Parameterization of Naïve Bayes for Spam Filtering

Trevor Stone
Department of Computer Science
University of Colorado at Boulder

Masters Comprehensive Exam

Fall 2003

Abstract

In this paper, I present the results of applying the Naïve Bayes algorithm to the problem of filtering unwanted junk or “spam” e-mail. I explore the effect of several parameters including corpus size and feature extraction methods. I compare my results to several published statistical spam-filtering approaches.

1 Introduction

Spam, also known as “unsolicited commercial e-mail” or “junk e-mail,” pollutes the once serene medium of electronic mail. Left unfiltered, recipients of spam must waste time deleting annoying and possibly offensive messages. When a user is inundated with a large amount of spam, the chance of overlooking a legitimate message increases. Spam also creates a burden on mail servers and Internet traffic, all for unwanted messages. Spam accounted for over half of the mail I received this year.

Researchers have examined several different approaches to spam filtering, many of which are widely deployed. One broad approach attacks spam at the network level. For instance, some servers block spam based on so-called Real-time Black Hole lists, which alert servers of undergoing spam flurries. Hall has achieved decent results with a similar approach.[11] Another broad approach examines the content of an incoming message for features which indicate its status as spam or legitimate. Deployed at the user level, this approach can incorporate facts about each user’s legitimate mail.

Taking the latter approach and treating e-mail filtering as a text classification problem, researchers have applied several statistical learning algorithms to e-mail corpora with promising results. (Including problems more difficult than spam filtering, such as author identification.[4]) However, their tokenization and feature extraction methods produce truncated feature lists and don’t take into account many pieces of potentially helpful information. In this paper I examine the performance of a Naïve Bayes classifier using varied approaches to feature extraction and tokenization as well as different corpus sizes. I also compare my results with those produced by a popular ad hoc filtering program.

The paper is organized as follows. In Section 2, I briefly review the published spam filtering results. In Section 3, I present the Naïve Bayesian classifier in the variants I use for my experiments, providing arguments for several of the design decisions. In Section 4 I describe the corpus of my e-mail. In Section 5, I present the tokenization schemes I use for feature extraction. In Section 6, I present the experimental results of my classifier’s performance on a month of my mail, using both a large and a small training corpus. Section 7 presents possible future work.

2 Related Work

Computerized text classification problems and solutions date back to the early 1960s when Naïve Bayes was first used in this domain.[14] To apply statistical learning methods to text classification problems, algorithms break documents into sets of features. Typically most of these features are tokens – usually words in the traditional sense. Additional features such as the author’s name, the document’s creation date, or the document’s length, are sometimes collected.

In contrast to many statistical learning problems, classification of natural language texts is not well defined in several respects. First, there isn’t always a clear way to determine what to count as a feature. For instance, in some cases punctuation and capitalization carry significant information while in other cases they merely clutter the vocabulary. Second, the vocabulary is not fixed. Documents under consideration may have tokens (such as misspellings, unusual, and new words) which were not present in the training data. Furthermore, documents are usually sparse, containing far fewer vocabulary words than they leave out. Obtaining enough training data to calculate reasonable probabilities of infrequent words is impractical at best. Third, while documents are usually rich in semantic information, extracting it is not easy.[12]

Researchers have applied a variety of statistical learning methods to the problem of classifying e-mail as legitimate or junk. In an early attempt, Sahami, Dumais, Heckerman, and Horvitz[19] applied the Naïve Bayes approach to the problem. Using a corpus with far more junk than legitimate mail, their best binary-classifier legitimate precision and recall figures (87.8% and 94.7%) came by considering as features the 500 words with highest mutual information along with 35 hand-crafted features. Multiple occurrences of a single feature in a given message were ignored. They looked for word tokens only in the subject heading and the message body. They achieved better legitimate-class results (96.2% precision and 100% recall) by splitting the classifier three ways and splitting the junk messages into two separate classes, though the junk subclassification was rather poor.

Androutsopoulos et al.[1] produced a widely-used public spam corpus, composed of 2412 legitimate messages from a linguistics mailing list and 481 spam messages received by Androutsopoulos. Their best results were 99.47% spam precision and 78.41% precision, achieved with stemming and a stop list and cost function $\lambda = 9$. However, all tokenization schemes with cost $\lambda = 9$ (including neither stemming nor stop lists) produce similar results.

Drucker, Wu and Vapnik[7] approached the spam filtering problem with the support vector machine technique. They extracted words from the subject and body from a corpus of messages from different message recipients. Stemming was applied to the words, collapsing words with the same root to a single token. Two vocabulary lists were then generated, with and without a stoplist, before being pared down to the 1000 best words. Their best results came when considering words from both the subject and the body without a stoplist. Their legitimate recall for this case was 100% recall and the precision was 95.3%.

