apply *add-1 smoothing* and recompute the Naive Bayes model's predicted scores for S. Did the predicted label change?

$$P(w_i|c) = \frac{count(w_i, c) + 1}{\sum_w \in V count(w, c) + |V|}$$

$P(\text{great}|+) = \frac{6+1}{23+6} = \frac{7}{29}$ $\quad$ $P(\text{great}|-) = \frac{2+1}{22+6} = \frac{3}{28}$

$P(\text{amazing}|+) = \frac{8+1}{23+6} = \frac{9}{29}$ $\quad$ $P(\text{amazing}|-) = \frac{1+1}{22+6} = \frac{2}{28}$

$P(\text{terrible}|+) = \frac{1+1}{23+6} = \frac{2}{29}$ $\quad$ $P(\text{terrible}|-) = \frac{4+1}{22+6} = \frac{5}{28}$

$P(\text{disappointing}|+) = \frac{1+1}{23+6} = \frac{2}{29}$ $\quad$ $P(\text{disappointing}|-) = \frac{6+1}{22+6} = \frac{7}{28}$

$$\log(P(+|S)) = \log(\frac{1/2 * 7/29 * 9/29 * 2/29 * 2/29}{1/2 * 7/29 * 9/29 * 2/29 * 2/29 + 1/2 * 3/28 * 2/28 * 5/28 * 7/28}) = -0.292$$

$$\log(P(-|S)) = \log(\frac{1/2 * 3/28 * 2/28 * 5/28 * 7/28}{1/2 * 7/29 * 9/29 * 2/29 * 2/29 + 1/2 * 3/28 * 2/28 * 5/28 * 7/28}) = -0.310$$

$P(+|S) > P(-|S)$; thus, the model would apply the **positive** label for sentence S

(c) What is an additional feature that you could extract from text to improve the classification of sentences like S(not necessarily in Naive Bayes), and how would it help improve the classification?

Instead of processing each keyword in a certain text, we can process neighbor words like 'not' or prefixes like 'un' and 'im' to get a better context on what the message of the text is. An example of where it may improve classification is in sentence S. If we look at "not disappointing" rather than just "disappointing" we may find that we shouldn't count it towards a negative label. This is because "not disappointing" associates more with "great" than it does with "disappointing".

2 Programming: Movie Review Sentiment Classification

2.1 Naive Bayes

Our Naive Bayes implementation closely followed the pseudocode on pg 6 Ch. 4 in the Jurafsky textbook. During training, we calculated the priors by summing each label (- and +) and then dividing by the total number of labels. For the likelihoods, we summed the words for each label in separate arrays; resulting in arrays that contained the total count of individual words for a given label. Then we summed all of the elements in each array to get the total count of words for each label. Finally, we adjusted the counts for Add-1 smoothing and divided by the size of the vocabulary plus the total word count for a given label. During prediction, our model retrieves the priors and likelihoods calculated during training for both labels (0 and 1) and assigns the label that has the larger value.

1. Accuracy on the training, dev, and test sets

Training set	Test Set	Train Accuracy	Test Accuracy
train.csv	test.csv	1368 / 1400 = 0.9771	197 / 250 = 0.7880
train.csv	dev.csv	1368 / 1400 = 0.9771	204 / 250 = 0.8160
test.csv	train.csv	249 / 250 = 0.9960	1043 / 1400 = 0.7450
train.csv	train.csv	1368 / 1400 = 0.9771	1368 / 1400 = 0.9771

We primarily trained on the "train.csv"; nevertheless, we felt inclined to experiment what would happen if we trained on a smaller data set such as "test.csv" and tested on a larger set. Our expectations were met that the model would perform worse when trained on less data.

2. Look at the output of your model on some randomly selected examples in the dev set. What do you observe? Why do you think the model has predicted the way it has for those specific examples?
We randomly chose 4 examples from the dev set to test. We noticed that the accuracy of the test was 75%; thus, we decided to try and understand why our model might have gotten it wrong. The following sentence was marked as negative by our model; however, the true label is positive

> "This film was just absolutly brilliant. It actually made me think. During the whole movie I was confused as hell. I loved everything about it...it was just so confusing and so twisted and weird, it was hard not to love it. All of the actors were phenominal, and no one could have done a better job...This is one of my favorites of the year...it deserves an oscar."

We figure that while there is a good amount of uncommon positive words, such as 'brilliant', 'phenomenal', and 'love', that will appear less often than more commonly-used negative words in the vocabulary. Because of this, the uncommon positive words could have been overtaken by the more-likely common negative words such has "no", "not", and "confusing" - having less influence in the overall classification.

3. List 10 words that, under your model, have the highest ratio of $P(W|1)/P(W|0)$(the most distinctly positive words) and list the 10 words with the lowest ratio. What trends do you see?

