TJHSST Computer Systems Laboratory
Senior Research Project Paper
Naive Bayes Classification
2008-2009

Christina Wallin
Period 3

May 31, 2009

Abstract

One application of computational linguistics is the classification and comparison of documents by computers. This project uses a naive Bayes classifier to classify a text as belonging to one of two classes based on the actual words in the text. It also compares the performance of a multivariate and multinomial types of naive Bayes classifiers, and optimizes the classification with parsing and other methods. With huge amounts of data becoming available online, a classifier which discriminates between two types of news will be increasingly useful.

Keywords: naive Bayes classifier, computational linguistics, binary independence model, Laplace smoothing, multivariate Bernoulli event model, multinomial event model

1 Introduction

The naive Bayes classification is a relatively simple method for classifying texts. It is based on the false assumption that all of the variables, in this case words in the documents, are independent of each other. For example, if a science article contained the word "rover," it would most likely also have the word "Mars." However, the naive Bayes would not take the added probability into account. This goal of project is to compare two different types of naive Bayes classifers, the multivariate Bernoulli event model and the multinomial event model, and to optimize the performance of these classifiers. With the great propagation of data on the Internet, an effective way to sift through the texts by means of classification will be essential.

The naive Bayes classifier in general works by first training the computer with data in order to find the relative probabilities of words for each class of texts. Then, because it makes the assumption that all of the variables are independent, it can find the probability that a specific text is in a class. At first, I have written a binary independence model, called so because it compares only two classes [4]. It is also classified as a multivariate Bernoulli event model because does not take into account the frequency of the occurrence of a specific word in the text, but merely its presence or absence [6]. My current naive Bayes also can discriminate words based on their stem instead of the actual form using a Porter stemmer, by which "walking" and "walked" would be counted as the same word.

The 20Newsgroups database [9], the data set used for classification, is divided into training and test sets, so I will be able to use that quality as a constant for training and testing different versions of the program. The method for determining the effectiveness of a classification system is the percentage of the test sets classified correctly.

Later in the year, I hope to optimize the algorithm by making a multinomial event model (explained below). I also will consider other optimizations such as accounting for the length of the texts and analyzing the data itself to see which sorts of texts are the best for classification.

2 Background

There have been many studies which have used the naive Bayesian classification method, and it was first used in a published paper in 1966 for a medical study on computer-assisted diagnosis [1]. Hand and Yu (2001) reviewed past uses of the naive Bayesian method for classification. Using theoretical and real data situations, they showed that the naive Bayes is not an excessively inaccurate method because of its false assumption that all of the variables, which in my project are occurrences of words, are independent [2]. Shen and Jiang (2003) explain a way to improve the naive Bayes by combining it with logistic regression, another method of classification involving a more complicated mathematical model to determine the probability that each variable is in the class. The naive Bayes method itself is a bit overconfident in classification, and so by combining it with logistic regression, the classification is improved [10].

Additionally, there are several different ways to implement a naive Bayes classifier. MacCallum and Nigam (1998) describe two common methods which I will use in my project, the multivariate Bernoulli event model and the multivariate event model. I first started coding the multivariate model, which keeps track of whether a word occurs or does not occur in a document. When it is calculating the probability that the document is a member of a specific class, it includes the probability both for the occurrence and non-occurrence of words. The multinomial model, however, which I will start this quarter, keeps track of the frequency of each word in the document. The multinomial is almost always more effective than the multivariate method, except for very small vocabulary sizes [6].

This project, since it both trains the computer to recognize the classes and deals with the words themselves in the text, is indeed at the intersection of linguistics and machine learning. For a different linguistic classification problem, dialectology, or the study of how language varies over geographical regions, Nerbonne (2003) explains computational methods which can increase the accuracy of the language study, specifically by lemmatizing the words, getting the root word. Since actual lemmatization is very difficult, this study used a Porter stemmer to approximate lemmatization with stemming. This stemmer, when given a word, gives back its stem based on various rules. The stemmer extracts the essential information out of a string and allows words which are just forms of the same word (e.g. stops, stopping) to be considered as such. This stemming eliminates morphological data, but it helps with understanding the lexical data, and for text classification the morphological part is inessential [7]. Thus, I am experimenting to see whether or not a Porter stemmer would aid in my classification.

