Bayesian Filtering: Beyond Binary Classification

Ben Kamens
bjk5@fogcreek.com
Fog Creek Software, Inc.

Abstract: Bayesian Filtering is used in FogBugz 4.0 to remove spam from incoming email. But it's also used very successfully to classify email into various categories, for example, to separate technical questions from sales-related questions. This paper describes a tournament algorithm used to classifying incoming email into multiple bins.

Bayesian spam filters are rapidly approaching a level of speed and accuracy that must have even Alan Ralsky[1] shaking in his boots. It is only a matter of time until these tools are widely adopted and their ability to remove over 99% of incoming spam puts a chokehold on the kings of junk mail. The technique is so promising when sorting e-mail into spam and not-spam categories that it would be a waste to ignore the possibilities of expansion.

This paper summarizes an attempt at using Bayesian filtering to automatically sort documents into a variable number of categories. This sorting capacity was specifically developed as a feature for Fog Creek's FogBugz software. Much of the sorter's success is attributed to Paul Graham's seminal work[2] on Bayesian filtering, Gary Robinson's subsequent variation[3], and projects such as SpamBayes[4] that have developed effective spam filters on the shoulders of both Graham and Robinson. By extending the use of Robinson's (or more accurately, Fischer's[5]) chi-squared combining rule beyond binary classification, the project led to the development of a 'tournament' style text classification algorithm which will be described below.

FogBugz is a tracking tool used by companies and development teams to keep track of project bugs and customer inquiries. Importantly, it has the ability to receive incoming e-mail from an ordinary POP3 mailbox and automatically create new bugs, or cases, to be tracked. The purpose of the Bayesian filter is to enable automatic categorization of these incoming cases into categories, or areas. The categories are defined by the user. For example, a company might use FogBugz to automatically classify their incoming mail into Spam, Technical Questions, and Customer Service Questions. By default, the separation of spam e-mail into a distinct area is included in this organization process. Anybody who sifts through 300 messages about male enhancement, mortgage payment, and Nigerian treasure every morning to find 4 e-mails from family or coworkers is well aware of the need for this filtering. At the time of writing, Bayesian spam filters are fairly common, and if you're not using one then I highly suggest you join the club. However, I believe that there are only a few other pieces of software which use Bayesian filtering to extend beyond binary spam/ham classification; POPFile[6] is probably the most widely available. Key differences: POPFile uses the Naïve Bayes technique of combining probabilities

[1] "Sick of spam? Then blame Alan Ralsky. He emails a billion of them a day." http://observer.guardian.co.uk/uk_news/story/0,6903,968089,00.html
[2] Paul Graham. http://www.paulgraham.com/spam.html
[3] Gary Robinson. http://www.linuxjournal.com/article.php?sid=6467
[4] The SpamBayes Team. http://spambayes.sourceforge.net/
[5] Gary Robinson. http://garyrob.blogs.com/whychi90.pdf
[6] The POPFile Team. http://popfile.sourceforge.net/

instead of Fischer's chi-squared rule, and the FogBugz sorter's Tournament algorithm strays from the POPFile approach.

Significantly, one installation of FogBugz has the ability to sort multiple mailboxes into multiple sets of areas *with entirely independent sorting procedures.* Let's say you work for Pfizer Inc. and you have two mailboxes. Your personal mailbox gets slammed with more spam than you can shake a stick at. Your inbox at work gets a few spams per day intermixed with at least 10 legit messages, most of which contain the words 'Viagra' and 'prescription'. For you and your Pfizer coworkers, 'Viagra' has a very different meaning depending on the originating mailbox. The FogBugz sorter can treat these two cases separately, and your personal mailbox will filter out the overload of undesired 'Viagra' messages while your inbox at work will understand that an email containing 'Viagra' is probably an important message.

It should be noted that as a byproduct of this project, the FogBugz sorter is not limited to e-mail classification; for instance, it currently serves as an automatic discussion board moderator for Fog Creek Software.

Essential Background

In order to understand the development of the classification process, some background information in Bayesian filtering is necessary. Reading the entirety of Graham and Robinson's articles will leave you feeling like an expert, but if you choose to press on (which you probably will, despite my advice), the following should be sufficient preparation for the rest of the paper.

