Project 3 - Naïve Bayes

Goal: To gain experience with implementing a naïve Bayes algorithm and with the design choices that accompany using one.

Description: Companies use sentiment analysis to monitor how their products are being talked about on social media in the hopes of forestalling the spread of any negative messages. A wide variety of natural language processing algorithms can be brought to bear on the problem of recognizing positive vs. negative sentiment in text. We are going to use one such approach: a Naïve Bayes classifier. We are going to apply this to the problem of recognizing whether a movie review on Rotten Tomatoes is positive ("fresh") or negative ("rotten").

Training: Your code will take in two training files – one containing examples of "fresh" reviews and one containing examples of "rotten" reviews – and will count the number of occurrences of each word in the two files, e.g. "outrageous" may occur twice in your fresh reviews, but forty times in rotten reviews. Once you've counted the number of times that a word occurs in fresh/rotten reviews and the total number of fresh/rotten training words, you can compute the conditional probability of each word occurring, given the two types of review. You should use a held out set to learn a probability for unknown words occurring in the text.

You should provide the user with an intuitive menu interface that allows them to do the following:

1. Load in two training files: one containing rotten and the other fresh reviews. You should prompt the user for the filenames.
2. Dump the word counts and conditional frequencies out to the screen or to a file in an easy to read format.
3. Prompt the user for a testing filename and then compute the probability that it is a fresh or rotten review. Display the two probabilities to the screen as well as your program's verdict as to whether the review is good or not.

Priors: In computing the probability that a review is positive or negative, you need a prior probability representing what proportion of your reviews are typically positive and what proportion are negative. You should use global constants for these values, so that the grader can manipulate them. To set their values for your submission, you should look up the number of rotten and fresh reviews for all of the movies currently in theaters.

Underflow: You may find that your probability calculations for long reviews become so small that they exceed the precision of a double variable. You may want to investigate what larger variable structures are available in your programming language, e.g. long doubles. Nonetheless, variable underflow may still be a problem. In NLP, this is commonly addressed by working with logarithms of probabilities instead of the raw probabilities (and instead of multiplying the raw probabilities, the logarithms are added). Alternatively, if you want, you can define a global constant that is a power of 10 (100 worked well for me) and every time you multiply in a conditional probability, you also multiply in your constant. It won't alter the program's verdict of fresh vs. rotten, since both probabilities will be multiplied by the same amount, and when displayed in scientific notation, it won’t change the mantissa of your answer, only the exponent, meaning that the grader can still verify that your code is computing the frequencies and probabilities correctly.

Creating Training Data: You will need to create 2 training files and at least 8 testing files for your sentiment analysis project. You will submit your training and testing files along with your project. However, you should expect that the grader will also create new training and testing files.

Your training files should contain at least 6000 words each (after being preprocessed). This will probably involve pasting together a lot of reviews. You should also create at least 8 testing files, each containing one review. Four of your testing files should contain fresh reviews and the other four should contain rotten reviews.

Preprocessing: You should experiment with preprocessing the text of the reviews before running your naïve Bayes classifier. For example, might find it helpful to remove all punctuation (except apostrophes) and convert all the letters to lowercase. Or you might want to remove the 100 most common words of English (like “the” and “and”) which are likely to be equally common in both types of reviews. However, you might prefer to leave those details in. Maybe positive reviews have more exclamation points or negative reviews have more ALL-CAPS. If you decide to preprocess your training and testing data, then you should be sure to include the code or tool commands that you used to preprocess the data in your submission so that the grader can apply the same preprocessing to her test-suite.

Coding Requirements: You are free to choose which language you would like to use in implementing the project. The grader already has C++, Java and Python compilers installed. If you wish to use a different language, you must include a README file with your project submission in which you give step by step instructions on where to download the compiler, how to install it (and any libraries that are needed) and how to run your project once the compiler is installed. If in doubt, you can assume that if your code runs correctly on the lab computers in CTC 114, then it will run correctly for the grader.

Your code is expected to follow all good programming practices, i.e. well-commented, broken into functions, thoroughly tested, etc.

Submission: Please zip up the entire contents of your project and submit the zip file via Canvas. Don’t forget to click the submit button after uploading the file!

Modifying the Project: If you would like, you can alter this project to be any kind of text classification task, e.g.

- *Author Detection:* collect two samples of writing by two different authors (the authors should share some similarities, comparing Dr. Seuss to Jane Austen is too trivial) and then train your system to correctly recognize the writing style of the author in question.

- *Genre Detection:* collect samples of writing from a pair of genres, e.g. opinion vs. news or romance vs. drama and then train your system to classify a new sample of writing as belonging to one genre or another.
- *Political Bias Detection:* collect samples of similar news stories told from a liberal and from a conservative political viewpoint and then train your system to recognize which slant has been applied to a particular story.
- …

The text classification task should be non-trivial and the two samples should contain a fair number of overlapping words – so language detection would not be appropriate since relatively few words are shared across languages unaltered. Your system should learn to accurately distinguish between at least two categories of text based on the frequency of various words being used. While the training files can be as large as you like, the test samples should be relatively small – a paragraph at most.