There are also three different types of words in a file, according to Korutchev and Korutcheva (2008). They are specific terms or keywords, which occur much more often in the file than in the general vocabulary, function words like "the" and "and," which are essential for meaning and occur rather frequently throughout the vocabulary, and common words, which have meaning but are not specific to a specific sort of text (for example, the word "explain"). Thus, if you find both the frequency of words for the general vocabulary and in a specific class, you can calculate some of the keywords of a document [3]. Using this principle, my classifier could potentially be more accurate if I weighted the words which are more frequent compared to the vocabulary as a whole more.

3 Development

3.1 Programming Language

The programming language used for these manipulations is Python. An aid for the linguistic sorts of operations is the nltk package, the Natural Language Toolkit [8]. This package provides capabilities like the Porter stemmer programmed into it, and also has methods for displaying the data with the PyLab package [5]. The PyLab package is a method of making graphs and charts, very much like matlab. Visuals can be made of the probabilities that selected words occur in each class, as compared to a text to be classified. These charts help the viewer understand the processes the computer is going through as it compares the text and the probability vectors.

3.2 Corpus

The corpus which I am using to classify texts is the 20Newsgroups database. There are 20 different genres of news stories, and it is divided into training and testing sets. The training and testing sets are separated in time by a little, and so it is more realistic. Selected genres of news are comp.graphics, rec.sport.baseball, talk.politics.mideast, and sci.space.

3.3 Dictionary formation

The overall design of a naive Bayes classifier can be found in Figure 1. First, the program reads in a file from the 20Newsgroups database or another source and parses it to remove all punctuation and capitalization. Regular expressions were used to remove all punctuation, and Python's built-in methods for strings were used to change everything to lower case. Then, it creates a dictionary of words occurring in that text and their frequencies. During the formation of the dictionary, a boolean turns on and off stemming by the Porter stemmer.

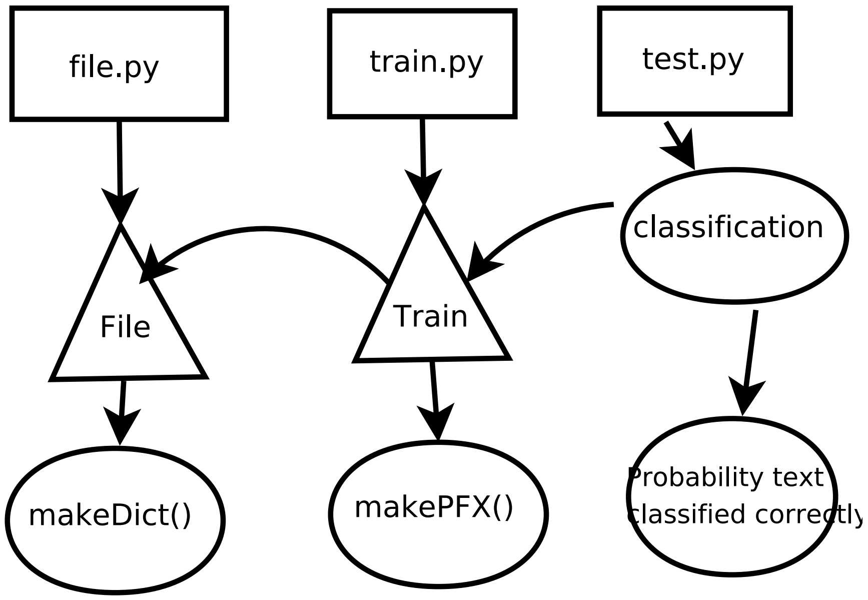

Figure 1: A model of the files and basic operations in the naive Bayes classifier.

3.4 Probability vector formation

The next step is to train the program as to what characteristics each class has. I did this with a probability vector (PFX) for each class based on the words occurring in them. This vector can be calculated in several different ways. The two methods I used were multivariate and multinomial event models.

3.4.1 Multivariate event model

This first model I created is multivariate, which means that it does not take into account multiple occurrences of a word in a document. For each individual text, the probability vector is binary, with a 1 if the word is in the text and a 0 if it is not. The probability vector for the class as a whole is a dictionary of all of the words occurring in the test cases with the probability for each variable (word) being the number of texts in the class which contain the word divided by the total number of texts in the class. Then, 1 needs to be added to the top and the size of the vocabulary to the bottom, so that there is never a probability of 0 that a term occurs. That process is called Laplace smoothing, and it accounts for words which are not found in the training cases but may be found in the test cases.

