Machine Learning for Naive Bayesian Spam Filter Tokenization

Michael Bevilacqua-Linn

December 20, 2003

Abstract

Background

Traditional client level spam filters rely on rule based heuristics. While these filters can be effective they have several limitations. The rules must be created by hand. This requires the filter creator to examine a corpus of spam and cull out characteristics. This is a time consuming process and it is easy to miss rules which are quite effective at detecting spam.
While the word "Viagra" is obviously a very good indicator of spam other non-obvious words are just as good. In the spam corpus I used, which will be described below, the word "webmaster" had a .98% chance of occurring in a spam as did "ff6600", the hex value for a light orange.

Even assuming a good ruleset is created heuristic filters are still a static detection method. As spam changes the rules need to be tweaked and new rules need to be added. Worse yet the rules are the same for everybody that uses a given filter. A spammer can simply run his new spam through popular filters before he sends it out to make sure it will get through.

Statistical filtering seeks to correct these problems. [3] A statistical filter computes the probability that a given e-mail is a spam instead of relying on somewhat arbitrary rules. To do so a corpus of spam and non-spam e-mails is built. Each corpus is then tokenized and the probability that any given token occurred in a spam is computed. Using this information it is possible to compute the overall probability that a new e-mail is a spam in a variety of ways.

While probabilistic filters help to filter spam more efficiently they introduce problems of their own. Definitions for tokenization must still be worked out and this is no trivial task. This paper explores machine learning techniques for tokenization definitions.

Goals

I aimed design and implement two different types of machine learning methods for tokenization. The first was a hill climbing algorithm [4] and the second was a [2]genetic algorithm. As a baseline for comparison I also implemented a filter based on a very simple tokenization method. My goals were to achieve better spam classification, that is more spams detected with fewer false positives (non-spams classified as spams), than the simple tokenization method.

Methods

The filters relied on a probabilistic method known as Bayesian filtering. Bayesian filtering uses a naive version of Bayes rule to compute the overall propagability that an e-mail is a spam assuming that the individual probabilities of each token appearing in a spam are independent of one another.

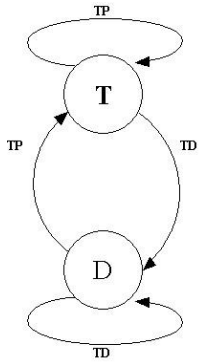

Figure 1: Automata for basic tokenization

Once the probability that any given token occurs in a spam is computed new e-mails must be tokenized in the same fashion. The overall probability that the new e-mail is a spam is then computed using the naive Bayes rule. For two tokens with probabilities a and b the combined probability is computed as $p = \frac{ab}{ab-(1-a)(1-b)}$. The combined probabilities for three tokens with probabilities a, b, c would be computed $p = \frac{abc}{abc-(1-a)(1-b)(1-c)}$ and so on.
In order to compute the spam probability for any given e-mail I used the fifteen most extreme tokens, that is the fifteen tokens whose values deviated most substantially from .5. If all tokens are used longer spams can be missed by the inclusion of many fairly neutral tokens.
The training corpus for the filters consisted of 500 spams and 2,732 non-spams. For the hill climbing and genetic tokenization filters a different corpus of 1,398 spam and 1,401 non-spams was used. Both corpuses can be downloaded from the SpamAssasin's website and contain a wide variety of spams and non-spams.

Baseline Tokenization

The baseline tokenization has two different types of characters, token-pieces and token-separators. The tokenizer simply builds up token-pieces until it sees a token-separator. At that point the token is returned. This tokenization is performed by the simple automata shown in Fig.1.

Token-pieces were alphanumeric characters, apostrophes, dashes and dollar signs. Everything else is a token-separator. Using this definition and a 90% threshold 65% of spams were detected with a false positive rate of 3.4%. The probabilities tended to cluster at the edges. Changing the threshold from 60% to 90% only changed the numbers of spams detected nominally until the threshold approached the higher percentages as can be seen in Fig.2. For the baseline tokenization a threshold of 90% was used.

Hill Climber

The hill climber was the simpler of the two machine learning techniques. It used the same simple automata described in Fig.1. Instead of relying on set classifications of token-part and token-separator for different characters it started out with every character as a token-piece. At each iteration of the algorithm a character was chosen at random and its classification was switched. If this switch helped classifications it was kept, otherwise it was discarded.

The fitness measure was simply the percentage of spams properly detected minus the percentage of false positives. Each character classification was switched only one in every five times it's number came up to prevent the same character from being toggled back and forth by unlucky random numbers.

