Carlos Adrian Bueso

Dr. Dmitriy Dligach - COMP329 Natural Language Processing - Homework 2 Written Report

For this project, we had to program a Naïve Bayes binary classifier from scratch. I decided to keep using the dataset from homework 1, which has 10,662 different movie reviews. The objective was to separate the positive sentiment reviews from the negative ones. The algorithm along with the theory behind the system was extracted from chapter 4 of Jurafsky’s and Martin’s ‘*Speech and Language Processing’* book.

First, the data was normalized and pre-processed. Multiple dictionaries were used to structure the data. **allData** contained every review as key and either the string ‘pos’ or ‘neg’ as value. **posData** and **negData** included the reviews separated by positive and negative sentiment. When reading the reviews, unnecessary punctuation was removed. The function **negTokenization()** was used to implement one of Jurafsky’s proposed optimizations, where after every negative token (e.g. doesn’t, isn’t, etc.) a string ‘NOT’ is appended to the following words until the next punctuation mark. The data was then split into 3 categories, including **trainingData** (70%), **developmentData** (15%)¸ and **testData** (15%).

The function **trainNaiveBayes()** takes a dictionary D (trainingData) as the parameter. It could be implemented to take a list of classes as well, but because the classifier is binary, I chose not to. We collect useful data like the number of documents in D, the number of documents for each class on D, and quickly calculate the log prior values for both classes. The function then proceeds to iterate through the trainingData, optimizing for sentiment analysis by removing duplicates of words on individual reviews. The frequency of the words is also counted and added to the dictionary **vocabularyD**, and two other dictionaries are created. **bigDocPos** contains all the words in the positive reviews along with their frequencies as values, and **bigDocNeg** is the same but for the negative reviews (bigDocs). Here we also take advantage of the iterations to remove **stopWords** from our vocabularyD.

The function then implements two more optimizations for sentiment analysis in Naïve Bayes, first allowing us to remove infrequent words from the bigDocs by choosing the least amount of frequency value that we wish to have (**removeBy**). The other optimization uses the positive words and negative words datasets to add **weight** to the words included in these lists on the bigDocs. Then we reach the calculations for likelihood. These are stored in a dictionary for each class, **loglikelihoodPos**¸and **loglikelihoodNeg**. We iterate through all words in vocabularyD, counting both the times the word shows up in bigDocs, and also calculating the total number of words on each bigDocs. Add-1 smoothing is used for computing the value of the loglikelihoods. Finally, the function returns the logprior for each class, loglikelihood for each class, and vocabulary of D.

The function **testNaiveBayes()** takes a **testdoc** with the data to test, along all return values from trainNaiveBayes() as parameters. It then proceeds to read each review word by word, checking if it is contained in vocabularyD, and calculating the product of the likelihoods of each word times the logpriors, all for each class. The values **sumCPos** and **sumCNeg** contain these computations. These then are compared to choose the highest one, finally allowing the system to classify the review as either positive or negative.

The system was first trained with the trainingData and tuned with the developmentData. With these two dictionaries, the accuracy was 49.87%. Later, for the final results, the trainingData and the developmentData were used as training data for the classifier, yielding an accuracy of 48.25% for the final result. 387 positive movie reviews were classified correctly, and 385 negative reviews were classified correctly. 413 positive reviews were classified incorrectly, and 413 negative reviews were classified incorrectly. 2 were not classified (kept as neutral). The main features that yielded the biggest accuracy upgrades were both the removeBy variable, along with the weight. By experimenting with different values, the system was able to reach a ~52% accuracy (later on disregarded as when removing the words from vocabularyD, the system had an excessively lower accuracy). The best values, removeBy = 70 and weight = 125 were chosen. For both the negative and positive reviews, the best features were the added words from the datasets (posWords.txt and negWords.txt), as the fact that one can manipulate their weight for the likelihood allows you to gain bigger control over their effect. Lastly, the negative tokenization also increased accuracy by around ~.5%.

Some examples from the test set show how some of these classifications were very confident and accurate, while others perished, mainly because of the use of words that do not imply sentiment at all:

*“peralta captures , in luminous interviews and amazingly evocative film from three decades ago , the essence of the dogtown experience . “*

**sumCPos** = -10447849853949.607, **sumCNeg** = -22108605491207.195 , classified correctly due to high use of positive sentiment words.

*“'dragonfly' dwells on crossing over mumbo jumbo , manipulative sentimentality , and sappy dialogue . “*

**sumCPos** = -2691077.417908821, **sumCNeg** = -2495944.0148347546 , classified correctly and system was confident due to high use of negative sentiment words.

*“with the prospect of films like kangaroo jack about to burst across america's winter movie screens it's a pleasure to have a film like the hours as an alternative . “*

**sumCPos** = 3389941316.9857106, **sumCNeg** = 9303093329.779675 , classified very confidently incorrectly as negative, probably due to absence of sentiment in general.

*“one of the best of a growing strain of daring films . . . that argue that any sexual relationship that doesn't NOThurt NOTanyone NOTand NOTworks NOTfor NOTits NOTparticipants NOTis NOTa NOTrelationship NOTthat NOTis NOTworthy NOTof NOTour NOTrespect . “*

**sumCPos** = 1.845116291394816e+24, **sumCNeg** = 5.554721806612482e+24, clearly shows the negative downside that negative tokenization can also have, as the system was really confident with this incorrect negative classification.

To conclude, I would assert the system is faulty. It still has a lot of features and optimization to add to make it more accurate. Some of these examples show how different these reviews can be, and how difficult is the task of truly picturing them through sentiment analysis. Some of the classifications show the power of Naïve Bayes, while others show how there is a lot more to account for with a system using Bayesian probability. Probabilistically, throwing a coin works better than my system, which makes me almost certain that a mistake was made somewhere. I will be working to fix it and find what was the part of the algorithm that I implemented erroneously.