A1 Project Report

Jiancheng Xiong, Atharva Tawde, Swayam Shah, Karthik Chaparala

October 2024

1 Problem: Naive Bayes Classifier

We have the data below which displays the occurrence of the words in a particular movie review and whether that review is positive or negative. The data is as follows:

Review	great	amazing	epic	boring	terrible	disappointing	Y
1	2	2	2	1	1	0	+
2	1	4	0	0	0	1	+
3	3	2	3	1	1	0	+
4	0	1	1	2	1	2	-
5	1	0	2	1	2	2	-
6	1	0	1	2	1	2	-

Table 1: Sample Dataset given

The table above defines the data as the table where each row is associated with a review and the columns tell us the number of occurrences of a unique word in that review. Column Y tells us if the review is positive or negative. Let's use the above data to create a simple classifier that classifies a review as positive or negative based on different words' occurrence and frequency.

1.1 Initial Classifier

Let's create a Naive Bayes classifier trained on the data above and use it on a test sentence to analyze the accuracy of our classifier. To train our Naive Bayes classifier, we generate the conditional probabilities of a review being a positive one given the presence of a word. Now let's compute the classification for the sentence given below:

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

This review contains 1 "great", 1 "amazing", 1 "terrible", 0 "epic", 0 "boring", and 1 "disappointing" We can create a new dataset to represent the data from sentence S: [1, 1, 1, 0, 0, 1].

The log-probability distribution of a review being positive for different words can be computed as below:

1. $P(word|positive) = \frac{\text{number of occurrences of this word in positive reviews}}{\text{total number of words appearing in positive reviews}}$
2. $P(positive|word) = \frac{P(word|positive) \cdot P(positive)}{P(word)}$
3. $lprob = log[P(positive|word)]$

The negative log probabilities can be found using the same formula, swapping positive for negative. Using the formula above, we can calculate the following likelihood table of the keywords (using log-probability, indicated by lP()):

Keyword	lP(*Positive* Given Keyword)	lP(*Negative* Given Keyword)
great	-0.446	-1.966
amazing	-0.032	-3.158
epic	-0.882	-1.136
boring	-1.836	-0.454
terrible	-2.370	-0.288
disappointing	-3.836	-0.193

Table 2: Log probabilities for each word in determining if the review is positive or negative

In sentence S, we have the dataset [1,1,0,0,1,1]. Using the likelihood table, we can calculate the predicted scores for sentence S.

- $score(+) = 1(-0.446) + 1(-0.032) + 0(-0.882) + 0(-1.836) + 1(-2.370) + 1(-3.836) = -6.684$
- $score(-) = 1(-1.966) + 1(-3.158) + 0(1.136) + 0(-0.454) + 1(-0.288) + 1(-0.193) = -5.605$

As score(-) > score(+), the classifier will classify S as **negative**.

1.2 Smoothed Classifier

Because the initial data we have is sparse, this may lead to the classifier's overfitting. To counter this, we can apply add-1 smoothing to the data by adding 1 to each occurrence in each review.

The smoothed data is as follows:

Review	great	amazing	epic	boring	terrible	disappointing	Y
1	3	3	3	2	2	1	+
2	2	5	1	1	1	2	+
3	4	3	4	2	2	1	+
4	1	2	2	3	2	3	-
5	2	1	3	2	3	3	-
6	2	1	2	3	2	3	-

Table 3: Smoothed Sample Dataset

We can then use the same formula to compute the smoothed log-likelihood table of the keywords:

Keyword	*Positive* Given Keyword	*Negative* Given Keyword
great	-0.653	-1.47
amazing	-0.466	-1.889
epic	-0.924	-1.059
boring	-1.3884	-0.6827
terrible	-1.474	-0.63487
disappointing	-1.7131	-0.51251

Table 4: Smoothed log probabilities for each word in determining if the review is positive or negative

Using the *smoothed* likelihood table and the previously determined dataset for sentence S, we can calculate the new predicted scores for sentence S.

- $score(+) = 1(-0.653) + 1(-0.466) + 0(-0.924) + 0(-1.3884) + 1(-1.474) + 1(-1.7131) = -4.306$
- $score(-) = 1(-1.47) + 1(-1.889) + 0(-1.059) + 0(-0.6827) + 1(-0.63487) + 1(-0.51251) = -4.506$

As score(+) > score(-), the classifier will now classify S as **positive**. With smoothing, the predicted label changed from the original classification.