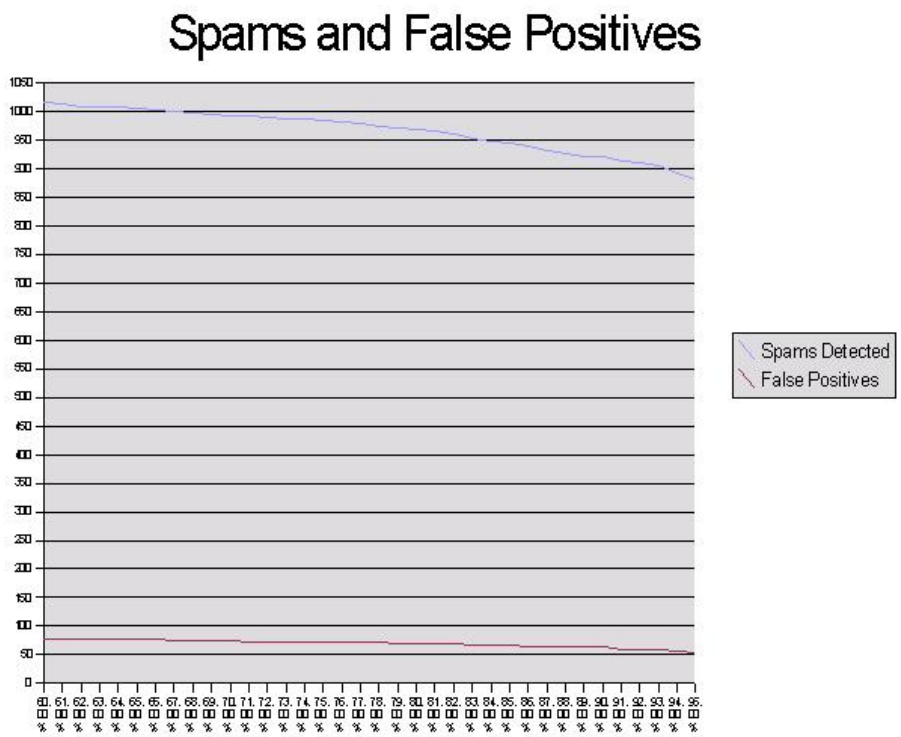

Figure 2: Spams and false positives detected at thresholds $65\% - 90\%$

Genetic Algorithm

The genetic algorithm was somewhat more complicated than the hill climber. Instead of changing the character classifications it changed the automata themselves. Instead of only two different character classifications there were three or more, one and only one of which is equivalent to the token-separator classification

If less than three were used the algorithm would be pointless as any automata could be reduced to Fig.1. For a simple example I will use three character classifications.

A population of twenty-five automata using three character classifications is created at random. Each automata is ranked for fitness using the same fitness classification described above. At each iteration of the algorithm every automata is either mutated or crossed-over with another automata.

Crossing takes one part of an automata and swaps it with the equivalent part of another automata at a given point (Fig.3). Mutation takes one transition in the automata and randomly changes it to go to another state (Fig.4).

At each iteration the automata over the average fitness level of the population were allowed to survive and the weaker automata were destroyed. New random automata were substituted in their place.

Results

Both machine learning algorithms were too slow to accomplish their tasks. Each iteration of the hill climber took roughly a minute and each iteration of the genetic algorithm took almost half an hour.

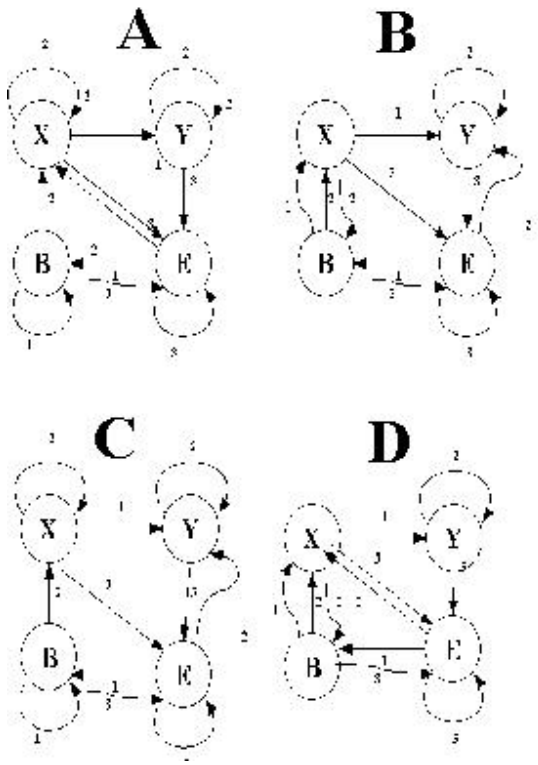

Figure 3: Automata A and B are crossed over at state E to produce C and D

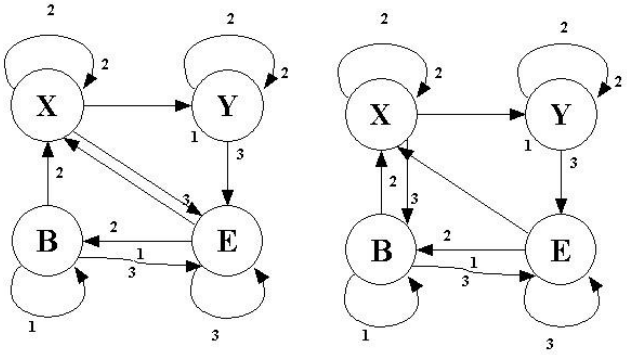

Figure 4: Automata A is mutated to B

The hill climber hit numerous local maxima which required it to backtrack thereby using up even more time. After six hours of running the hill climber had still not achieved its goal and it was not getting any closer because it ran into local maxima after local maxima.
While a faster implementation of these algorithms than mine is certainly possible and a better fitness measure [1] may help with the problem of local maxima, extremely good performance can be achieved with fairly simple tokenization methods. [5] As this is the case tokenization, for the time being at least, should be left in human hands.

Bibliography

[1] Hooman Katirai. Filtering junk e-mail: A performance comparison between genetic programming and naive bayes. 1999.

[2] George Luger and William Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison Wesley Longman, Harlow, England, 1998.

[3] Patrick Pantel and Dekang Lin. Spamcop: A spam classification and organization program. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[4] Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw Hill, New York, New York, 1991.

[5] William Yerazunus. Sparse polynomial binary hashing and the crm114 discriminator. 2003.