Naïve Bayes Classification

Christina Wallin, Period 3
Computer Systems Research Lab 2008-2009

Abstract

One area of computational linguistics is the classification and comparison of texts into classes. This comparison could be author classification, spam filtering, or in the case of this project, the classification of texts from different genres of news stories. The goal of the project is to classify a text as belonging to one of two classes based on the actual words in the text. This method of classification is called a naive Bayes classifier. In this age of huge amounts of data available online, a classifier which can discriminate between two types of news is becoming ncreasingly useful.

Background

The naive Bayes classification is a relatively simple method for classifying texts based on the false assumption that all of the variables, in this case words in the documents, are independent of each other. Even though this assumption is false, this project is done to achieve fundamental understanding concerning the effectiveness of the naive Bayes as compared to other methods, and to find a way of improving upon the performance of this classifier. However, it is also done to try to provide a way for news stories from different genres to be classified.

Methodology

The first step in classifying a document is to read and parse the file using the file.py program, removing all punctuation and case and making a dictionary with the words occurring and their frequencies. In this step, it is possible to use a Porter stemmer to stem words to their roots—for example, “running” and “runs” would both go to “run.”

Next, for each class/genre, train.py trains the program as to what characteristics are most prevalent. It does this with the words themselves, creating an array containing the probability that each word occurs in the class for all the files.

There are two different ways to calculate the PFX, with multnomial and multivariate methods. The multinomial method takes into account the frequency of the words in the texts, whereas th multivariate methods depend upon the number of files in which a particular word is in. The stopwords “and,” “a,” “an,” and “the” can also be removed in this step.

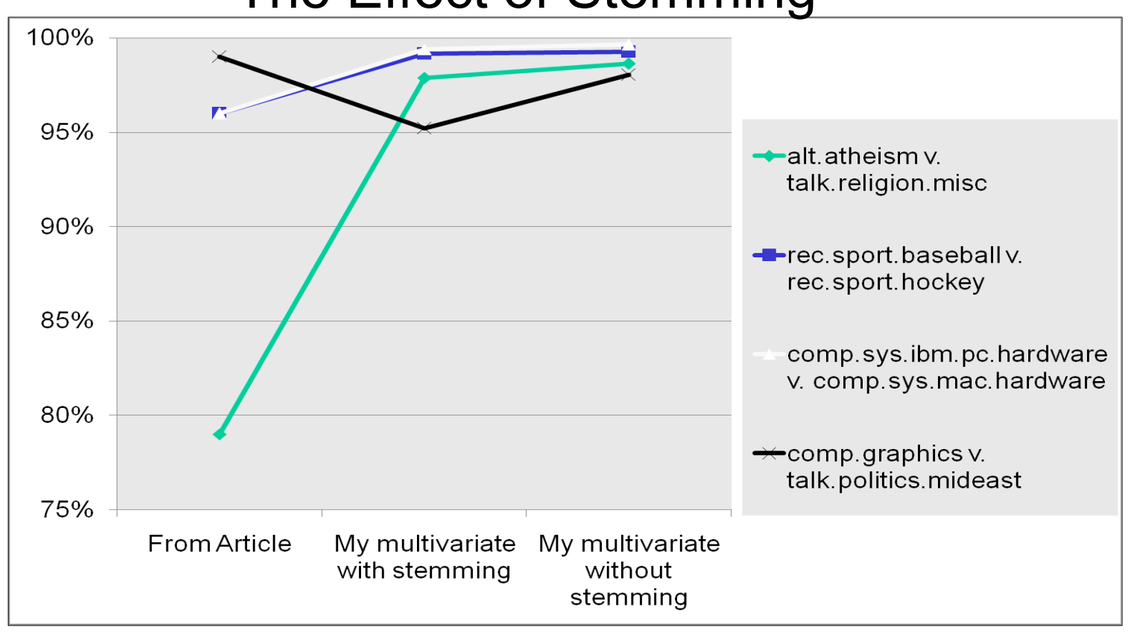
Then with this probability vector (PFX) for each class, I can calculate the probability that a text is of a specific class in test.py by generating the probability vector for that specific text and comparing it to the PFX for each class. The probabilities of each word occurring based on each class are multiplied in order to determine what the probability that the file is in a certain class is.