3.4.2 Multinomial event model

The second model, created during third quarter, is a multinomial model, meaning that it takes into account the number of occurrences total, not only the number of files in which the word occurs. The probability vector for each individual text is the frequency of the word in that text. The probability vector for the class as a whole is again a dictionary of all of the words occurring in any text in the two classes being compared, with the probability being the number of times the word occurs throughout all of the classes divided by the total number of words. Then, by Laplace smoothing, one must be added to the top and the size of the vocabulary to the bottom, so that there is never a probability of zero.

3.4.3 Stopwords

The final optimization I tried during fourth quarter was removing stopwords from consideration. Stopwords are words which occur very frequently in the English language, and my array of said words was: ”a,” ”an,” ”and,” and ”the.” These words were simply ignored while calculating the PFX vectors for the rest of the words, and thus were not taken into consideration later on in the classification stage either.

3.5 Classification

Then with this probability vector for each class, by Bayes' theorem which states that each vector is independent, you can calculate the probability that a text is of a specific class by generating the probability vector for that specific text and comparing it to the PFX for each class, thereby classifying the texts. For this, each variable (i.e. the occurrence or non-occurrence of each word) has to be calculated in order to form the probability that it is in a particular class. Then, the individual probabilities would usually be multiplied in order to determine what the probability would be. However, the multiplication can sometimes have problems, and so it is better to take the sum of the logarithms of the probabilities that the variable is an element of the class.

3.5.1 Bayes' theorem

Bayes' theorem is the assumption central to the naive Bayes classifier. This theorum states that all of the variables are independent in their determination of the class. In this case, the variables are individual words. The assumption is obviously false, because certain words usually occur together. For example, in a news article in the genre of news about space, if the word 'rover' occurs, the word 'Mars' is more likely to occur. However, even though the assumption is false the naive Bayes classifier is a rather effective tool for classification[4].

4 Testing

As far as testing goes, the 20Newsgroups database from which the news stories come is separated into training and testing sets. The testing sets come in folders with the correct class in the name, and so with Python methods to check the folder from which the text came whether or not the class was correctly determined can be showed. However, the fact that the program produces the correct class does not mean that the algorithm is working correctly. So, the tests cannot solely be based on that.

4.1 Dictionary formation

The program has been tested by using small data sets and checking manually whether or not the frequencies were correct. The portion which used regular expressions to remove punctuation was tested by printing out the parsed strings and seeing if the regular expression did the correct operation.

4.2 Manual testing

I test the program initially by seeing whether the individual components work. For example, I determine whether or not the probability vector is determined properly for a very small data set, and if it works on a small scale it should also work on a large scale. The naive Bayes itself is an algorithm of the sort that can be tested with specific small data sets. Additionally, Shen and Jiang (2003) also use the 20Newsgroups corpus for the naive Bayes classifier, so I can check the performance of my naive Bayes classifier against theirs. My percentages correctly classified are both higher and lower than theirs for the multivariate (see Table 1), suggesting that my naive Bayes is overconfident. However, the percentages in the article are for only 90 percent of the training data being used, while I use all of it, and I am estimating the percentages from the article off of a graph. This would indeed make mine more accurate, though I do not know how much more accurate it would be. Additionally, the naive Bayes in the article is a multinomial model, not a multivariate model, so I can't depend exactly upon the data. For the last test case with my multinomial (see Table 2), the percentage is much higher, suggesting that my program works correctly for multinomial, and thus overall [10].

4.3 Sample data

I also tested the program by fabricating 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.

4.4 Automated testing

I wrote a script to run multiple tests at one time, so that the tests could be performed more accurately and swiftly.

5 Results and end material

5.1 Initial results

At first, I compared texts from different genres of the 20Newsgroups corpus. With this program, the frequencies of words in different sorts of texts can be compared. The differences in words in different genres of news is the basic premise behind the naive Bayes classifier, and a visual representation with PyLab of the top 20 frequent words is in Figures 2 and 3. The two news stories which were used were about space science and baseball respectively, and they were approximately the same length. One difference which is striking is that the baseball text has quite a few words which are the most frequent which deal with baseball, while the only word in the top 20 with respect to frequency which is even remotely science-related in the space article is "black". This difference suggests that the scientific writing is more verbose and less to the point than the baseball writing. This perhaps would not be the case with different articles in the same genre, but the correlation is interesting.