Katirai[13] compared the performance of genetic programming and Naïve Bayes classifiers in the spam filtering domain. The training set for this experiment had seven times as many junk messages after removing duplicate spam. The corpus was tokenized into words, applying both stemming and a stop list. The best results showed genetic programming with junk precision of 95.45% and recall of 70% while Naïve Bayes stood at 95.83% precision and 76.67% recall. Katirai also noted that if certain regular punctuation sequences (such as signatures and boundaries of forwarded messages) are removed, repeated punctuation is an information-rich feature.

Diao, Lu, and Wu[6] compared the Naïve Bayes approach to a decision tree classifier in the e-mail classification domain. Their goal was a little more broad than simple spam filtering; they classified messages as either “interesting” or “not interesting.” The latter class included both typical spam and otherwise legitimate messages which the user didn’t care for. Furthermore, they combined such classified messages from several different users. This pair of approaches isn’t very practical, since one message may be very interesting to one user and entirely uninteresting to another. They varied the inclusion of textual features (stemmed and stop-listed alphabetic strings), structured features (header information such as the sender’s domain or the presence of attachments), and several handcrafted features, such as the number of exclamation points. The best results for both decision trees and Naïve Bayes were obtained by using both header and body features.

Carerras and Màrquez[2] compared the performance of the AdaBoost algorithm on LingSpam to previously-reported results on LingSpam for Naïve Bayes, k-Nearest Neighbor, Decision Trees, and Stacking. Using cost-sensitive classification, the best junk precision/recall results are as follows: Naive Bayes: 99.45/77.57%; K-Nearest Neighbor: 98.8/81.9%; Stacking: 98.8/84.8%; Decision Trees (not cost-sensitive and on a different corpus): 88.71/89.81%; and AdaBoost: 99.35/94.8% (or 100/81.91% with a higher cost function). The features were not stemmed or stoplisted, and presumably were taken just from the body and with a binary feature model.

Sakkis et al.[20] experimented with several parameters for a K-Nearest Neighbor classifier on LingSpam. Their best results were 99.45% spam precision and 67.57% spam recall. They also found that increased training corpus size and increased dimensionality improved the performance of the classifier.

Gee[8] compard Latent Semantic Indexing to a previously published Naïve Bayes result on the LingSpam corpus. The best Latent Semantic Indexing results (which actually came from two separate configurations) were 99.88% legitimate precision and 99.79% legitimate recall. This compares favorably with the Naive Bayes results of 100% and 98.3%, respectively.

O’Brien and Vogel[18] compare Naive Bayes to the “Chi by degrees of Freedom” approach. The latter is often used in author detection, and is used under supicion that most spam is sent by a small number of prolific profiteers. They also compared the effectiveness of both classifiers working on words or characters as features. Naive Bayes with words produced spam precision of 100% and recall of 76.9%, while operating on characters produced 100% in both categories (on a test corpus of less than 70 messages). The Chi approach produced 100% precision and recall when operating on words and 97.5% precision/100% recall working with characters.

Manco, Masciari, Ruffolo, and Tagarelli[15] developed a system to classify incoming messages as “interesting” and “important” (or neither) in an unsupervised manner. Features were stemmed and stoplisted words from the body and several header-based features like sender domain. The best results for this K-Means algorithm presented in this case has recall of 90.59% and precision of 73.48%. While these results are significantly worse than those listed above for spam, the problem is more difficult and the classes more similar.

Graham posted an influential article on the web[9] presenting results of 0 false positives (updated later to four out of several thousand messages) and false negative rates near 1/1000. In his updated version,[10] he advocates using all header information, considering trailing punctuation part of the attached tokens, and adjusting for case only when exact case matching fails. His approach offers several advantages over those mentioned above. First, very little information is discarded. When classifying, he only considers tokens which have been seen at least a few times before, but his vocabulary can grow dynamically and indefinitely. While some features will be examined infrequently, when they are used, they are often highly indicative of one class or another. Furthermore, using full header information can expose features automatically created in the process of e-mail sending; this information is difficult to spoof. Finally, his corpus is just the set of messages he received over an extended period. This makes the classification task more realistic (users typically receive messages on a wide variety of topics) and allows specific header information to be used (since information like the receiving host will be normalized).

