Machine Problem 1 Report

Brian Lee

February 2017

1 Architecture

I have three files for this assignment: nbc.h, nbc.cc, and main.cc. My main.cc file performs most of the preprocessing and the output formatting, as well as timing the training and testing of the Naive Bayes Classifier.

My nbc.h file contains the declarations and the member variables and containers of my naive bayes classifier. It also holds the implementation of a struct I created called Word, which holds the word string, and the various numbers and probabilities associated with each individual word. The member variables contained are the number of good and bad reviews from the training set, the probability of reviews being good or bad, and a vector of Words.

Finally, my nbc.cc file holds the implementation of all of the functions in my header file. The main functions in question are my training and testing functions. In addition, there are also two helper functions which perform a couple functions, pardon the pun, and make my life a bit easier.

2 Preprocessing

I admit, I didn't spend too much time preparing the documents. I placed the .txt files into ifstreams, placing the training.txt file into two different ifstreams so that I could both train and test with it, and I fed the training and testing functions the files line by line since each review was a single line in the files. I admit that I didn't remove the label at the end which denotes if the review is positive or negative, and that might have inflated my results. - EDIT: I've changed the training code to ignore the strings "1" and "0" so that my results are no longer inflated.

3 Model Building

In my naive bayes classifier, I used a vector of structs called Word which contains individual words from the training set, the number of occurrences, and a few other statistics vital in determining probability. Speaking of, those are some of

the most important statistics help by the Words: the probabilities of the words occurring given that the review are good or bad.

To sum up how my training function worked: I'd pass in a single review and an int value holding the label at the end of the review. I'd then parse the line word by word and check it against the vector of structs. If a word was not in the vector, a new Word would be created and added to the vector. If the word was in the vector, then the corresponding Word would have its statistics updated.

One of the things I made sure to do was to allow each Word to only be updated once per review. This is done for a few reasons, chief among them being that I found myself unable to get my classifier to ignore words that are worthless in determining a good or bad review like "a", "it", "the", etc. The second reason is because this would give certain words more weight, and unless I was using TF-IDF weighting, that likely cause a few problems.

In addition, with each review, I would update a series of statistics and variables in the nbc itself, namely the probabilities of good and bad reviews.

4 Results

After changing my code so that my results are not inflated, my results dropped by about .1. My accuracy on the training set is now .795 and my accuracy on the testing set is now .646. My training time is longer than I'd like, taking around 30 seconds to train and 52 to test.

When it comes to bad reviews, some of my strongest words were "Nope", "Yuck", "not", "warning", and "worst". When it came to good reviews, "Wow", "Crazy", "Star", "Trek", and, hilariously, "Elvis" were some of the strongest. The probabilities in question were determined by taking the probability of the words appearing and the review being good/bad, and dividing the result the the probability of the review being good or bad, which led to... inflated numbers, like a probability of 2 for several words.

Additional words include "lame"(bad), "delight"(good), "musings"(good), "rarely"(bad), "enjoyed"(good), "cheap"(bad), "redundant"(bad), "poor"(bad), "witty"(good), and "heartfelt"(good).

5 Challenges

The main problem I had was figuring out where to start. I had never done anything like this without a framework of some sort, as I was given in CMPSC 130A, so I was adrift, for lack of a better word. So, I began with my main.cc file. By following the requirements in the output and extrapolating the needs of my classifier to train and test, I managed to figure out how to build my naive bayes classifier, the training and testing functions, and the member variables that the classifier would need.

After that, my problem was figuring out how to hold all of the individual statistics for each word, a problem solved by using a struct that would hold

all of the individual statistics along with the string containing the word. The problem of holding all of these structs was solved by the use of a vector, having been chosen for its relative flexibility.

Beyond that, my only problems were in parsing the training and testing files. A quick search of c++ libraries gave me the fstream library for the files, istream library for passing in the files line by line (or review by review, rather), and the sstream library for parsing words.

6 Weaknesses

First of all, words like "it", "a", and "the" are not ignored, which inflates the vector and costs training and testing time. Second, the nbc is not as accurate as it could be as it does not weigh the words through TF-IDF weighing, nor does it stem words, which would give more accurate statistics per word. In addition, the vector is not the most efficient structure when it comes to searching (giving linear time), nor do I believe that the struct I used is the most efficient way of holding the statistics.

In solving the former, a list or library of words to exclude would likely help with accuracy, although it may cost time. Implementing TF-IDF weighing for each word in the vector would greatly increase accuracy, though again at the cost of time. And different data structures than the ones I used would likely save on time with larger loads, though sacrificing it on shorter ones.