In "A Statistical Approach to the Spam Problem"[3], Robinson explains:

> For each word that appears in the corpus, we calculate:
>
> - $b(w)$ = (the number of spam e-mails containing the word w) / (the total number of spam e-mails).
> - $g(w)$ = (the number of ham e-mails containing the word w) / (the total number of ham e-mails).
> - $p(w) = b(w) / (b(w) + g(w))$
>
> $p(w)$ can be roughly interpreted as the probability that a randomly chosen e-mail containing word w will be a spam. Spam-filtering programs can compute $p(w)$ for every word in an e-mail and use that information as the basis for further calculations to determine whether the e-mail is ham or spam.

There are many techniques available for using the collection of $p(w)$ values to obtain a prediction regarding the likelihood that a document is associated with a specific category. Of these, Robinson's technique, which is borrowed from R.A. Fischer's combination of probabilities into a chi-squared distribution, has been extensively tested and is used by the most successful filters, including SpamBayes. Robinson provides ample theoretical justification for this improvement in practical accuracy over the original filters conceived by Graham [1]. The resultant value from Robinson's technique, which he labels '*I*', is used to create a prediction:

> *I* is an indicator that is near 1 when the preponderance of the evidence is in favor of the conclusion that the e-mail is a spam and near 0 when the evidence points to the conclusion that it's ham…it is a useful characteristic of *I* that it is near .5 in such cases [where there is strong evidence in favor of both possible conclusions], just as it is near .5 when there is no particular evidence in one direction or the other. When

> there is significant evidence in favor of both conclusions, *I* takes the cautious approach….[t]he SpamBayes project…takes advantages of this by marking e-mails with *I* near .5 as uncertain.

When attempting to categorize into an arbitrary number of categories, the logic surrounding '*I*' and uncertain predictions are among the problems encountered. However, SpamBayes is only one of many successful projects that have reported excellent statistics from Robinson's indicator. It has proven to be an extremely promising route. I first tried to use the simplest and purest technique of expanding this method beyond binary classification.

The Rank Algorithm

The Rank algorithm slightly modified Robinson's probability calculations to support multiple categories:

For each category 'X',

- b(w) = (the number of 'X' e-mails containing the word w) / (the total number of 'X' e-mails).
- g(w) = (the number of 'Not X' e-mails containing the word w) / (the total number of 'Not X' e-mails).

Thus, if we are examining a category list of 'E-mail from my Girlfriends', 'Family', and 'Other E-mail', each token will have four associated *p(w)* values: *p(w in 'Girlfriend e-mail')*, *p(w in 'Family')*, *p(w in 'Other E-mail')*, and *p(w in 'Spam')*. As suggested by Robinson and SpamBayes, Fischer's chi-squared rule is then used to merge all of the *p(w in X)* values into one probability score which is indicative of the association that a single document has with category *X*. The Rank algorithm then selects the highest indicative score as the most likely category and predicts accordingly. This explanation is slightly simplified; special cases for Spam predictions were considered in order to avoid false positives, and rules were designed to allow for uncertain predictions. These special cases and rules are discussed later as they end up providing justification for the Tournament algorithm.

First Problem with Rank Categorization: Imbalanced Training

When training a Bayesian sorter, the SpamBayes team suggests that, "imbalance between the number of ham and spam trained can cause problems, and worse problems the greater the imbalance; the system works best when trained on an approximately equal number of ham and spam." [2]

In the Rank algorithm, our concepts of 'ham' and 'spam', as used by SpamBayes, change as each category takes a turn at being 'X' in the above equations. Consider the following situation, which appears to have a perfectly balanced set of training categories.

Category	Spam	Work	Girlfriends	Family	Other	Ex-girlfriends
Doc Count	20	20	20	20	20	20

With each calculation of a category's indicative score, we are examining probabilities, from *b(w)* and *g(w)*, of 'X' versus 'Not X'. This means that even with this perfectly balanced example, each calculation tests a training set of 20 documents associated with 'X' against 100 documents associated with 'Not X'. Clearly this problem becomes more pronounced once a user has trained hundreds or thousands of documents, and the Rank algorithm suffers from the accuracy pitfalls mentioned by the SpamBayes team. This imbalance caused by multiple categories introduces a bias that must be reduced in order to achieve high accuracy.