3 Approach

While several algorithms mentioned in Section 2 outperform it, Naïve Bayes has several advantageous properties. First, a classifier is constructed by a single sweep across the training data and classification requires just a single table lookup per token, plus a final product or sum over each token. Other approaches like Support Vector Machines, Boosting, and Genetic Algorithms require iterated evaluation; approaches like k-means require several pairwise message comparisons while decision tree building is significantly slower than Bayesian table construction. The quick training and classification times of Naïve Bayes allowed several variants to be tested. Furthermore, since Naïve Bayes need only store token counts, rather than whole messages, storage requirements are small, the classifier can be updated incrementally as individual messages are classified, and classification data can be shared between users without (much) loss of privacy.

I implemented four variants of the Naïve Bayes text classification algorithm. Each makes the independence assumption that the probability of tokens occurring in a message is independent, since the exact probability of a message's occurrence is very difficult to compute. In practice, this assumption does not hold, but history is replete with successful examples.[14] In all four variants I also ignore the prior probabilities of the two categories. Spam accounts for a little more than half of my mail, so the prior probabilities would thus be close to 0.5 and cancel each other out. Incorporating a high prior spam probability would also be undesirable: misclassified legitimate mail is much worse than misclassified spam and the chance that a user will notice a misfiled message shrinks as the volume of spam grows.

The first variant follows [17]. The message is classified with the label *lbl* which maximizes the product of $P(a_i|lbl)$ as i ranges over all tokens where

$$P(t|lbl) = \frac{t_{lbl} + 1}{n_{lbl} + |Vocabulary|},$$

t_{lbl} is the number of occurences of token t in messages with label lbl, n_{lbl} is the total number of token occurences in messages labeled lbl and $|Vocabulary|$ is the number of unique tokens across all messages. This variant counts each occurrence of a token separately. A slight variation on this algorithm collapses multiple token occurrences; a repeated token is only included once in the product; t_{lbl} is the number of lbl labeled files containing t and n_lbl is the number of files labled lbl. To prevent machine underflow, the final multiplication is replaced by a sum of logs; this conversion maintains ordering.

The other two variants are based on Paul Graham's popular implementation [9] and [10]. This implementation takes advantage of the Binary Independence Model[14] and merely calculating the probability that the message is spam and classifying it as such if that probability is greater than 0.5. The second major difference in Graham's approach is to only use the probabilities from the "most interesting" tokens – those for which the probability of spam is furthest from 0.5. In this implementation,

$$P(spam|t) = \frac{t_{spam}/n_{spam}}{\frac{t_{spam}}{n_{spam}} + \frac{t_{legit}}{n_{legit}}}$$

where t_{lbl} is again the number of appearances of token t in messages labled lbl, n_{lbl} is the total number of tokens in lbl messages. In cases where t only appears in one class of messages (and hence produce a 0 probability for the other), the probability from that token is estimated to be $0.01/(n_{lbl}/10+1)$ (or one minus that quantity, as appropriate). As before, I also tested a variant where multiple occurrences of a single token in a message are ignored.

Note that Graham's approach calculates a different quantity than the standard Naïve Bayesian algorithm.

$$\frac{t_{spam}/n_{spam}}{\frac{t_{spam}}{n_{spam}} + \frac{t_{legit}}{n_{legit}}}$$

is $P(spam|t)$ — the probability that a message is spam given that it includes a particular token. It then multiplies a subset (in my case, 20) of those probabilities to estimate $P(spam|message)$. If all tokens in the message were multiplied together a very long message with lots of spam-indicative tokens would be classified as legitimate, since the fractional multiplications would bring the product below 0.5. The traditional algorithm, on the other hand, calculates $P(t|class)$ – the probability that token t will occur in a message of type class. All of these probabilities are then multiplied. There is no need to ignore neutral tokens, as their multiplicative effect is equal on both classes.

In all cases, I ignored all tokens which appeared in only one file; such tokens are often randomly-generated or time-stamped unique identifiers.

4 Corpus

E-mail messages are semi-structured text documents. Plain messages have a header section and a body section. The header contains several optional and required fields, some of which follow set formats (like the `Date` and `Received` fields) while others (like the `Subject` and `X-Mailer` field) can contain almost any text. The body is essentially unrestricted in its content. Using the MIME protocol, several parts or "attachments" can be included in a single body. This allows messages to include encoded multimedia, or alternative textual presentations like HTML. Each MIME part includes a header describing the type content, encoding method, and so forth.

The corpus for this experiment consisted of 13911 messages I received in the thirteen months from October, 2002, to October, 2003. 6891 (49.6%) were hand-classified as legitimate, 7020 were spam. The data may be noisy; during development I detected a few misclassified messages in the test set (which were subsequently reclassified). This underscores the importance of a filtering method robust to misclassifications and noise.

