CSE143: Assignment 1 Writeup

Daniel Carrera, Jeremy Lin, Sanim Muktadir, Junchen Wang

May 4, 2022

1 Problem: Naive Bayes

(a) Our vocabulary has 45 word counts in total with 23 labeled positive and 22 labeled negative. The given sentence S contains four words from our vocabulary with one count in S for each. We then calculated the likelihood probability for those features using:

$$\frac{c(X_i|+/-)}{c(Total+/-Instances)}$$

And the prior probabilities using:

$$\frac{c(Total+/-Instances)}{c(TotalInstances)}$$

x_i	$Pr(x_i\|y=+)$	$Pr(x_i\|y=-)$	$\Pr(y=+)$	$\Pr(y=-)$
great	$\frac{6}{23}$	$\frac{2}{22}$	$\frac{23}{45}$	$\frac{22}{45}$
amazing	$\frac{8}{23}$	$\frac{1}{22}$		
terrible	$\frac{1}{23}$	$\frac{4}{22}$		
disappointing	$\frac{1}{23}$	$\frac{6}{22}$		

Table 1: Table of Likelihood and Prior Probabilities.

These figures were then used in the natural log form of the Naive Bayes classifier we derived using log rules to determine the likelihood of a positive and negative classification given sentence S.

$$\Pr(y=+|x_i) = e^{\ln(A)-\ln(e^{(\ln(A)}+e^{\ln(B)})}$$

$$Where \qquad \ln(A) = \ln(\Pr(y=+)) + \sum_i \ln(\Pr(x_i|y=+))$$

$$\ln(B) = \ln(\Pr(y=-)) + \sum_i \ln(\Pr(x_i|y=-))$$

After calculations, we obtained the resulting log probability for positive and negative classification. We then exponentiated that result in order to obtain the actual probabilities.

$$P(+|S) = e^{-0.76204} = 0.467 = 46.7\%$$
$$P(-|S) = e^{-0.62869} = 0.533 = 53.3\%$$

The Naive Bayes classifier would label the sentence S as negative because the probabilities tell the model that a negative classification is more likely.

(b) Additive Smoothing

x_i	$Pr(x_i \mid y=+)$	$Pr(x_i \mid y=-)$	$\Pr(y=+)$	$\Pr(y=-)$
great	$\frac{7}{29}$	$\frac{3}{28}$	$\frac{23}{45}$	$\frac{22}{45}$
amazing	$\frac{9}{29}$	$\frac{2}{28}$		
terrible	$\frac{2}{29}$	$\frac{5}{28}$		
disappointing	$\frac{2}{29}$	$\frac{7}{28}$		

Table 2: Table of Likelihood and Prior Probabilities with $\alpha = 1$ smoothing and |V| = 6.

To apply add-1 smoothing, we used the following formula with an α = 1 for every likelihood probability used in our Naive Bayes formula:

$$\frac{oldNumerator + \alpha}{oldDenominator + \alpha|V|}$$

Using these smoothed figures with the natural log form of the Naive Bayes classifier results in these probabilities:

$$P(+|S) = e^{-0.65087} = 0.521 = 52.1\%$$
$$P(-|S) = e^{-0.73728} = 0.478 = 47.8\%$$

The Naive Bayes classifier now labels the sentence S as positive.

(c) Future Improvements

One feature that could be added is the ability to recognize bigrams, especially ones such as “not disappointing” where the “not” negates the negative sentiment. Another useful feature would be to recognize n-grams for words like “other” to reduce the negative classifications of phrases like “makes other superhero movies look terrible”. These features would help improve classification because we start to look at the entire text as a whole, and not just the identities of individual words. Most importantly though would be an increase in data size to develop a fuller corpus.

2 Programming: Hate Speech Detection Using Naive Bayes

Our Naive Bayes model is based upon the natural log form of the classifier we showed previously. Our fit() function splits the training data into a positive and negative vector corresponding to their true label. The vectors hold the counts of each feature throughout all of the training data. This is then used in our predict() function, where we calculate the smoothed and logged positive and negative terms of the Naive Bayes equation, i.e. $\ln(A)$ and $\ln(B)$ respectively. The higher term determines the label of each document. Initially, we split and counted the data within the predict() function and also calculated the denominator of Naive Bayes, resulting in the training of test data and high run times of over five minutes. After fitting correctly and only comparing the positive and negative numerators, our model runs in under ten seconds.

2.1 Accuracy

$$\text{Train set: } \frac{1294}{1413} = 91.58\% \qquad \text{Test set: } \frac{183}{250} = 73.20\% \qquad \text{Dev set: } \frac{173}{250} = 69.20\%$$

We see a high accuracy on our training set and significantly lower ones for the test and dev sets. Although we are smoothing the data with additive smoothing, there is still overfitting to the training set. The dev set is particularly low, with its accuracy almost 22.38% lower than the training data and 4.00% lower than the test set.