The imbalance problems became even more pronounced when I tried to create special cases to help eliminate false spam positives and allow for 'Undecided' predictions. Since each category 'X' suffers from the imbalance, it is possible that each indicative probability score is extremely low; the probabilities may even continue to drop as more documents are trained into many categories and more categories are created. Whereas SpamBayes, due to its binary classification, may deal with scores as high as .9 to .99 for Spam or Not Spam indicators, this is much less likely with an arbitrary number of categories. Consider this feasible result of the Rank algorithm:

Category	Spam	Work	Girlfriends	Family	Other	Ex-girlfriends
Indicative Score	.35	.09	.2	.0005	.001	.003

Note: All indicative scores due not have to add up to 1 because they are not true probabilities, as explained by Robinson in "Why Chi?"[5]*. This is why I call them "indicative scores." (I'll leave the statistical jargon and explanation up to Robinson and other experts)*

With the above results, can any category be confidently predicted? Even if Spam is the correct category, how do we set a Spam cutoff value with such low (and decreasing) probabilities (SpamBayes usually uses .8 or .9)? It is also important that 'Girlfriends' is apparently more similar to Spam than any of the other categories. At what point should the Rank algorithm consider the 'Girlfriends' runner-up value to be significant enough to necessitate an 'Undecided' prediction in order to avoid a false positive? Nobody wants to lose a message from a significant other. These troubles only thicken when categories increase and probabilities drop even further. Finding a constant that can be used to determine whether or not a runner-up value is significant becomes nearly impossible.

I know, you're thinking, "Stupid example! It should be trivial to filter in this case." You can imagine some sort of whitelist / blacklist setup that automatically moves e-mails into the right category based on the e-mail addresses of my girlfriends and family. If you are one such naysayer, then you are grossly underestimating the quantity of addresses we're talking about. I just don't have the sort of time required to type all those in. The former list alone could take weeks. You're also forgetting the malicious nature of Ex's – I wouldn't put it past them to create a fake Hotmail account in order to sneak some hate-mail through my blacklist. My ideal filter needs to learn to pick out many other characteristics besides the e-mail's From address.

Second Problem with Rank Categorization: Category Similarity

The $p(w)$ calculation above results in a probability associated with every token from an e-mail, but both Robinson and Graham suggest only including the most interesting tokens in the probability combining process. An interesting probability is defined as "furthest from .5," and Graham uses only the 15 most interesting tokens while SpamBayes uses 150. Either way, the number of tokens used in calculation is a constant that is usually smaller than the number of tokens produced by a single e-mail.

Why does this technique create a potential problem for a multiple category sorter? The trouble arises when two categories are similar to each other and both are dissimilar from a third category. This is a common occurrence; consider sorting between Spam, Girlfriends, and Family categories. Spam tokens are going to look drastically different from the tokens that are indicative of either Girlfriends or Family. However, tokens such as ‘love’, ‘Ben’, ‘dinner’ and so on might be equally associated with both Girlfriends and Family areas. These tokens have the property of being extremely high indicators of an e-mail that does *not* belong in the Spam category. For example, the calculations for a ‘love’ token in the Family category may be:

- b(w) = (number of ‘Family’ e-mails containing ‘love’) / (total number of ‘Family’ e-mails).
  = 10 / 12
- g(w) = (number of Not ‘Family’ e-mails containing ‘love’) / (total number of Not ‘Family’ e-mails).
  = 1 / 50

These numbers are likely to be very similar for the Girlfriends category calculation (well, maybe…). The ‘love’ token is going to be one of the most interesting tokens for the Girlfriends, Family, and Spam calculations. For all three category calculations, ‘love’ strongly suggests that the sorter should keep this e-mail away from the Spam category.