This is not all of the mail I received in that period – several mailing lists were automatically filed into their own folders and not included in the corpus (though some low-volume or important mailing lists arrive in my inbox and hence were included). Filtering mailing list traffic is a slightly different problem which has some different solutions available (such as moderating messages sent to the list).

I considered as spam all unsolicited mail sent without regards to my identity. Some unwanted commercial email was classified as legitimate. For instance, I considered solicited any mail resulting from signing up for services or purchasing products over the World Wide Web. In my experience such companies usually honor requests to be removed from their mailing lists, and the messages occasionally contain material of interest. I also considered legitimate unsolicited commercial messages which indicated that the sender knew something interesting about me, such as my city of residence, group membership, first name, or semantic website content. While such messages are often unwanted, I'm more likely to be interested in them and I think I ought to respond to personal effort with at least a personal glance at the subject. Such messages tend to be few, since they require human effort, and are less likely to be fraudulent than typical spam. For this purpose, I don't consider machine-determined information like my username or website URL to be interesting information about me.

I pre-processed the messages in several ways.

- I receive e-mail sent to several addresses, some of which have skewed percentages of spam or legitimate mail. For this experiment, I replaced all occurrences of these addresses (in either the header or body) with a single address. In practice, using unique e-mail addresses for different correspondents (such as signing up for an online service) can be detected with simple hand-crafted rules to automatically file mail without applying a statistical filter. On the flip side, a filter which can extract the few legitimate messages out of the sea of spam sent to an address used to, say, post to Usenet would be quite useful.

- My mail server runs a virus checker. While e-mail viruses and spam share many features in common, filtering viruses is a separate (easier) problem from filtering spam. All virus-laden emails were kept out of the corpus (save an overlooked one or two). Also kept out were mistaken warning messages claiming I sent a virus when it was just someone using my address in the `From:` header. Messages were also cleaned of the header indicating the messages were virus-free.

- My mail server also runs SpamAssassin[5], which tags suspicious incoming messages with a spam label and diagnostic information indicating which spam-indicating rules the message met. All of this information was scrubbed from messages in the corpus, but SpamAssassin's classifications were noted and used for comparison in Section 6.
- I removed annotations made by my email client, such as a `Replied:` header which includes the time I replied to a message. Such information would obviously not be available in incoming mail.
- MIME Attachments that could be identified as text (including those listed with a content-type of `application/octet-stream`) were retained, converting them from an encoded form like base64 if necessary. Non-text attachments like images and application-specific data files, were removed. All MIME header information, such as the attachment's content-type and file name, was retained as text.
- Finally, headers incurred by collecting the data, such as `Received:` headers indicating the path from a mail server where I forward my mail to another, were removed.

The corpus was organized into three groups. All messages received in October, 2003 (1200 spam and 809 legitimate) were used as a test set. This set was tested with two different training sets, one with mail from October, 2002, through August, 2003, and another with only mail from September, 2003. The former contains 5194 legitimate and 5011 spam messages, the latter 889 legitimate and 809 spam. I split the training sets in this way to examine the effect of corpus size. A new filter user is unlikely to have a year-long backlog of spam and non-spam e-mail to seed a filter. Thus, ideally a small training time could produce a highly effective classifier. Strong performance by the latter training set may also be attributable to recency effects. Many e-mails are sent as responses to previous messages (of my 6984 legitimate messages, 1164 had an `In-Reply-To:` header) and often include quoted portions of previous messages in the "conversation." Subsequent messages may thus be easily detected by the sender's names and e-mail addresses or quoted terms. Thus, a recent training set will likely perform better than an older set of equal size.

5 Tokens

As I indicated in Section 2, most published spam filtering systems break messages into features in fairly simple ways. Notably, little header information beyond the subject is examined in most research systems. Fields such as `From:` can be information-rich, picking out the addresses of legitimate correspondents while fields like `To:` can help identify spam; I receive a lot of spam which is also addressed to people on the same system with alphabetically adjacent usernames. It's also possible that other headers contain valuable information; spam and legitimate messages might typically be sent at different times or from different time zones. To determine how helpful different parts of the message are, I created four training sets from the corpora. One set (labeled "body") tokenized just the body. Another set (labeled "rawhead") used just the header potion and treated it like body text. A third set ("combined") combined the first two, treating the entire message as a string of tokens. The fourth set ("head") used the structured nature of the headers. It treats as a feature each field/token pair. Thus, in the fourth set, the token "Mary" in the `To:` field is treated separately from the token "Mary" in the `From:` field. Unlike some of the researchers mentioned in Section 2, I didn't make any use of deeper semantic information contained in the headers just field/token combinations. A production system might find extra semantics (such as the range of time during which the message was sent) a useful optimization.

