Using AdaBoost and Decision Stumps to Identify Spam E-mail

Tyrone Nicholas

June 4, 2003

Abstract

An existing spam e-mail filter using the Naive Bayes decision engine was retrofitted with one based on the AdaBoost algorithm, using confidence-based weak learners. A comparison of results between the two is presented, with respect to both speed and accuracy.

1 Background

1.1 Introduction

Unsolicited commercial e-mail, or "spam", has in recent months escalated from a minor (if irritating) nuisance to a serious drain on human and computer resources. By some estimates, over 40 percent of all email sent in the United States is spam, compared to 7 percent in 2001. Estimates of lost productivity are as high as $10 billion year.

Software to combat spam e-mail, until recently, was based on simple keyword filters; if a given term was present in a message's headers and/or body, the mail was labelled as spam. This rapidly became unscalable as each potential spam term had to be manually added to the system, resulting in little time saved for users. Artificial intelligence techniques are an obvious alternative.

1.2 Academic Approaches

Zhou et al (2003) proposed using interfaces used in peer-to-peer networks to handle spam. They reported 97 percent accuracy using this technique. This was deemed beyond the scope of this course and will not be investigated here.

From a machine learning perspective, spam filtering is just text categorization, with categories of spam and nonspam (also known as "ham"). Androutsopoulos (2000) reported that Naive Bayes obtained accuracy

an order of magnitude higher than Microsoft Outlook's keyword patterns. Drucker et al (1999) compared the performance of Ripper, Rocchio, support vector machines, and boosting. They concluded that either boosting or SVMs were the methods of choice; boosting offered a somewhat higher accuracy but had a much longer training time.

There are some avenues in support vector machines that have not yet been examined for spam, notably transductive SVMs (Joachims, 1999) or "logical" SVMs (Vapnik and Wu, 1999). Nonetheless, because of this greater accuracy, it was decided to pursue a boosting-based algorithm. One that has received much attention recently is AdaBoost (Schapire, 2001). Schapire and Singer (2000) showed that in text categorization, AdaBoost can give high levels of stability and accuracy if properly parameterized.

Boosting algorithms work over another base algorithm, called the weak learner. The slow performance of Drucker et al (1999) is at least partly due to the choice of C4.5 as the weak learner; though it may have improved accuracy, this is not necessary for a weak learner, not at the cost of performance. Carreras and Marquez (2000) used boosting with simple decision stumps - decision trees with only a single node, with the presence or absence of a single term as a predicate - and were able to obtain with that significantly better accuracy than with Naive Bayes. Counterintuitively, using decision trees up to five levels deep obtained only marginally higher accuracy than stumps, with a performance penalty.

It was therefore hoped that AdaBoost, with decision stumps as its base learner, could be used as the basis for a spam filter.

1.3 Industry / Community Approaches

Graham (2002) introduced the open-source community to the basic concept of Naive Bayes filtering. Since then, several open-source systems have become available, and their authors boast of success rates of 99 percent or more, well beyond anything achieved in the academic literature. These systems have not undergone academic peer review, and their results have been received with caution by academe.

However, one thing that has been far better realized outside the academy than inside it is the fluid nature of spam emails themselves. A simple ASCII interpretation of text, as is standard in the published research, is woefully inadequate in the "real" world. What the human eye sees is often very different from text; the use of HTML is a prime example. HTML and JavaScript also make possible other tricks, such as embedding text in hyperlinked images, or even putting boilerplate letter-from-a-friend content hidden in HTML comments, specifically to throw off Bayesian filters. Other tricks include deliberately misspelled words, extra whitespace between letters, and nonstandard characters (Graham-Cummings, 2003).

If the research shows that boosting could outperform Naive Bayes, then one candidate for a project would be to take an actual running Naive Bayes system, complete with the advanced text and HTML processing and replace its decision engine with one based on boosting, and compare the results.

2 System Design

2.1 Popfile

Popfile, written by John Graham-Cummings (http://popfile.sourceforge.net), is a POP3 proxy server; it connects to an ISP's server, downloads the email, and labels it according to its cached probabilities. Initially, the user must manually reclassify every incoming message, but after a remarkably short while it can function independently. The author found it successfully able to label nearly all incoming spam after training it with just 50 examples.

In addition to its Naive Bayes decision engine, Popfile contain a powerful preprocessor that will intelligently many of the tricks describe in Graham-Cumming (2003), including HTML parsing. No other open-source spam product the author reviewed is quite as mature, as of yet.

In this project, the stopword, parsing, and decision engines of popfile were integrated into a set of other programs to carry out the boosting tests. Since Popfile is implemented entirely in Perl, that was the language chosen for this project as well. Given the potentially high performance requirements, this should be considered suboptimal.

2.2 Corpora

Most of the open-source spam filters were tested on the developers' own email, giving rise to the concern that the algorithms were biased towards a particular sample. Unfortunately, the corpora used in most academic systems are not much better. The Ling-Spam corpus introduced in Androutsopoulos (2000) used an linguists' mailing list as a source of ham mail, something not exactly representative of other users. Ling-Spam has less than 3,000 messages, compared to the over 10,000 recommended by Schapire and Singer (2000) for AdaBoost to provide superior performance to Naive Bayes. The benchmark PU1 corpus (Androutsopoulos, 2000) is gleaned from just one individual's emails, and has just over 1,000 messages.