1.3 Additional Features

This classifier is far from perfect—many features are left out that could impact the classification. One additional feature that could be implemented is a way to extract information from words near the "keywords" to give more context to them. For example, currently, the sentence "not great" would be seen as positive, while if we have the context of the previous word/words, we could see that this is a negative phrase.

2 Programming: Movie Review Sentiment Classification

2.1 Naive Bayes

Our Naive Bayes model aimed to improve the binary sentiment classification that AlwaysPredictZero tried to achieve (miserably). By implementing our novel NaiveBayesClassifier, we upgraded our accuracies: a training set accuracy of 79.57%, a dev set accuracy of 76.40%, and a test set accuracy of 72.00%.

Upon analyzing randomly selected samples from the dev set, it was observed that certain words had significantly influenced the model's predictions. Words that appeared frequently and had strong positive or negative log probs swayed the overall sentiment.

	Highest Ratio		**Lowest Ratio**
1	Great	1	?
2	And	2	Bad
3	Love	3	Movie
4	Her	4	Worst
5	As	5	No
6	Also	6	They
7	His	7	Or
8	Very	8	Even
9	She	9	Don't
10	Is	10	Why

Table 5: Highest and Lowest Ratio Words

These lists reflect that distinctly positive words, like "great" and "love" correlate strongly with positive sentiment, while negative words, such as "bad" and "worst", align with negative sentiments. Interestingly, we can see that neutral and common words such as "and" and "is" also appear in the positive sentiment list, meaning their contextual use may impact classification!

2.2 Logistic Regression

Some of the main problems with the Naive Bayes classifier are resolved by the logistic regression model. Naive Bayes's sensitivity to word frequency; many instances of a word that might disproportionately affect the classification, is one of its main drawbacks. By more accurately weighing features, logistic regression lessens this problem and produces more balanced judgments.

Furthermore, real-world data rarely meets the Naive Bayes classifier's assumption that features—like words in a text—are conditionally independent. In contrast, logistic regression is a more adaptable model for classification tasks because it does not rely on this independence assumption. On applying the Logistic

regression classifier, the accuracy of the training set increased substantially but further advancements were small and fine-tuned by choosing the correct learning rat and the number of times to train on the training set.

The accuracy of the Logistic regression classifier without L2 regularization is **99.94%** on the Training set, 100% on the development set, and 86% on the testing set. This is substantially better than the naive Bayes classifier on both the training set and the development set. The main reason for such an observation could be supported by the fact that logistic regression does not completely rely on equal weights for each word but changes the weights with each iteration over the training set.

To further increase the accuracy on the testing set, L2 Regularization was applied to increase the accuracy with an increase in the L2_lambda but then the accuracy dropped significantly as the L2_lambda got bigger. The observations are as follows.

Accuracies	0.0001	0.001	0.01	0.1	1	10
Training	99.94%	99.94%	67.33%	53.39%	50%	50%
Dev	100%	100%	70.61%	60%	70%	50%
Testing	86%	86.40%	60%	52%	50%	50%

Table 6: Train, dev, and test set accuracy with different regularizations

2.3 Bonus Classifier

For our bonus classifier, we decided to improve the accuracy of our NaiveBayesClassifier. To do so, we aimed to eliminate stop words from our input tokens, in an attempt to make the model focus more on words that provided actual context about the movies, rather than the stop words that contributed little substance. Our original naive Bayes classifier's top 10 highest and lowest positive correlates can be seen in Table 3 above.

In this table, we can see that a notable proportion of these terms consisted of stop words; 8/10 for the highest ratios and 6/10 for the lowest. Given that stop words provide very little context on their own their inclusion in our dataset has resulted in results that are muddled and meaningless.

To increase our accuracy and improve our results, we first compiled a comprehensive list of stop words in a list. During our tokenization process, we ran a check that only added words that were not on our list, meaning our data now comprised of only non-stop words. Afterward, our highest and lowest ratio words were:

	Highest Ratio		**Lowest Ratio**
1	Great	1	Acting
2	Love	2	Awful
3	Young	3	Nothing
4	Well	4	...
5	Best	5	Just
6	Always	6	Even
7	Excellent	7	Worst
8	Film	8	Movie
9	World	9	Bad
10	Wonderful	10	?

Table 7: Highest and Lowest Ratio Words

The removal of stop words from our data has allowed for more informative and meaningful words to populate our highest and lowest ratios, significantly impacting our accuracies; training accuracy is up 4.36%, from 79.57% to 83.93%, and test accuracy is up 4.8%, from 72% to 76.80%.