Most researchers provide little justification for their tokenization technique, which is usually alphabetic words in the subject and body, often stemmed or filtered with a stop list. As people have deployed filters, spammers have developed unusual ways of conveying their information. For instance, sexual pill spams have included the strings `Vviagra`, `Via.gra`, `V1agra`, and `V|agra` in the subject line. While these can get past simple hand-made rules, proper tokenization will recognize them all. `V1agra`, for instance, occurred 7 times, all spam. A tokenizer which recognized `via.gra` as a single token might outperform one which split it into tokens `via` and `gra`. Alternatively, it might not; my 11-month training corpus has 1 legitimate and 119 spam occurrences of `Via`, 1 legit and 164 spam of `gra` while `via` occurred 609 times in legitimate mail and 223 times in spam.

Punctuation alone may be elucidating. Say and Akman illustrate several beneficial uses of punctuation in computational linguistics in [21] and [22]. Several researchers mentioned in Section 2 have treated excessive exclamation marks as a feature. My 11-month corpus had 63 spam messages and 121 legitimate messages with the string `!!!` while it had 151 spams and 37 legitimate messages with `!!!</` (something exciting at the end of an HTML tag). The amount of whitespace may even prove interesting. 61 spam and 30 legit messages followed `!!!` with one space while the ratio was 8/26 for `!!!` followed by two spaces.

To explore these issues, I tested a variety of tokenizers, each in both an "ignore case" and a "retain capitalization" mode. The most simple (labeled "character") merely treated each character as a separate token. `!` was nearly four times more common in spam than legitimate email while `,` was almost twice as common in legitimate mail as spam. Since many characters appear in almost each message, I also implemented a tokenizer ("twochar") which treated each character pair as a token. `zx` occurred in 15 times as many spams as legitimate messages while `LJ` was in five times as many legitimate messages as spam.

Another tokenizer I used ("alphanumeric") was fairly typical, treating all strings of consecutive alphanumeric characters tokens and ignoring all other strings. Any non-alphanumeric character (rather than just whitespace) was used as a token separator, so this method picks up portions of hostnames (`com` appears in almost all spam messages) or URLs (`cgi` is five times more common in spam). Some production systems have ignored strings of all numbers, I included them. `000099` (an HTML code producing blue text) had a 203/6 spam/legit ratio while `001` (indicative of course numbers, for instance) had a 78/233 ratio and `0700` (my time zone) is nearly twice as common in legitimate mail.

To examine what can be gleaned from punctuation, I also implemented a tokenizer ("alphanumpunc") which treated all alphanumeric strings as tokens and all non-alphanumeric strings as tokens. This tokenizer could probably be improved, as it treats, for instance, 10 spaces as a separate token (with a 265/138 spam/legit ratio) than 10 spaces followed by a carriage return (a 4/0 ratio). However, despite its high incidence of infrequent tokens, it picks out a host of features with high information content. `...` separating two alphanumeric strings occurs with a 344/929 ratio while `...` followed by two spaces has a 1/78 ratio.

The final token scheme ("basic") combined punctuation and alphanumeric characters. In this case, token boundaries were whitespace with optional adjacent non-alphanumeric characters; the characters `@`, `=`, `<`, `>`, `"` with optional adjacent non-alphanumerics; and any number of `/` characters preceded by an optional `:`. In most normal cases, this tokenizer behaves much like the alphanumeric classifier — words are tokenized as such, since leading and trailing punctuation is usually whitespace adjacent, and so gets ignored. Unlike the simple alphanumeric tokenizer, however, this tokenizer picks up on whole domain names and prices (rather than segments), hyphenated words (especially common in headers such as `Content-Type`, which is twice as likely in spam), and words like "don't". This tokenizer also has room for improvement; one common spam technique is to insert html comments with random strings to break up tokens, e.g.

```
Fo<!--ek76d92dg24j-->r th<!--via2d38nm5z-->e fir<!--645zpu3ltznh13-->st
tim<!--azwjt21617-->e ev<!--5x5cbphzdv3-->er
```

This tokenizer doesn't isolate `!--` as a separate token (occurring 13 times more often in spam messages).

6 Results

In this section, I present the results of two sets of tests. One set used a small training set built from messages received in September, 2003. The other set used a training set from messages dating from October, 2002 to August, 2003. The test set in both cases was messages received in October, 2003.