5.2 Stemming

With the simple naive Bayes classifier, I tested the effect on the correct percentage of classification of stemming the words with a Porter stemmer. I expected that it would improve the classification, for it would allow words which are very similar to each other to be counted

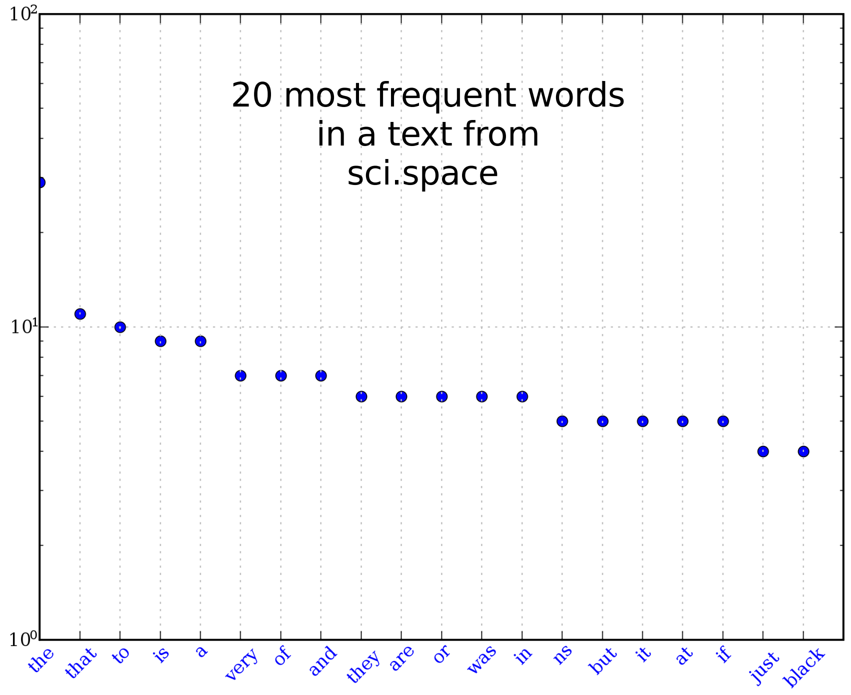

Figure 2: The frequency of the top twenty most frequent words in the sci.space genre from the 20Newsgroups database, plotted with PyLab.

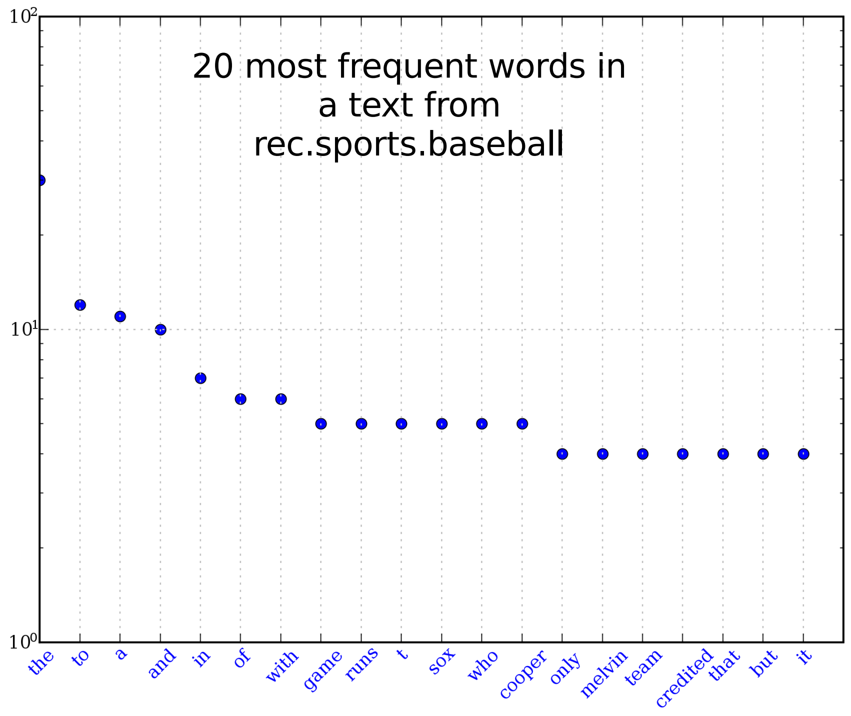

Figure 3: The frequency of the top twenty most frequent words in the rec.sports.baseball genre from the 20Newsgroups database, plotted with PyLab.