2.2 Output Analysis: Random Documents

Sentence	Predicted Label	True Label
I'm happy for the two of you .	0	0
Well I wonder if its started messing with the daughter yet .	1	1
a nice kick to the nuts will always work I find it a dirty move to kick another man in the knackers but after all .. a black does n't equate to the same standard	1	1
When she goes outside , the subhuman ape thing is usually out there .	1	1
The UAF are just a bunch of middle class faggots , the EDL should steam into them every time they show up to counter a demonstration , they 'll soon run home to daddy .	1	1

The random sentences were selected by generating 5 random numbers between 0 and 250 (the size of the dev set). If a sentence index happened to be one of the random numbers, the code would store the sentence and its predicted value. After evaluating the output of the five stored sentences and predictions, we concluded that all the sentences had been correctly labeled by the Naive Bayes model.

The sentences labeled hate speech had multiple hate speech words, while the sentence labeled non hate speech had none. We believe the model predicted the sentences this way by heavily weighting the hate speech words compared to non-hate speech words, meaning the count of those hate speech words in documents labeled hate speech was very high and vice versa.

2.3 Output Analysis: Highest/Lowest Ratio Words

Top ten most likely hate speed words in descending order:

"jews" "asian" "liberal" "non" "aids" "apes" "dumb" "non-white" "jew" "scum"

Top ten least likely hate speech words in descending order:

"check" "thanks" "www" "_" "sf" "father" "sports" "information" "com" "k"

Most of the words with the highest $P(w|1)/P(w|0)$ ratio were racial in nature and are most likely the groups targeted in the hate speech documents, leading to their high likelihood in being classified as hate speech. In contrast, most of the words with the lowest ratio appear to be filler or neutral words such as the start and end of a web address, "www" and "com". There were no words in either of the lists that we believed were incorrectly labeled. The trend we observed was that as the $P(w|1)/P(w|0)$ ratio increases, the likelihood of seeing hate speech words also increases.

3 Programming: Hate Speech Detection Using Logistic Regression

Our model uses Binary Logistic Regression as well as Stochastic Gradient Descent to classify the sentences, both of which we found in the lecture slides. Our fit() function runs for the number of epochs we manually set it to go through before converging. It determines the most optimal beta value for each feature through gradient descent in order to maximize the regression. In predict(), it uses the optimized weight in a final regression to classify the sentences. L2 Regularization was provided during section and is achieved through the subtraction of $2\lambda\beta_t$ during the calculation of β_{t+1}.

We found that a max epoch of 20 and learning rate of α = 0.1 allowed the best balance between run time and minimal loss with a run time of 50 seconds as well as a total loss of 114.36. We could improve the loss with a smaller α of 0.01 and more epochs, but we decided that our code was not efficient enough for the increased number of epochs.

3.1 Accuracy Without L2 Regularization

# of epochs = 20, α = 0.1

$$\text{Train set: } \frac{1392}{1413} = 98.51\% \qquad \text{Test set: } \frac{181}{250} = 72.40\% \qquad \text{Dev set: } \frac{173}{250} = 69.20\%$$

Our training accuracy is significantly higher than Naive Bayes by 6.93%, but test accuracy is marginally lower by 0.80% and dev accuracy is the same. We believe we are overfitting in comparison to the additively smoothed model of Naive Bayes. We are not sure if we can see the benefits of weights using our Logistic Regression model over the assumed independence of features in Naive Bayes, but this could be an issue with the size of our data set. A larger data set may yield different results and lead to new insights.

3.2 L2 Regularization: Testing Lambda Values

λ	TrainAcc	TestAcc	TotalLoss	Final β_1	Notes
0	98.51%	72.40%	114.36	-0.315	β_1 = weight of first feature
0.0001	89.03%	74.00%	384.84	0.313	
0.001	70.13%	62.80%	617.13	0.404	
0.01	62.14%	60.00%	674.69	0.084	Only visible change was TotalLoss by $3 * 10^{-13}$
0.1	57.82%	58.40%	701.00	0.000671	No visible changes after first epoch
1	49.75%	52.80%	740.24	121.915	Overflow in first epoch after exp() in regression
10	50.53%	47.20%	0.00	NaN	Overflow in first epoch with all beta calculations

Tested on test.csv # of epochs = 20 α = 0.1 TotalLoss is calculated as $\sum_n |Y_i - \hat{P}(X_i)|$

After a certain point, as the lambda values went up, the training and testing accuracy began to decrease. We believe it is due to underfitting of the data from the higher lambda values resulting in a smaller weight overall and thereby not maximizing the regression. We see this trend of the β values moving closer towards 0 until $\lambda = 0.1$ where an overflow error occurs. We predict it is due to the β values becoming so small that they wrap around and result in an overflow. Total loss also trends upward with a greater λ.

$\lambda = 0.0001$ yielded the best results with only a 15.03% difference between the training and test accuracies compared to the previous 26.11% difference with no L2 regularization, showing the benefits in reducing overfitting with this technique. The accuracies were also very similar to those found within our smoothed Naive Bayes model.