Several important numbers are reported in the results. The most important number is false positives — that is, legitimate messages classified as spam. The cost of an individual false positive varies, since some messages are less interesting than others. However, a missed e-mail can potentially cause a user material loss or emotional grief. It is therefore imperative for the number of false positives to be as low as possible, ideally 0. The converse problem, false negatives, are much more benign. When a user receives a false negative, he just needs to mark it as spam so the system will better recognize its ilk and then delete the message. Most if not all users would rather receive several pieces of spam than lose one legitimate message. The rates of false positives and negatives are also reflected in the precision and recall values. Precision (based on false negatives) is the percentage of messages classified as legitimate which truly legitimate. Recall (based on false positives) is the percentage of truly legitimate messages classified as such. Note that below I always present precision and recall figures for legitimate messages, even though a "yes" classification denotes spam.

6.1 Small Training Set

Using a training set built from messages received in September 2003 I applied to the October 2003 test set all combinations of algorithm variant, token type, message portion, and case sensitivity. Table 1 shows a ranked sample of the combinatorial results. The table includes all combinations resulting in fewer than 3 false positives as well as the best-performing combination for each variable instance.

Table 1 shows several interesting results. First, all good approaches I tried outperformed the precision of SpamAssassin by at least 19% and no classifier had worse precision. This may indicate that personalized filters are a big win or that SpamAssassin would benefit from more token knowledge. (One of SpamAssassin's "rules" is a Naïve Bayes algorithm, but it obviously doesn't have the benefit of the user's legitimate corpus.)

Second, all of the classifiers with two or fewer false positives used the "combined" (body + head as unstructured text) portion. This isn't surprising, since that has the most information available to it. Several classifiers that looked only at the body had 5 or fewer false positives while 11 classifiers using just structured headers had fewer than 20 false positives, but only one rawhead classifier had fewer than 20. On the other hand, the best false negative rates were produced by looking just at the headers, with both structured and raw classifiers consistently missing just 20–40 spam. This indicates that combining the unstructured body with structured headers may produce a classifier with minimal false positives and very low false negatives.

One notable absence from the table above is character-based tokens. The best performance from character bigrams resulted in 19 false positives while the best classifier using single characters as tokens weighed in with 44 false positives. These results are somewhat surprising, since one might intuitively expect filtering based on character counts to perform little better than chance. While these results are clearly insufficient to be used as the sole basis for a filter, they may still prove useful as a component in a filter. Character bigrams might, for instance, be collected from many users and used to form initial training sets for new filter users. This would maintain user privacy while making it somewhat

Table 1: Performance on the small training set

alg	token	portion	case	falsep	falsen	prec	recall
pgt	anp	com	yes	0	62	92.9%	100%
pgf	anp	com	yes	0	68	92.2%	100%
pgt	anum	com	yes	0	68	92.2%	100%
pgf	anum	com	yes	0	71	91.9%	100%
pgt	anum	com	no	0	128	86.3%	100%
pgf	basic	com	yes	1	69	92.1%	99.9%
pgf	anp	com	no	1	71	91.9%	99.9%
pgf	basic	com	no	1	81	90.9%	99.9%
pgf	anum	com	no	1	89	90.8%	99.9%
pgt	basic	com	no	1	125	90.9%	99.9%
pgt	basic	com	yes	2	75	91.5%	99.8%
pgt	anp	com	no	2	99	89.1%	99.8%
sa	-	-	-	2	339	70.4%	99.8%
pgf	basic	body	yes	3	94	89.6%	99.6%
mlf	anum	com	no	5	64	92.6%	99.4%
pgt	anp	head	no	11	44	94.8%	98.6%
mlf	basic	rawhead	no	16	39	95.3%	98.0%
mlf	twochar	head	yes	19	55	93.5%	97.7%
mlt	basic	com	yes	21	47	94.3%	97.4%
pgt	char	head	yes	44	29	98.7%	94.6%

Symbol explanations:

alg = algorithm variant:
pgt = Graham-based with integer features
pgf = Graham-based with binary features
mlt = standard with integer features
mlf = standard with binary features
sa = SpamAssassin

token = tokenization method:
anp = alphanumpunc
anum = alphanumeric
char = character

portion = portion of the message used for features:
com = combined

case = does different capitalization mean different tokens?
falsep = number of false positives
falsen = number of false negatives
prec = legit precision
recall = legit recall

difficult for spammers to insert known legitimate tokens.

All three word-based tokenizers perform competitively, since they share many tokens. However, the two that use punctuation in tokens give slightly better precision. This indicates that production filters should consider examining punctuation for interesting tokens. This might even be a good place to employ character or character bigram counts. Also note that most classifiers did approximately as well using and ignoring case. This indicates that capitalization information typically doesn't buy much, but again can shave a few errors down. Graham [10] suggests first checking a token with capitals included and, if that doesn't match an existing token, convert it to lower case and use that as the token.