together. However, I found that it did not help classification to stem–it was actually less effective whenever the stemmer was used, especially for the 4th test case (Table 1). This makes sense, though, because when you stem the words, you are losing information about the words. Perhaps some genres use the past tense more, or something else like that, and so that information should not be lost. Thus, stemming the words did not improve the performance of the naive Bayes classifier.

Percentage classified correctly, Multivariate with stemming			
Classes compared	In article [10]	With stemming	Without stemming
alt.atheism v.			—
talk.religion.misc	79%	97.86%	98.63%
rec.sport.baseball v.			—
rec.sport.hockey	96%	99.16%	99.25%
comp.sys.ibm.pc.hardware v.			—
comp.sys.mac.hardware	96%	99.40%	99.66%
comp.graphics v.			—
talk.politics.mideast	99%	95.21%	98.03%

5.3 Multivariate and Multinomial

I also tested the effect of using a multivariate or a multinomial event model (see above for a description of each). I expected that the multinomial version would increase the classification percentage, since it accounts for the number of times a word occurs. However, with the class pairs alt.atheism and talk.religion.misc; and comp.sys.ibm.pc.hardware and comp.sys.mac.hardware, the multinomial event model was slightly less accurate than the multivariate model. These results suggest that the methods themselves work better or worse for different types of data, and it would be interesting to do more analysis on the test cases to see what the differences are, and which types of texts are classified better by which of the two methods. However, the multinomial results might be slightly less accurate than they could be, because the lengths of the individual files are not taken into account. For example, if a word occurs two times in a 25 word article, that is more significant than two occurrences in a 250 word article. If the lengths of texts in a class differ greatly, then the classification could be off for the multinomial.

