CSE 143 Assignment 1

Kevin Bachelor
Sabrina Fogel
William He
Will Vartanian

April 28 2024

1 Problem: Naive Bayes

1.1 Naive Bayes, *No Smoothing*

In order to implement a Naive Bayes classifier by hand, we began by looking at the Bayes rule for our scenario:

$$P(+|S) = \frac{P(S|+)P(+)}{P(S)}$$

Our sentence S contains the following tokens: {great, amazing, terrible, disappointing}, so those are the tokens we will be examining probabilities for.
Out training dataset D contains 3 positive and 3 negative reviews, making for a total of 6 reviews. Thus, we have:

$$P(+) = 3/6 = 0.5,\ P(-) = 3/6 = 0.5$$

In order to calculate $P(+|S)$, we need to split $P(S|+)$ into all of the relevant tokens contained within S. This means that our above Bayes rule expands into:

$$P(+|S) = P(\{\text{tokens}\}|+)P(+)$$

The above formula is proportional to:

$$P(+|S) = \log(P(\{\text{tokens}\}|+))P(+)$$

Looking at Table 1 for pre-variable probability information for the extracted tokens, we can calculate that:

$$P(+|S) = 0.5 * \log(6/23 * 8/23 * 1/23 * 1/23) = -1.88$$

$$P(-|S) = 0.5 * \log(2/22 * 1/22 * 4/22 * 6/22) = -1.84$$

G	A	E	B	T	D	Y	Total
6	8	5	2	1	1	+	23
2	1	4	5	4	6	-	22

Table 1: Pre-Variable Probability for Naive Bayes without Smoothing

1.2 Naive Bayes, *add-1 Smoothing*

Laplace Smoothing, or *add-1 Smoothing*, would add 1 to every data point in the original data table D. This helps tackle the problem of zero probability prevalent within Naive Bayes classification.
Since all of the tokens within sentence S had non-zero data points in the initial data table D, our calculations did not change significantly. Our calculations with Laplace smoothing are as follows:

$$P(+|S) = 0.5 * \log(7/29 * 9/29 * 2/29 * 2/29) = -1.72$$

$$P(-|S) = 0.5 * \log(3/28 * 2/28 * 5/28 * 7/28) = -1.73$$

1.3 Naive Bayes, *Additional Feature*

In addition to the features already extracted from the text, we could extract negation words (e.g "not", "no", "never", etc.). This would be able to help us differentiate between sentences such as: "This film was disappointing" and "This film was *not* disappointing", which clearly have very different meanings to us, but the Naive Bayes classifier currently only looks at the token "disappointing".

2 Movie Review Sentiment Classification

2.1 Naive Bayes

Question 2.1 asked us to implement a Naive Bayes classifier with *add-1 smoothing*, as described in the lectures and reading.
In `classifiers.py`, we implemented the `NaiveBayesClassifier` class with methods `init()`, `fit()`, and `predict()`.
We decide to set up lists of prior probabilities and likelihood probabilities. The `init()` method set these lists to `None`.
In the `fit()` method, we used the shape of the argument array X in order to give us the number of

sentences and the size of feature dimensions (aka the number of features). We also used the argument array Y in order to find the classes (by taking the unique values using `np.unique()`). We then initialized `self.prior` and `self.likelihood` using `np.zeros` for their respective array sizes.
After all of the setup, we then computed the prior and likelihood probabilities with add-1 smoothing. Lines 60 and 61 calculate the prior probability for each class by using `np.mean` to calculate the number of times each class c occurs in the target label Y divided by the total number of sentences. The likelihood probabilities for each feature given each class are calculated in Lines 63-65. These code lines loop through `self.classes` in a similar way to Lines 60 and 61, however they differentiate in Lines 64 and 65. Line 64 counts the occurrences of each sentence belonging to each class (for each iteration, class c) and stores that information in `X_c`. `X_c` is then used in Line 65, where we calculate the likelihood probabilities with *add-1 smoothing*. The *add-1 smoothing* is applied when we add 1 to the numerator and the number of features to the denominator, thus eliminating any zero probabilities.
In the `predict()` method, we set up a blank array `y_pred` which we would later use to store our predictions. Then, similarly to our `fit()` method, we looped through `self.classes` and computed the prior and likelihood probabilities using `np.log`. Using these computed values, we then iterated through each sentence in X and computed the posterior probability for each class. After computation, we appended each calculated posterior probability to a list containing all posteriors for each class, and then selected the label with the highest log posterior probability to append to `y_pred`. We then returned the list `y_pred` as a `np.array`.

1. **Accuracy Reports** The AlwaysPredictZero model, which was used as a basic classifier to compare the others to, had a training accuracy of 49.93% and a test accuracy of 50.00%. Our NaiveBayes model had a training accuracy of 99.79% and a test accuracy of 71.20%. These numbers are a significant positive improvement from the AlwaysPredictZero model.

2. **Random dev Samples** 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?

3. **Words with Highest/Lowest Ratio** List the 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?

2.2 Logistic Regression

In the write-up, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report the accuracy without L2 regularization on the train, dev, and test sets. How does it compare to Naive Bayes?

2. Add L2 regularization with different weights, such as $\lambda = \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$. In your write-up, describe what you observed.

2.3 Bonus