Finally, Graham's algorithm [9], which examines only the most informative tokens, is shown to be a big winner over the traditional approach which multiplies probabilities of all tokens together. This is likely due to the presence of many tokens which are each slightly more likely in spam which, when multiplied together, outweigh single damning tokens. This is supported by the observation that the keeping the top classifier constant, save changing the algorithm to the standard, the token version reports 30 false positives while the one-token-per-file approach produces only 9. Binary- and integer-token count versions of Paul Graham's algorithm typically produce similar results with the former typically correctly classifying a few more messages. The counterexample to all of this is, of course, the character-based tokenizers. Binary-token counts are often insufficient for the character token case, since most emails contain at least one copy of most characters. The traditional "multiply everything" approach also serves the character and twochar cases well, since 20 tokens pales in comparison to the average 4800 characters per message.

The algorithms do have some room for parameter adjustment. The threshold in Paul Graham's can be set arbitrarily. I set it at .5, but a most of the false positives in the top classifiers could be saved by raising the threshold to .6.

6.2 Large Training Set

Again, I present the top-performing classifiers in Table 2.

This table presents some interesting results. Compared to the small training set, the large set produced a few more classifiers with two or fewer false positives. (Not shown above are 11 classifiers with two false positives.) Several of the classifiers with one false positive failed on a particular message, sent via text messaging from Japan (a country from which I'd previously received mostly spam). This message contained two previously-seen all-caps words and was a reply to a message I sent, though I hadn't yet received any messages from this user at this address. It could, however, be caught by keeping track of addresses to which I'd sent e-mail and allowing responses to those. Other 1-miss classifiers tripped up on a message that looks a lot like spam. Not only does it advertise via bulk email, it's commercial (it's a message to college football ticket holders informing them of college basketball ticket prices), and it is full of HTML and JavaScript code.

Table 2: Performance on the large training set

alg	token	portion	case	falsep	falsen	prec	recall
pgf	basic	com	no	0	120	87.1%	100%
pgt	anp	com	yes	1	82	90.8%	99.9%
mlf	anum	com	yes	1	96	89.3%	99.9%
mlf	anum	com	no	1	111	87.9%	99.9%
pgt	anp	com	no	1	119	87.2%	99.9%
mlf	anum	body	yes	1	147	84.6%	99.9%
pgf	basic	body	yes	1	150	84.3%	99.9%
pgf	anp	body	yes	1	164	83.1%	99.9%
mlf	anum	com	no	1	191	80.9%	99.9%
pgf	anp	body	no	1	200	80.2%	99.9%
pgf	basic	body	no	1	211	79.3%	99.9%
sa	-	-	-	2	339	70.4%	99.8%
mlf	basic	head	yes	3	43	94.9%	99.6%
mlf	char	head	yes	3	176	89.4%	99.6%
mlf	basic	rawhead	yes	7	30	96.4%	99.1%
mlf	twochar	com	yes	14	85	96.4%	98.3%
mlt	basic	com	yes	21	36	95.6%	97.4%

Symbols are explained under Table 1.

The most notable feature of Table 2 is the dramatic drop in precision. The top classifiers have two to three times as many false negatives as the small test. This could be due to human error in classifying the large corpus or it might be due to temporal smoothing — the probability of receiving legitimate mail with spammy tokens increases as more messages accrue. Alternatively, new spam techniques and popular spam products may have developed recently, so a recently trained filter will have new tokens to work with which are likely to occur in newly-arrived mail. Since a large corpus seems to increase recall while a small recent corpus seems to increase precision, a production system might include both, consulting one when an incoming message is on the borderline.

Another striking feature is that the character tokenizer's best performance gave only three false positives; the second best character/head-based performance produced 4 false positives and 201 false negatives. Thus, with enough training data, most legitimate messages can be identified purely based on the distribution of characters in header fields. Since many of these fields are set automatically in the e-mail process, it's difficult for spammers to spoof this information to masquerade their mail as legitimate.

With a large corpus size, the standard Naïve Bayes implementation (with tokens either on or off on a per-file basis) produces results comparable with Paul Graham's implementation. A larger training set provides a more accurate estimate of token probabilities, so the result is not as impacted by error.

In the large training set experiment, capitalization continued to make little difference. As in the smaller set, tokenizing punctuation improved precision a little, and with a little work might do even better.