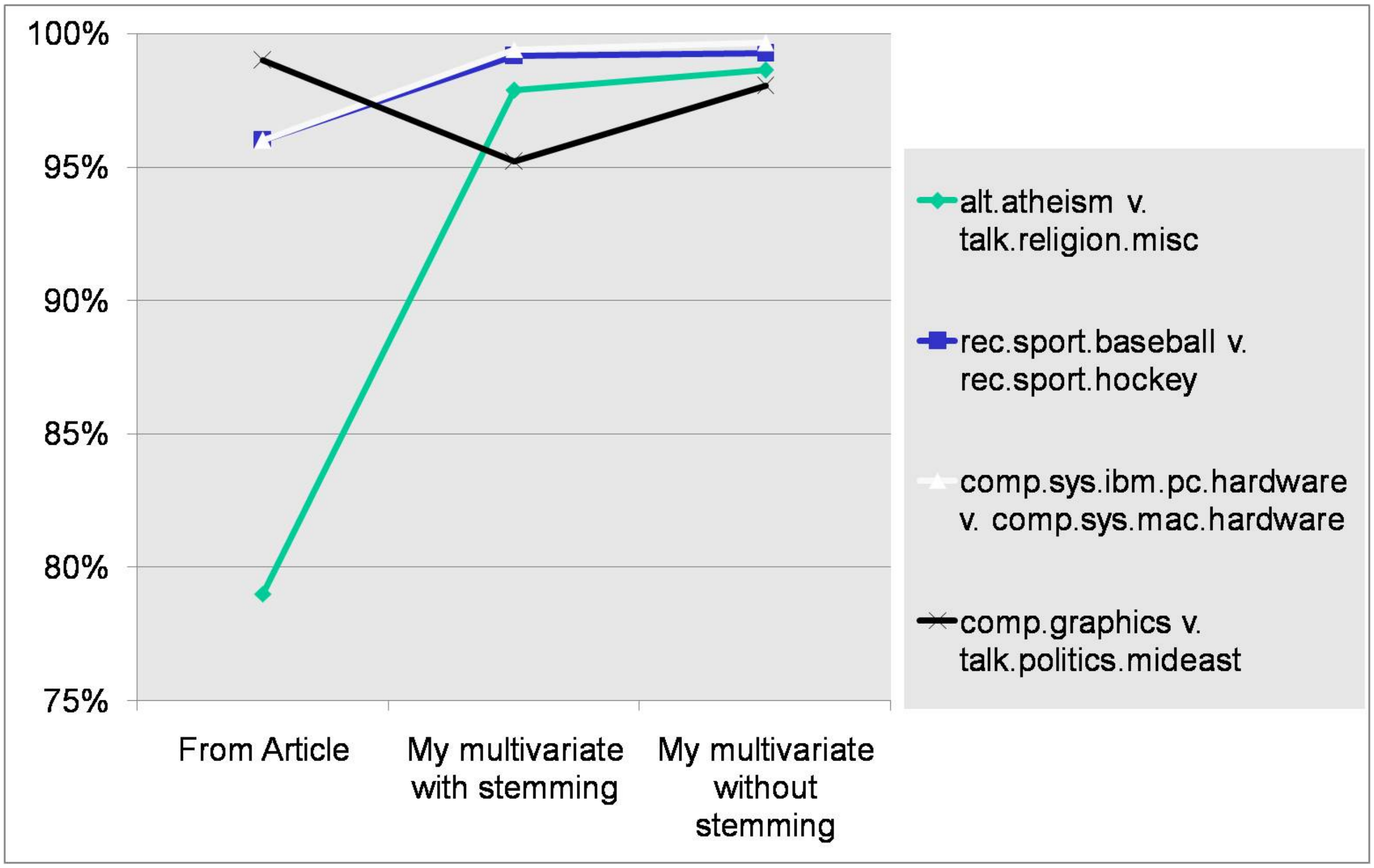

Figure 4: The effect of stemming on a multivariate Naive Bayes.

Percentage classified correctly, Multivariate vs. Multinomial without stemming			
Classes compared	In article [10]	Multivariate	Multinomial
alt.atheism v. talk.religion.misc	79%	98.63%	— 97.90%
rec.sport.baseball v. rec.sport.hockey	96%	99.25%	— 99.42%
comp.sys.ibm.pc.hardware v. comp.sys.mac.hardware	96%	99.66%	— 99.49%
comp.graphics v. talk.politics.mideast	99%	98.03%	— 99.30%