The program has been tested by using small data sets and checking manually whether or not the frequencies were correct, and using my testing program. I also tested the program by making sample data sets based on a programmed-in probability for each word. With this perfect data, I was able to check my PFX vector and found that it figured out the same probability as was programmed in. Thus, the probability calculation is correct. I was also able to check whether the testing part classified the perfect data correctly, which it did.

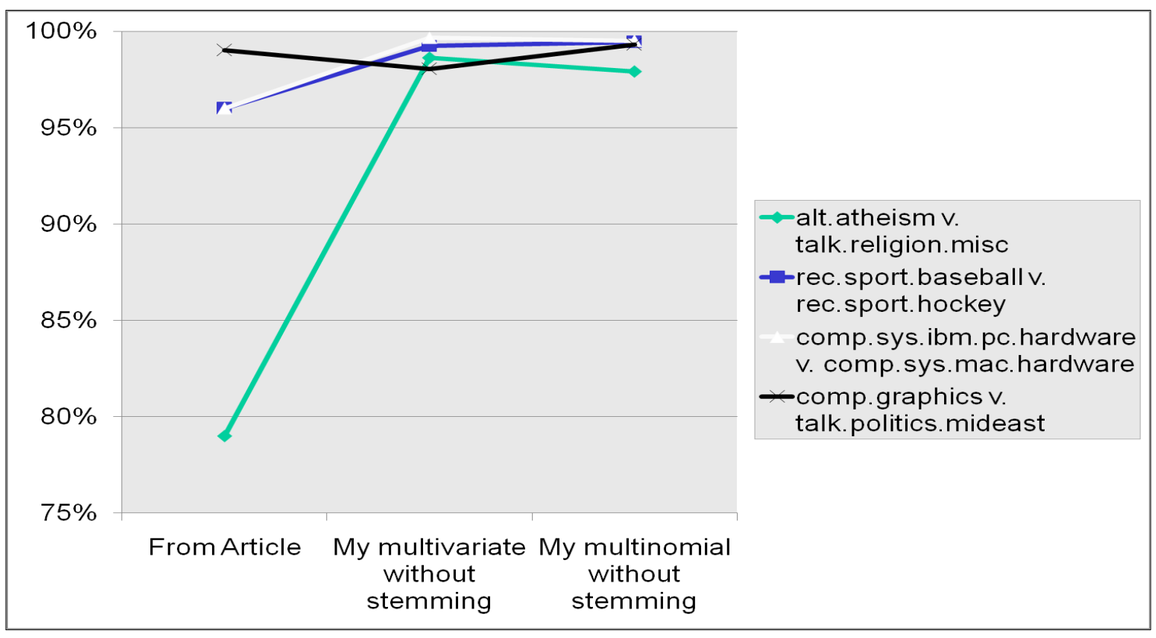
Results and Conclusions

A working naïve Bayes classifier with both a multivariate and multinomial method of testing has been created. Three different experiments have been doing, to determine whether or not using the Porter stemmer to stem words improves the percentage of files classified correctly, and to see whether the multivariate or the multinomial works better, and whether ignoring stopwords like “and” and “the” helps. The graphs below show the percentage classified correctly. I found that it in fact it does not help to stem words, and was in fact about a percentage point less effective. The multinomial and multivariate methods have approximately the same accuracy, splitting the test cases. Ignoring stopwords does help in most cases, though not in all. Many of these optimizations do not help, because they are destroying data, thus decreasing the accuracy.

The Effect of Stemming



Multivariate v. Multinomial



The Effect of Stopwords