CSE143 Assignment 1 Report

Sanjay Shrikanth, Matthew Daxner, Collin McColl

May 3, 2022

1 Naive Bayes: Hand Calculations

(a) $log(P(+)) = log(P(-)) = log(0.5) = -0.69314$
$log(P(`great' \mid +)) = log(6/23) = -1.3437$
$log(P(`great' \mid -)) = log(2/22) = -2.3978$
$log(P(`amazing' \mid +)) = log(8/23) = -1.0560$
$log(P(`amazing' \mid -)) = log(1/22) = -3.091$
$log(P(`terrible' \mid +)) = log(1/23) = -3.1354$
$log(P(`terrible' \mid -)) = log(4/22) = -1.704$
$log(P(`disappointing' \mid +)) = log(1/23) = -3.1354$
$log(P(`disappointing' \mid -)) = log(6/22) = -1.2992$

$$log(P(y = + \mid S)) \propto logP(+) + \sum_i log(P(S_i \mid +)) = -9.3639$$

$$log(P(y = - \mid S)) \propto logP(-) + \sum_i log(P(S_i \mid -)) = -9.1861$$

Since $-9.186 > -9.364$, the model would classify S as negative $(-)$

(b) With +1-smoothing: $P(+) = P(-) = 0.5$
$log(P(+)) = log(P(-)) = log(0.5) = -0.69314$
$log(P(`great' \mid +)) = log(6 + 1/23 + 6) = -1.4213$
$log(P(`great' \mid -)) = log(2 + 1/22 + 6) = -2.2335$
$log(P(`amazing' \mid +)) = log(8 + 1/23 + 6) = -1.17$
$log(P(`amazing' \mid -)) = log(1 + 1/22 + 6) = -2.639$
$log(P(`terrible' \mid +)) = log(1 + 1/23 + 6) = -2,674$
$log(P(`terrible' \mid -)) = log(4 + 1/22 + 6) = -1.7227$
$log(P(`disappointing' \mid +)) = log(1 + 1/23 + 6) = -2.674$
$log(P(`disappointing' \mid -)) = log(6 + 1/22 + 6) = -1.3862$

$$log(P(y = + \mid S)) \propto logP(+) + \sum_i log(P(S_i \mid +)) = -8.6329$$

$$log(P(y = - \mid S)) \propto logP(-) + \sum_i log(P(S_i \mid -)) = -8.6748$$

With add-1 smoothing, the classifier now labels S as positive (+)

(c) Additional Feature

An additional feature that might be worth extracting in order to increase accuracy is tracking the negation of phrase like "not disappointing". When a word is negated, it is meant to say the opposite thing, the negation of a positive word should be some negative("not good"), adding this feature would help catch these cases.

2 Hate Speech Detection

2.1 Naive Bayes

Model Description: We implemented a simple Naive Bayes model with Add-1 (Laplace) smoothing to improve the model's performance on the dataset and mitigate any zero word probabilities. Following the equation

$$P(x_i \mid y) = \frac{n_{i,y} + \alpha}{n_y + V\alpha}$$

we extracted the count of the hate and non hate words to get the conditional probabilities of each word in the training set. Then, to classify the test set, we used the logarithm Naive Bayes formula. If the sum of the log-prior and the log-probability of the sentence for one class was greater than that of the other, then the `predict` function would pick the class with the highest probability.

Experimental Procedure: We first implemented the Naive Bayes classifer without Laplace smoothing. Initially, we were having divide-by-zero errors, leading to some probabilities being omitted and lowering our model's predictive capabilities. After implementing smoothing, our test set results shot from 53% to 72.8%, which was a significant boost.

1. Accuracies for the given datasets:

Training	93.63%
Dev	70.40%
Test	74.00%

2. Looking at random examples in the dev set, we observed that often, sentences that involve negative diction or common victims of hate speech were correctly classified. For the predictions that differed from the proper label, it was mainly due to the occurrence of negative words

like 'anti' or certain race words like 'white' that our classifier deemed them as hate speech even though the overall context of the statement proved otherwise. This is the main pitfall of Naive Bayes, as the classifier cannot take sentence context into consideration especially when the context clues are far apart.

3. The 10 words with the highest and lowest ratio of $P(w \mid 1)/P(w \mid 0)$ are:

Highest Ratio	Lowest Ratio
Jews	check
non	thanks
leave	information
mud	sports
liberal	sf
non-white	15
dumb	www
asian	_
apes	[
filth	10.00

Looking at both lists, we notice that the words with the highest ratios are indeed words that would likely appear in hate speech, as most of them are insults that are used to demean a group of people. Also, the highest ratio words involve races that are also often subject to hate speech. We observed that for the lowest ratio words, they are either neutral words, arbitrary characters or numbers which are all 'words' that would not occur in hate speech.

2.2 Logistic Regression

Model Description: Our model simulates the Binary Logistic Regression classifier, which is given by the equation

$$\sigma(x_i) = P(y = 1 \mid x_i, \beta) = \frac{1}{1 + \exp(-\sum_{i=1}^{F} x_i \beta_i)} \tag{1}$$

where x is a feature vector and β is a learned weights vector that is applied to the feature vector as a linear combination. For our model, we defined x to be a simple, fixed sized vector that has the same size as the vocabulary of the data, where each position represents a unique word occurring in the sentence. For x, if a word k exists in sentence x_i at least once, then k's entry in x_i is 1, otherwise 0. For the `predict` function, our model converts each sentence to a feature vector and feeds it into the logistic regression equation to get $P(y = 1 \mid x_i, \beta)$. If the equation outputs 0.5 or greater, our model outputs 1, otherwise 0.

Experimental Procedure: We first implemented our feature transformation, changing our input sentences to a feature vectors of 1s and 0s. To get β, we implemented Stochastic Gradient Descent, which aims to derive a weights vector that accurately predicts the hate classification of the sentence. Regarding convergence, we looked at the equation

$$\beta_{t+1} = \beta_t + (y - \sigma(x_i))\alpha x_i \tag{2}$$

where $y - \sigma(x_i)$ is the approximate gradient of the sigmoid function. We initially implemented a threshold which tracks $\beta_{t+1} - \beta_t$ and exits the algorithm when its small enough. However, doing so took too much time and outputted poor prediction results, indicating possible overfitting. We then looked at tweaking the number of iterations, and α, which are all hyper-parameters. We found that with 100 epochs and $\alpha = 0.005$, we got the best prediction results on the test set. For our implementation of L2 Regularization, we added $2 \cdot \lambda \cdot \beta_t$ tos each update of β. *Note:* We kept α and the epoch number constant through all tests of different values for λ.

1. Accuracy without L2 Regularization:

Training	97.95%
Dev	73.20%
Test	79.60%

   Our Logistic Regression model performs significantly better on the training set and much better on both the dev and test set, with around $6 - 7\%$ increase on all datasets.

2. Accuracies with L2 Regularization

λ	Training	Dev	Test
0.0001	51.38%	49.20%	47.60%
0.001	50.53%	49.60%	47.20%
0.01	54.71%	56.00%	55.60%
0.1	56.83%	61.60%	60.00%
1	48.05%	47.20%	46.40%
10	50.53%	49.60%	47.20%

   It appears that with L2 Regularization, the model was less accurate on all fronts. While a lambda value of 0.1 was the most optimal, the training, dev, and test accuracies were still lower than those observed before regularization was added. However, regularization did seem accomplish its task of preventing over-fitting and shrinking the gap between training and test accuracies. It is possible that regularization requires a larger data set in order to be useful to our model.