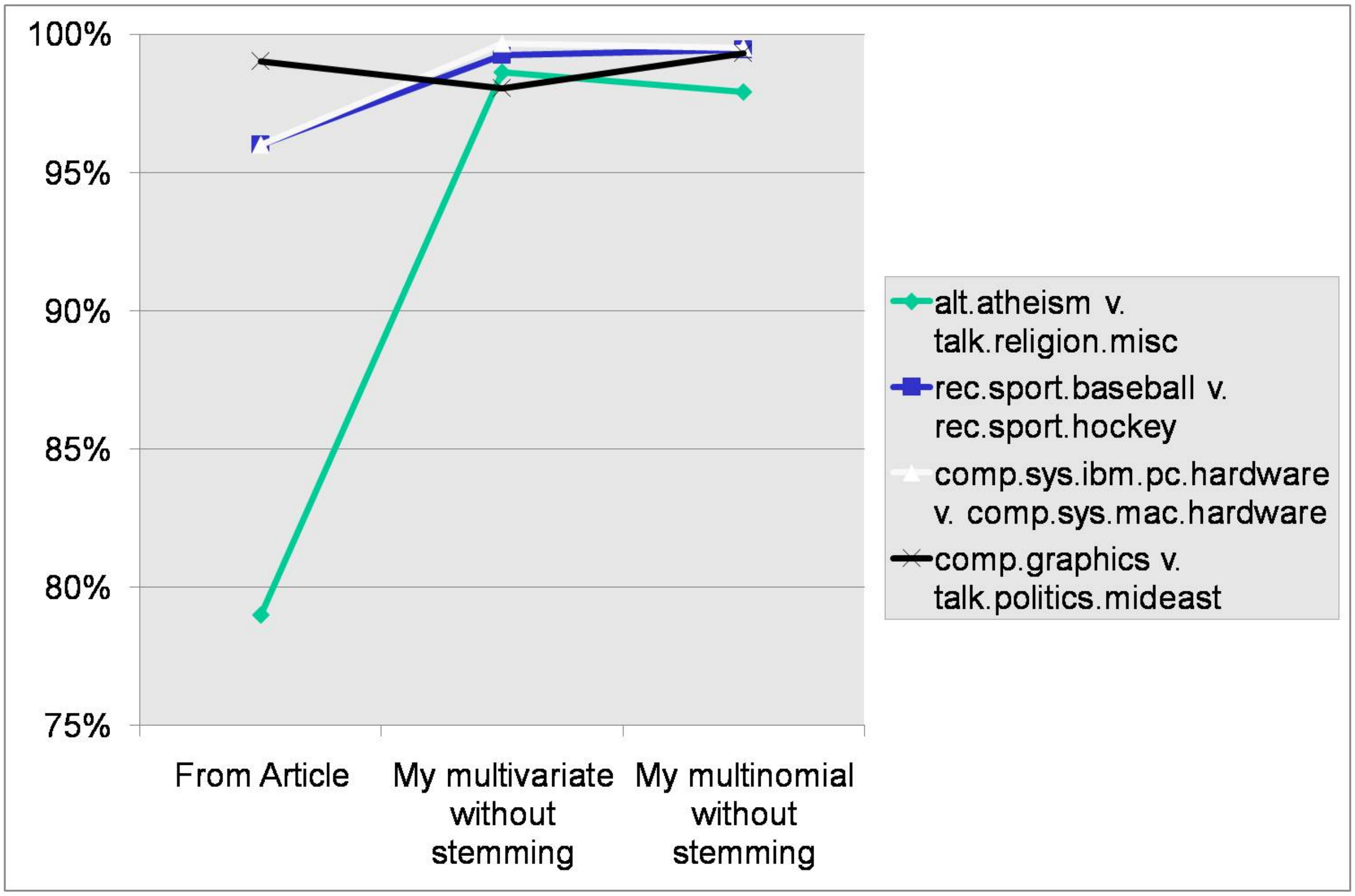

Figure 5: A comparison of the percentage correctly classified of multivariate and multinomial Naive Bayes classifiers

5.4 Stopwords

My final experiment was to consider the optimization of not considering stopwords. Since these words are irrelevant to the content of the piece, I postulated that they would not have an effect on the classification, and would in fact improve it, if removed. This was the optimization which worked the best, for it improved the percentage classified correctly for all but one of the test cases, and it only decreased slightly for that test case. For one of the test cases, in fact, the percentage classifed correctly improved by almost 1.5%. For this sort of computational linguistical operation, the effectiveness of the classification is very dependent upon the data itself, and some things which help some data sets could be expected not to help others.

