Naive Bayes Classifier

by David Allen ([redacted])

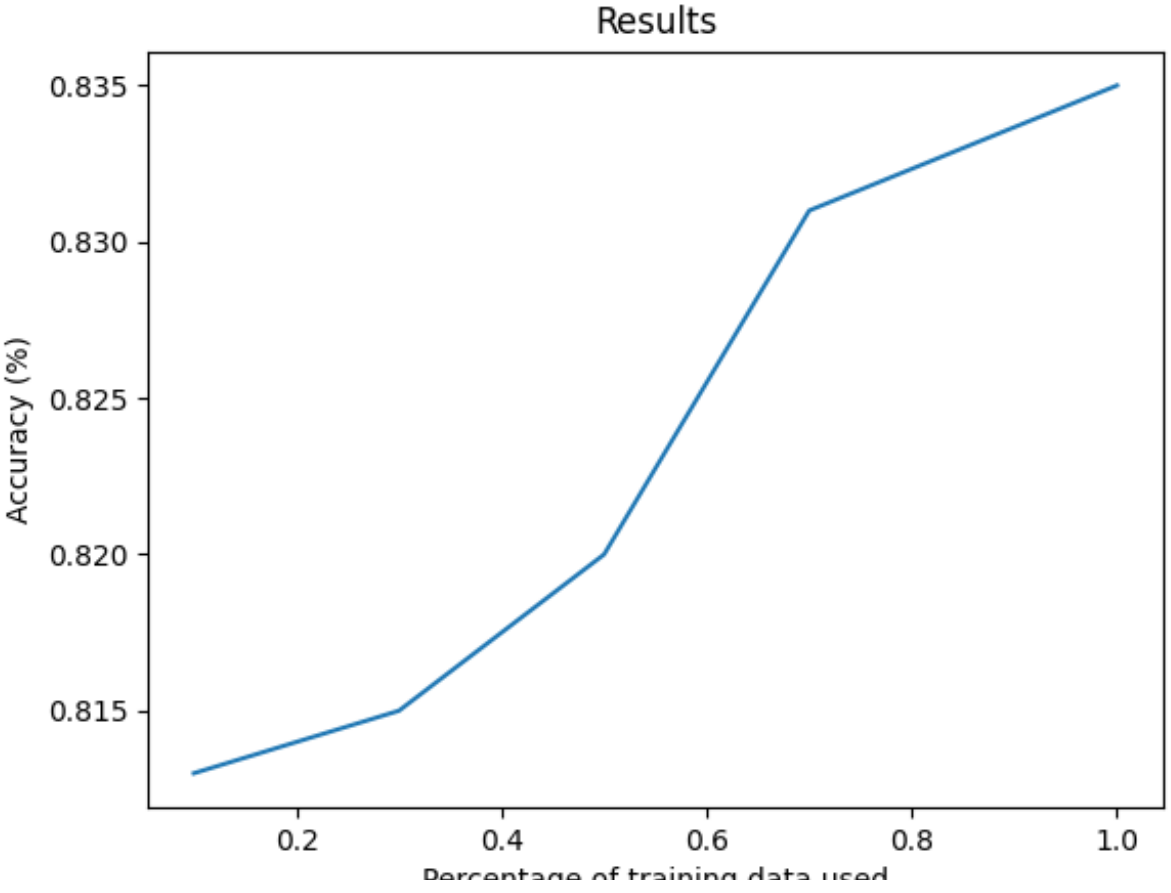
Implementation

The Naive Bayes Classifier was implemented in native Python. It takes a csv list of reviews as training data with their corresponding scores as labels. The top 1000 most frequent words in the data are selected as features. Likelihoods are calculated based on the frequency of each feature word using Bayes' Rule. A review is tested by taking the product of priors and every likelihood for each word in the review for each class and finding the argmax. For our problem, the model predicts a '5' for positive reviews, and '1' for negative.

Laplace smoothing is also implemented, with the option to disable.

Analysis

The algorithm achieved around 83% accuracy with 100% of the training data. As the amount of training data increased, so did too the accuracy. The change in accuracy is not dramatic, only showing an overall difference of a couple percentage points. Thus it seems that the amount of data available to Naive Bayes does not have a huge influence on performance. This may be advantageous when dealing with a problem that has a small dataset.



Percentage of training data used

Looking at the confusion matrix for the algorithm's performance, we see that precision is about 80% for positive reviews and 90% for negative reviews. These values are not too far off from each other, suggesting that the algorithm was about as good at predicting positive reviews as it was negative reviews. The slight deviation could be due to the slight imbalance in the dataset.

```
Confusion matrix
487 | 122
43  | 348
```

In this case positive class was guessed a little more frequently.

Side Note: When running the algorithm multiple times, I occasionally get an accuracy of 86~87% with a confusion matrix that looks like this:

```
Confusion matrix
472 | 76
58  | 394
```

I was not able to determine where this deviation comes from. When this occurs, the precision is almost equal for both classes, again around 86~87%.

Shortcomings

While the algorithm performed decently, especially given that the baseline for the problem is around 50% (the classes are roughly evenly distributed), 80% is not a state-of-the-art score by any means. Ideally we would like to improve this score by understanding where Naive Bayes falls short.

The Naive Bayes algorithm makes a major assumption, that every feature is independent of each other. This is certainly not the case for most words in our review classifier. For example, the word "great" may appear in many positive reviews. But if it appears after the word "not", then it is more likely to appear in a negative review. However, the algorithm doesn't account for the words surrounding the features, only the features themselves. Thus reviews that negate positive statements lead to ambiguous results.

An example of an ambiguous review:

> *"RUDE AND UNHELPFUL... where do all these **great** reviews come from... the staff? owner's friends??? I have tried to support this **local** business 3 times... and each time they have had no interest in making me feel **welcome** in the shop or answering my questions. I have come in to buy violin parts/accessories for my 8 year old, I don't have any music background or **knowledge**... could they act slightly **helpful**? Each time it seemed I was tearing them away from some **very** important thing they were doing behind the counter. Also, they wanted to sell me a set of violin strings for $87 that I later found out are $60-65 everywhere else in the Valley and on the internet. I will not try a fourth time, believe me, I am sure there are other Valley music stores that will be **happy** to have my business."*

As you can see, there are many potential words in this review that may have a high likelihood to appear in a positive review. The algorithm misclassified it as positive, even though it is

clearly not so. The Naive Bayes algorithm alone is not good enough to distinguish this type of review.