7 Future Work

My implementation for these experiments is written fairly inefficiently, but with some optimization and parameter tuning it could be acceptably used to make actual filtering decisions (assuming results similar to those presented; i.e. zero one false positive per 800 legitimate messages). Studying alternative learning algorithms therefore may not be fruitful. The approach of improving performance by adding extra features such as punctuation and header/token pairs for the majority of other statistical approaches, which use vectors of both present and missing features. Tuning a punctuation tokenizer seems especially promising.

In the quest to find optimal tokenizers, the PhD work of de Marcken[3] may be helpful. He studied statistical methods for learning both vocabularies and grammars from unsegmented data streams (both text and continuous speech). Ideally, spam filters could evolve tokenizers at the same rate that spammers devise ways to spoof tokenizers.

8 Conclusion

In this paper I used the Naïve Bayes algorithm to classify over 2000 pieces of legitimate and unwanted "spam" e-mail I received. I examined the effect of several parameters in this process including training set size and recency, algorithm implementation details, feature extraction method, and message portions examined. Most significantly, I learned that

- case sensitivity doesn't matter much
- examining punctuation can improve precision a little; the Graham-based algorithm (considering only the most informative tokens) outperforms standard algorithm description, especially with small corpora. This echoes the results obtained by McCallum and Nigam in [16]
- precision is dramatically affected by size or recency
- character frequency can perform well given sufficient training data.

My best classifier variant classifies messages with 100% legitimate recall and 92.9% legitimate precision with room for improvement — comparable to or better than most published results.

References

[1] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras, and Constantine D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. In *Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML)*, pages 9–17. European Conference on Machine Learning, May 2000.

[2] Xavier Carreras and Luís Màrquez. Boosting trees for anti-spam email filtering. In Ruslan Mitkov, Galia Angelova, Kalina Bontcheva, Nicolas Nicolov, and Nikolai Nikolov, editors, *Proceedings of RANLP-01, 4th International Conference on Recent Advances in Natural Language Processing*, pages 58–64, Tzigov Chark, BG, 2001. Tzigov Chark, Bulgaria.

[3] Carl de Marcken. *Unsupervised Language Acquisition.* PhD thesis, MIT, 1996.

[4] Olivier de Vel. Mining E-mail authorship. In *Workshop on Text Mining, ACM International Conference on Knowledge Discovery and Data Mining*, 2000.

[5] SpamAssassin development team. SpamAssassin version 2.55, 2003. www.spamassassin.org.

[6] Yanlei Diao, Hongjun Lu, and Dekai Wu. A comparative study of classification-based personal e-mail filtering. In Takao Terano, Huan Liu, and Arbee L. P. Chen, editors, *Proceedings of PAKDD-00, 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 408–419, Kyoto, JP, 2000.

[7] H. Drucker, Donghui Wu, and V.N. Vapnik. Support vector machine for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, September 1999.

[8] Kevin R. Gee. Using latent semantic indexing to filter spam, 2003. To appear in ACM Symposium on Applied Computing, Data Mining Track, 2003.

[9] Paul Graham. A plan for spam, 2002. www.paulgraham.com/spam.html.

[10] Paul Graham. Better bayesian filtering. In *Proceedings of the First Annual Spam Conference*. MIT, 2003. www.paulgraham.com/better.html.

[11] Robert Hall. A countermeasure to duplicate-detecting anti-spam techniques. Technical Report 99.9.1, AT&T Research Labs, 1999.

[12] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*, chapter 17, pages 658–659. Prentice Hall, 2000.

[13] Hooman Katirai. Filtering junk E-mail: A performance comparison between genetic programming and naive bayes. Unpublished manuscript: citeseer.nj.nec.com/katirai99filtering.html, 10 1999.

[14] David D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, 1998.

[15] Giuseppe Manco, Elio Masciari, Massimo Ruffolo, and Andrea Tagarelli. Towards an adaptive mail classifier, 2002.

[16] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification, 1998.

[17] Tom M. Mitchell. *Machine Learning*, chapter 6, pages 177–184. McGraw-Hill, 1997.

[18] Cormac O'Brien and Carl Vogel. Spam filters: Bayes vs. chi-squared; letters vs. words. Technical Report TCD-CS-2003-13, Trinity College Dublin, April 2003.

[19] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[20] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C.D. Spyropoulos, and P. Stamatopoulos. A memory-based approach to anti-spam filtering for mailing lists. *Information Retrieval*, 6(1):49–73, 2003.

[21] Bilge Say and Varol Akman. Information-based aspects of punctuation. In *Proceedings of the Association for Computational Linguistics Workshop on Punctuation*, pages 49–56, Santa Cruz, California, 1996.

[22] Bilge Say and Varol Akman. Current approaches to punctuation in computational linguistics. *Computers and the Humanities*, 30(6):457–469, 1997.