

The Bayesian Spam Filter Project

M. B. Zanussi

February 20, 2004

1 Testing Methodology

The following section describes the testing methodology used for the Spam-BGon suite of products:

In setting up the tests for each tokenizer, training was performed in BSFTrain on a random sampling of known, current email. Each tokenizer used the same email sets as the others, with training starting on small email sets and progressing through larger sets after each iteration. There were 16 sets total of spam email and 16 sets total of normal email, comprised of 1, 5, 10, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300 and 325 emails each, respectively. (The training data is located under `./tests/` and has the names `norma` thru `norm1` and `spama` thru `spam1`).

To test performance of the filter, a random sampling of 40 current emails was fed thru BSFTest (it should be noted that none of the 40 emails were part of the training sets, although they were of similar “type”, i.e. related subject matter). 20 were known spam emails and 20 were known normal emails. (The email data is located under `./tests/` and has the names `n01.txt` thru `n20.txt` and `s01.txt` thru `s20.txt`). This set of 40 emails was used with each of the three tokenizers against the aforementioned training data (all 16 sets). Data was collected and organized, and an accuracy rating was calculated and graphed. See Figure 1. (All test results can be found under `./tests/testruns/`).

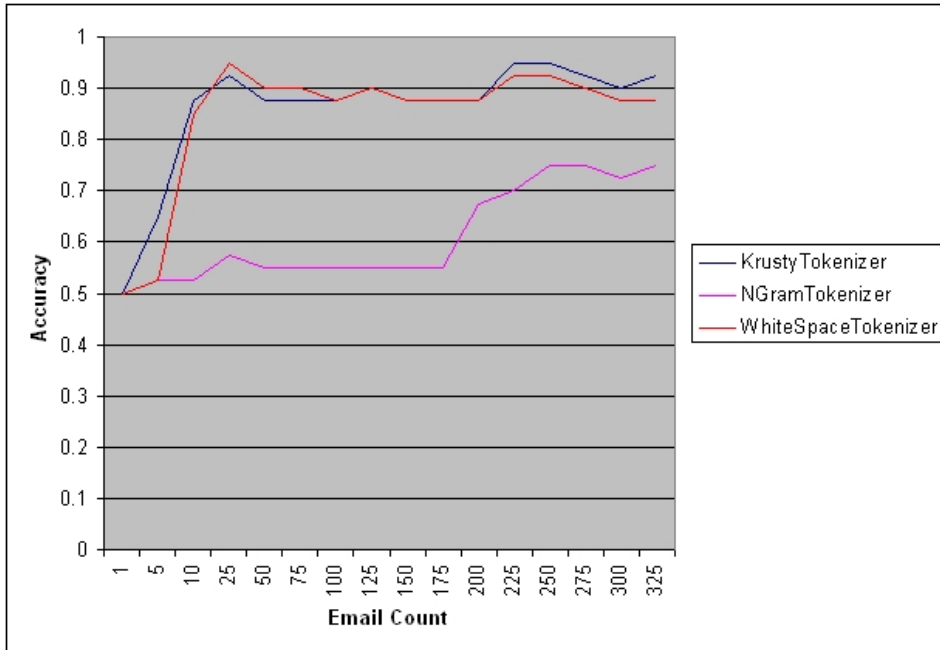


Figure 1: Tokenizer accuracy.

2 Performance

2.1 WhiteSpaceTokenizer

The WhiteSpaceTokenizer splits the analyzable section of an email message at whitespace characters and discards any and all punctuation. It's a fairly simple tokenizer and generally produces a smaller token total than other tokenizers (see Figure 2) and also tends to produce tokens of much greater length, and that can make it somewhat difficult to produce consistent spam hits. Overall, the WhiteSpaceTokenizer had an accuracy rate of approximately 84.53% for the entire test run. Starting at about 200 emails in the training data, the rate improves to around 89.58%. For the sample sizes involved, I would consider this a decent rate, but not that great, and probably won't increase that much more with further training as it appears the rate is leveling off.

2.2 KrustyTokenizer

The KrustyTokenizer is based on the WhiteSpaceTokenizer but introduces an important distinction: rather than discarding all punctuation, it treats punc-

tuation found in HTML tags and attributes as delimiters, just as whitespace would be treated. So, for example, given the following HTML in an email,

```
<br><br>
```

under WhiteSpaceTokenizer, the tokenizer would break up the input into the following tokens:

```
brbrimg  
srchttpwwwiampammercomeatmejpg
```

Those are two fairly unique tokens, and really doesn't say much about what the email contains. They might prove difficult to match up and produce hits with other email passed to the tester. On the other hand, under KrustyTokenizer, the same line of input produces the following "more useful" tokens for classifying spam:

```
br  
br  
img  
src  
http  
www  
iampammer  
com  
eatme  
jpg
```

Here, KrustyTokenizer is creating more tokens from the input (from Figure 2 one will notice that the total number of tokens have increased significantly) and the HTML is much more obvious. Under this scenario, more hits should occur within the spam data as more HTML tags are recognized, and thus should decrease the number of spam that get thru the filter.