This result doesn’t seem like much of a problem until two categories are so similar that they share many interesting tokens. Such a result is common. I know my previous example is contrived, so consider examining categories ‘Customer Support’ versus ‘Technical Issues’ versus Spam. The most interesting tokens will be selected (justifiably) due to their ‘Not Spam’ indication. As a byproduct, many of the tokens that distinguish between ‘Customer Support’ and ‘Technical Issues’ are ignored from calculation; they were interesting, but not as interesting as the tokens that screamed “*this is not a spam e-mail!*”

It’s easy to see the problem now: for this example the Spam indicative score will be very low while the Customer Support and Technical scores will be high but indistinguishable. In this case the sorter is smart enough to avoid a false spam positive, but it will often fumble when trying to complete the prediction due to a lack of Customer Support versus Technical evidence. The Rank algorithm accuracy decreases quickly when multiple categories are similar and these are drastically dissimilar from another category (usually Spam).

It seems to me that there are two ways to solve this problem of evidence. The first, and most obvious, is to consider more tokens during the combining calculation. However, this is not advised by Graham, Robinson, or the SpamBayes team, and testing has shown that including all tokens hurts overall accuracy. I believe the loss of predicting capability can be explained by the inclusion of too many *p(w)*’s that lack interest (are near .5). These tokens push the indicative score toward .5 and away from the decisive endpoints. Graham specifically explains that including all tokens results in a filter that “tend[s] to miss longer spams, the type where someone tells you where they got rich from some multilevel marketing scheme.” [3] Since considering a larger number of tokens is ill-advised, the remaining solution lies with the Tournament algorithm.

Unlike Rank, the Tournament algorithm pits categories directly against one another, instead of ‘X’ versus ‘Not X’. Accordingly, the initial *p(w)* calculation is modified:

For each tournament round of category ‘X’ versus category ‘Y’,

- b(w) = (the number of ‘X’ e-mails containing the word w) / (the total number of ‘X’ e-mails).
- g(w) = (the number of ‘Y’ e-mails containing the word w) / (the total number of ‘Y’ e-mails).

What does this calculation mean? *b(w)* and *g(w)* are being calculated based on a seemingly ridiculous assumption: for each tournament round between ‘X’ and ‘Y’, we are calculating a probability score that indicates how closely an e-mail correlates with ‘X’ *in a world where only ‘X’ and ‘Y’ e-mails exist.* Thus, for each tournament round, we use this temporary assumption to create the binary world that SpamBayes benefits from. The algorithm follows a sort of “playoff” style by pitting categories against one another and dropping the loser out of consideration. Again, there are special cases for the Spam and Undecided predictions. It turns out that the Tournament algorithm solves both problems associated with Rank.

First Tournament Solution: Fewer Imbalances

Consider the training set example that causes problems for Rank:

Category	Spam	Work	Girlfriends	Family	Other	Ex-girlfriends
Doc Count	20	20	20	20	20	20

The Tournament algorithm avoids the ‘X’ versus ‘Not X’ bias by comparing categories two at a time. Thus, in this example, every tournament round is perfectly balanced.

Tournament process: [[[[[Spam vs. Work] vs. Girlfriends] vs. Family] vs. Other] vs. Ex’s]

[[‘X’ vs. ‘Y’] vs. ‘Z’] indicates that ‘X’ competes against ‘Y’, and the winner competes against ‘Z’

At each step of the process, a category with 20 trained documents competes against a category with 20 trained documents. Thus, the process abides by SpamBayes’ suggestion of reducing training imbalances. As a result, each round results in a telling indicative score that is much more useful than the low probabilities associated with Rank. The reason that these scores are so informative is because the correct pieces of evidence are chosen during each round, as explained below.

Second Tournament Solution: Distinguishing between Similar Categories

The Tournament algorithm also has an improved ability to distinguish between similar categories. For instance, if we return to the situation of sorting e-mail into Girlfriends, Family, or Spam, we will find that Tournament is not troubled by the inclusion of a constant number of interesting tokens.