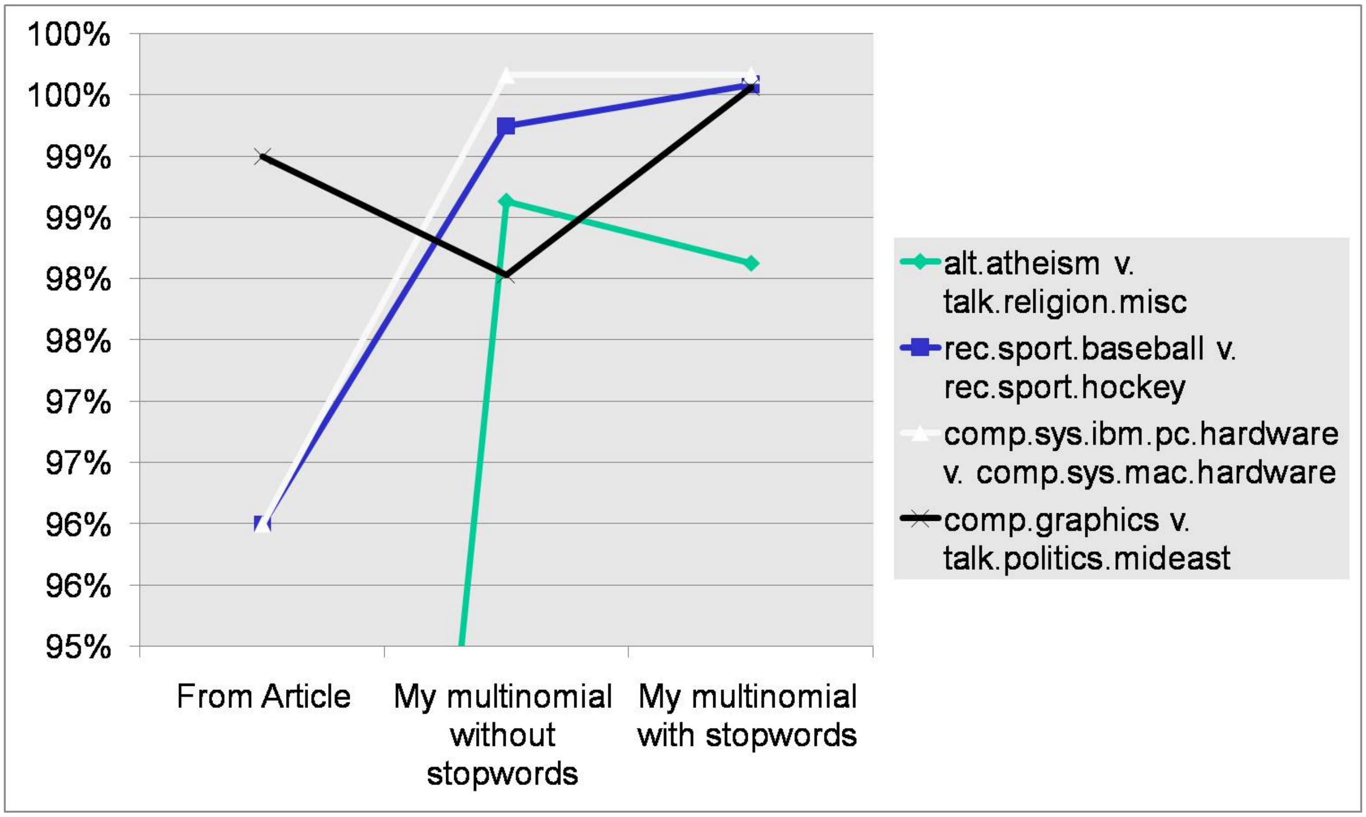

Figure 6: A comparison of the percentage correctly classified of a multinomial Naive Bayes classifier with and without stopwords.

5.5 Conclusions

I have been able to successfully create a naive Bayes classifier, which discriminates between two genres of news with rather good accuracy. For all but one of the test cases, the multinomial method works better. However, most of the optimizations attempted had either no effect or hurt the percentage classified correctly. A potential reason is that with these optimizations, you are eliminating information, and so it doesn't do as well. Thus, the simple, naive Bayes classifier without optimizations is often the most effective–a prototype of computational simplicity.

6 Acknowledgements

I would like to thank my Computer Systems teacher, Mr. Latimer, for his help and support.

References

[1] Boyle, J.A., W.R. Greig, et al., "Construction of a model for computer-assisted diagnosis: application to the problem of non-toxic goitre," *Quarterly Journal of Medicine*, **3**5, 565–588, 1966.

[2] Hand, D., and K. Yu, "Idiot's Bayes: Not So Stupid after All?", *International Statistical Review / Revue Internationale de Statistique*, **6**9, 385–398, 2001.

[3] Korutchev, K. and E. Korutcheva, "Text as Statistical Mechanics Object," *CoRR*, `http://arxiv.org/abs/0810.3416`, October 2008.

[4] Lewis, D., "Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval," *In Proceedings of the 10th European Conference on Machine Learning*, 4–15, Chemnitz, Germany, 1998.

[5] "Matplotlib: python plotting," `http://matplotlib.sourceforge.net/`

[6] McCallum, A. and K. Nigam, "A comparison of even models for Naive Bayes text classification," *In AAAI-98 Workshop on Learning for Text Categorization,"* 1998.

[7] Nerbonne, J., "Linguistic variation and computation", *EACL '03: Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics*, 3–10, Budapest, Hungary, 2003.

[8] "NLTK Home (Natural Language Toolkit), `http://www.nltk.org/`

[9] Rennie, J., "Home Page for 20Newsgroups Data Set," online available at `http://people.csail.mit.edu/jrennie/20Newsgroups/`

[10] Shen, Y., and J. Jiang, "Improving the Performance of Naive Bayes for Text Classification," CS224N, Spring 2003.