   Highest ratio words: mathieu, vance, elvira, kolchack, bettie, awards, beetle, ballet, polanski, carell
   Lowest ratio words: rambo, hire, poorly, insult, prom, gram, boll, freddy, below, franco
   NOTE: Highest ratio on the left, lowest on right

   We noticed that the lowest ratio words have contained words with negative connotations such as "poorly" and "insult"; whereas, the highest ratio words contained positive words such as "awards". We noticed many names in both portions, perhaps they describe actors/characters that were well received movies or poorly rated films.

2.2 Logistic Regression

Our Logistic Regression implementation uses the Stochastic gradient descent algorithm described in class; thus, we update our weights after every data point (review, true label). We started our weights W set to 0. then treated the word counts provided in X as features. We made a helper sigmoid method that takes in a parameter z. We used the sigmoid function with parameter $\mathbf{W} \cdot X + b$. After every data point we would update our weights using the aforementioned parameter for the sigmoid function. This process occurs for all reviews for a specified amount of iterations. After some experimentation, we found 50 iterations to train and test our data well. Additionally, through the same process we set the learning rate to 0.1. Our model performs predictions by dotting a given set of features with the weights from training and adding the updated bias into our sigmoid method. A label is then assigned by comparing the result with the threshold of 0.5 where anything greater than 0.5 is given 1 else 0.

1. Report the accuracy without L2 regularization on the train, dev, and test sets. How does it compare to Naive Bayes?
   The Following results are after performing 50 iterations and a learning rate of 0.1

Training set	Test Set	Train Accuracy	Test Accuracy
train.csv	test.csv	1400 / 1400 = 1.000	197 / 250 = 0.7880
train.csv	dev.csv	1400 / 1400 = 1.000	205 / 250 = 0.8200
test.csv	train.csv	154 / 250 = 0.6160	733 / 1400 = 0.5236
train.csv	train.csv	1400 / 1400 = 1.000	1400 / 1400 = 1.000

   It appears that this model has a higher train accuracy but requires abundant training compared to Naive Bayes to update weights. Furthermore, Logistic Regression has more parameters to "play" with such as the learning rate and iterations. Additionally, without regularization, this model could be overfitting which may explain why it's higher train accuracy makes little to no increase in test accuracy compared to Bayes.

2. Add L2 regularization with different weights, such as λ = 0.0001, 0.001, 0.01, 0.1, 1, 10. In your write-up, describe what you observed.
   We noticed that our Logistic Regression slightly improved as we increased λ and then started to degrade our model as it became "too large." We figured this behavior aligns with regularization as larger λ seems to cause under-fitting; whereas, smaller λ was causing over-fitting. Overall, there wasn't much improvement using L2 regularisation but noticeable deterioration as λ increased.

λ	Train Accuracy	Test Accuracy
0.0001	1400 / 1400 = 1.000	195 / 250 = 0.7800
0.001	1400 / 1400 = 1.0000	197 / 250 = 0.7880
0.01	1400 / 1400 = 1.0000	198 / 250 = 0.7920
0.1	1317 / 1400 = 0.9407	179 / 250 = 0.7170
1	1870 / 1400 = 0.6214	138 / 250 = 0.5520
10	875 / 1400 = 0.6250	153 / 250 = 0.6120

Model parameters: $\alpha = 0.05$ *iterations* $= 100$.

3. Remove stopwords (e.g. a, the, in)
   CustomFeature() in *utils.py* was altered to remove stopwords during feature extraction. We decided to look for a list of stop words:
   `https://www.geeksforgeeks.org/removing-stop-words-nltk-python/` and removed them from the vocabulary. First We used the NaivesBaye classifier with this different list of features. Then we tried it with the LogisticRegression classifier. We also experimented with what would happen if we only used stop words as features.

Ignore Stop Words?	classifier	Train Accuracy	Test Accuracy
yes	NaiveBayes	1382 / 1400 = 0.9871	199 / 250 = 0.7960
no	NaiveBayes	929 / 1400 = 0.6636	159 / 250 = 0.6360
yes	Logistic Regression	1400/ 1400 = 1.0000	193 / 250 = 0.7720
no	Logistic Regression	830 / 1400 = 0.5929	153 / 250 = 0.6120

   Model parameters: $\alpha = 0.05$ *iterations* $= 100$.

   We noticed that our NaiveBayes classifier did slightly better when we removed stop words from the vocabulary. On the other hand, we saw a slight decrease in our test accuracy for our LogisticRegression model. We figure we would have to adjust some of our LogisticRegression parameters to better suit the change in features. To use this feature you need add the "-f customized" flag and choose any model that you want to run!