Suppose an e-mail is being sorted and the first round features Girlfriends versus Spam. As before, a token such as ‘love’ or ‘Ben’ will be extremely indicative of ‘Not Spam’, and this token will be used to create a high indicative score for Girlfriends. Spam will lose and be eliminated from the Tournament. Therefore, in the following round of Girlfriends versus Family, tokens that scream “*this is not a spam e-mail*” no longer need to be considered. ‘Love’ may not seem interesting anymore because it often appears in both the remaining categories. However, tokens such as ‘nerd’, ‘geek’, and ‘dumped’ will alert the sorter that this message looks a lot more like Girlfriends than Family. In fact, in the *p(w)* calculations for Girlfriends versus Family, an entirely new set of tokens will be defined as ‘interesting’, and these will be the tokens that help differentiate specifically between the remaining categories. Now the sorter becomes aware of the right tokens at the right time in order to shed light on the e-mail’s nature.

As a result, the Tournament algorithm does not struggle to differentiate between two (or more) similar categories that both differ drastically from a third category. It is smart enough to temporarily isolate two categories and look at the most relevant tokens, thereby benefiting from the advantages of binary classification.

Obviously, the Tournament algorithm must take special consideration of rounds featuring Spam. In testing, Tournament has produced significantly less false positives than Rank by implementing an extra calculation. Spam is considered to be a normal contender in the tournament, and therefore must compete like any other category. However, the category must also win a general Spam versus ‘Not Spam’ competition. In practice, this prejudice has helped eliminate false positives, drastically decrease the amount of spam placed in Undecided, and maintain a low number of false negatives.

To Be Continued

Testing has revealed that the Tournament algorithm significantly increases sorting accuracy when compared to Rank, but this research is far from complete. I readily concede that I am not a statistician nor am I a math expert. However, I do know that this technique works, and it works well. I’m sure there are plenty of improvements to be made to the algorithm, and you’ve probably thought of a few. If you choose to implement a form of this Bayesian filter and discover something new, please e-mail me. I would be very interested in your experience.

Notes:

[1] “It may be worthwhile to mention here a key difference between this approach and many other combining approaches that assume the probabilities are independent. This difference applies, for example, to such approaches as the ‘Bayesian chain rule’ and ‘Naive Bayesian classification’. Those approaches have been tested in head-to-head combat against the approach described here using large numbers of human-classified e-mails and didn't fare as well (i.e., didn't agree as consistently with human judgment).

The calculations for those approaches are technically invalid due to requiring an independence of the data points that is not actually present. That problem does not occur when using the Fisher technique because the validity of the calculations doesn't depend on the data being independent. The Fisher calculation is structured such that we are rejecting a null hypothesis that includes independence in favor of one of the two alternative hypotheses that are of interest to us. These alternative hypotheses each involve non-independence, such as the correlation between ‘sex’ and ‘porn', as part of the phenomenon that creates the extreme combined probabilities.”
Gary Robinson. http://www.linuxjournal.com/article.php?sid=6467.

[2] “A variety of approaches to training have been tried - things that we found:
* the system generates remarkably good results on very few trained messages, although obviously, the more the better.
* training on an extremely large number of messages may start to produce slightly poorer results (once you're training on thousands and thousands of messages).
* imbalance between the number of ham and spam trained can cause problems, and worse problems the greater the imbalance; the system works best when trained on an approximately equal number of ham and spam.
* It's really important that you're careful about selecting your training sets to be from the same sources, where-ever possible. Several people got caught out by training on current spam, and old ham. The system very quickly picked up that anything with a 'Date:' field with a year of 2001 was ham. If you can't get ham and spam from the same source, make sure the tokenizer isn't going to pick up on the non-clues.
* "mistake-based training" - where you only train on messages that the system got wrong - results in the system taking a lot longer to get really good at what it does. You're better off feeding it the things it gets right, as well as the things it gets wrong.”
The SpamBayes Team. http://spambayes.sourceforge.net/background.html

[3] “[T]hey calculated probabilities differently. They used all the tokens, whereas I only use the 15 most significant. If you use all the tokens you'll tend to miss longer spams, the type where someone tells you their life story up to the point where they got rich from some multilevel marketing scheme. And such an algorithm would be easy for spammers to spoof: just add a big chunk of random text to counterbalance the spam terms.”
Paul Graham. http://www.paulgraham.com/better.html