Indeed, that appears to be the case as KrustyTokenizer improves on the accuracy rate of WhiteSpaceTokenizer (see Figure 1). Overall, it had an accuracy rate of approximately 85.94% for the entire test run. Starting at about 200 emails in the training data, the rate increases to about 92.08%. Not a huge difference to be sure, but it does suggest that this type of tokenizer might be better in the long run than plain old WhiteSpaceTokenizer. Much

like WhiteSpaceTokenizer, the rate seems to be leveling off and it doesn't appear it would gain much more advantage with a lot more training.

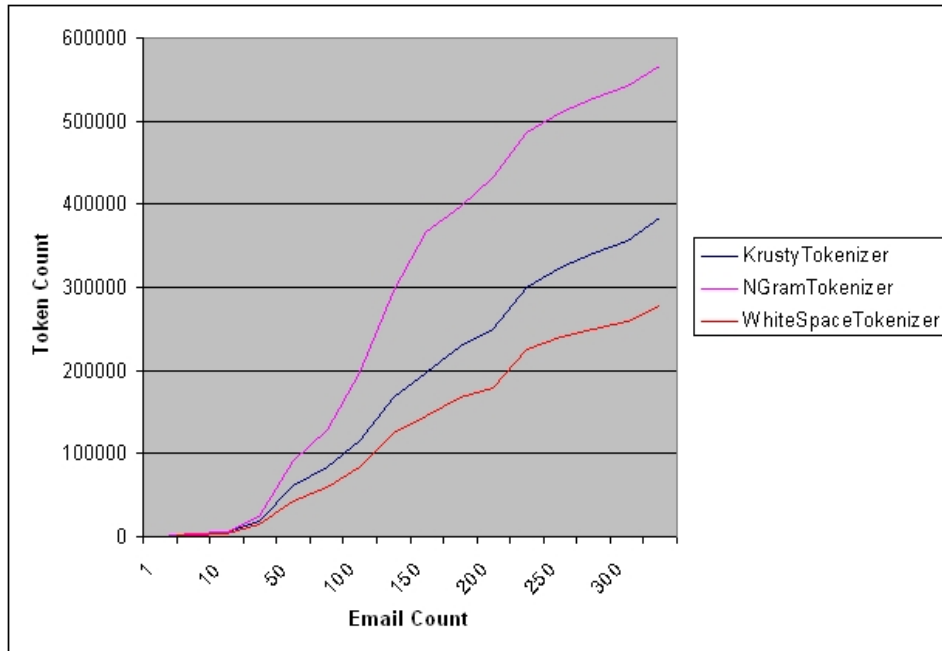


Figure 2: Total token counts for each tokenizer.

2.3 NGramTokenizer

The NGramTokenizer splits the analyzable section of an email message into tokens of n contiguous characters, where n is a value greater than 0. Whitespace and punctuation are ignored in this tokenizer. For the testing here, an n of value 5 was used. Unfortunately, the results were quite poor. Overall, NGramTokenizer had an accuracy rate of about 61.09%, far below that of the other two tokenizers. At 200 emails, the rate did increase to 72.50%, and from Figure 1 it appears that the trend is continuing upward. But really, up until about 200 emails, the rate is abysmal. Until this model has been trained with enough data, NGramTokenizer simply seems to produce a lot of tokens that don't recognize much of what's thrown at it and thus produces poor accuracy.

Since the tokenizer is creating tokens of 5 characters in length all the time, the total number of tokens produced was quite large, as evidenced

by Figure 2. In fact, the real downside to this tokenizer is the amount of memory it chews up. Whereas the other two tokenizers created statistical models of about 1.5 MB each for 325 emails, NGramTokenizer's model was a hefty 8 MB plus. Saving and loading files of this size is not that fast (due to apparently not-so-fast serialization and the use of object streams).

3 Improvements

Obviously, there's always room for improvement. Here are just a few ideas I can think of:

1. **More Training** - It could be that that NGramTokenizer is the greatest thing since sliced bread, or that KrustyTokenizer is really bad at its job. Running much more email through the trainer would clarify such things.
2. **Better Algorithm** - Sure, the naïve Bayes approximation as implemented here works, but it might not work good enough. Paul Graham in *Better Bayesian Filtering* suggests a different approach (which produces wildly accurate filtering) that perhaps could be explored in SpamBGon.
3. **Serialization** - Although it doesn't affect the overall results, the serialization as implemented here just isn't working efficiently. Sticking to my original idea of using two separate models for normal and spam would help, but the bottleneck is really in the loading and saving of the models. I'm not sure serialization is good for spam filtering, what with the amount of training required.
4. **Better Tokenizers** - KrustyTokenizer, for example, shows some promise, and perhaps extending it to treat things like Base64 encoding (a tokenizer hairball) differently or keeping track of images or URL references to further filter on would be useful.
5. **Blacklists & Whitelists** - Verifying address and domains right off the bat would be an improvement, as well as being able to add to such lists after finding new spam.