A very large archive of spam can be found at the site `www.spamarchive.org`. This now holds more than 200,000 spam messages. Unfortunately, it does not contain a ham archive, and the author's own collection is not large enough to be suitable for that purpose (not to mention bias). Mitchell (unpublished) has proposed a boosting method that does not require negative data, but it remains to be seen how robust this is to variegated types of data.

In the end, this project makes use of the test corpus made available at `spamassassin.org/publiccorpus`. This archive contains 1,896 spam and 4,149 ham messages, and includes several hundred specifically aimed to "confuse" conventional systems, in both directions. Two subsets were created for faster testing, the "tiny" dataset of 26 ham and 17 spam messages (chosen as all files with a name starting with "0000") and the "small" set of 296 ham and 197 spam (files starting with "000").

2.3 Algorithm

2.3.1 Boosting

The boosting algorithm consists of the following steps. We have a set of m emails, each consiting of a feature vector x_i. For each i there is a $y_i \in +1, -1$, where $+1$ indicates a spam and -1 a ham. (Please, no jokes.)

We initialize a distribution $D_1 : \mathbb{R} \rightarrow \mathbb{R}$ as the uniform distribution ($D_i = 1/m$ for all i). This distribution represents the probability that the training sample i is a reliable predictor of the outcome y. This distribution is continually updated using each hypothesis, then used againt train the next one.

In pseudocode, the execution is as follows (Schapire and Singer, 2000):

```
for (t=1; t<=T; t++)
        h_t = WeakLearner(X, D_t)
        foreach i
```

$$D_{t+1}(i) = \frac{D_t(i) exp(-y_i h_t(x_i))}{Z_t}$$

```
        next
next
```

where Z_t is a normalization factor chosen so that $\sum_{i=1}^{m} D_{t+1}(i) = 1$ as befits a distribution.

Carreras and Marquez (2000) found that T had to be in the order of 500 in order to achieve good results.

2.3.2 Weak Learner

Drucker et al (1999) had used C4.5 decision trees as weak learners; they reported high accuracy but extremely slow execution time (and for this reason recommended using SVMs instead). Carreras and Marquez (2001) used decision stumps instead; with a single node, execution should be faster. Candidate stumps are simply the presence or absence of a word in an email.

Schapire and Singer (1999) present the following criterion for choosing the term to use as a stump. Choose the word j such that the term:

$$Z_t = 2 \sum_{j \in \{0,1\}} \sqrt{W_{+1}^j W_{-1}^j}$$

is minimized, where

$$W_b^j = \sum_{i=1}^{m} D_t(i) * [\![x_i \in X_j \wedge y_i = b]\!]$$

where for any predicate π, $[\![\pi]\!]$ is the indicator function for π. X_j represented the set of samples for which the word j is present.

In practice, it was found that iterating over the complete word set was computationally prohibitive. Even the "tiny" set of 43 emails still had over 4,000 distinct words. The full data set contained over 78,000 words, slowing the pace of testing to a crawl. Iterating through the entire set seemed pointless when only one would be picked. A heuristic was added to screen out words; only those words with a relative frequency ratio of greater than 10 were used. That is, a word was only a candidate for a weak learner if it occurred at least 10 times as often in one set as in the other. This eliminated 95 percent of the words, but on the full data set still left several hundred word candidates. More sophisticated algorithms to choose weak learners are an obvious target for future work.

2.4 Architecture

Although Popfile works as a proxy server, this is obviously unsuitable for batch testing. Those modules needed specifically for parsing and word mangling `MailParse.pm` and `WordMangle.pm`, as well as the Naive Bayes engine (`Bayes.pm`) were invoked directly from scripts implemented by the author. The `MailParse` module was extended to allow for persistence, since loading in and parsing all 6,000 messages took over 20 minutes, even on a Sun Blade workstation.

The following modules were built:

- `Corpus.pm` - ADT to hold actual word frequencies, persistable.
- `WeakLearner.pm` - Encapsulated the weak learner selection algorithm.
- `AdaBoost.pm` - Actual iteration of the algorithm
- `RunBoost.pm` - Wrapper for AdaBoost; handled n-fold cross-validation.
- `RunBayes.pm` - Wrapper for Naive Bayes.
- `ParsedMail.pm` - ADT for parsed mail messages, supporting persistence.
- `mkcorpus` - Standalone script to generate corpus file.
- `run` - Runner script.

The program is invoked by the command `./run [rounds] [T] [size]`, where `rounds` is the number of folds in n-fold cross-validation, `T` is the number of iterations to run AdaBoost, and `size` selects the data set to use (must be one of "tiny", "small", or "full". The program will read in all files from two subdirectories called `ham` and `spam`, and verify that they are saved parsed-mail files made earlier with the `ParsedMail` script.

3 Results

3.1 Evaulation Criteria

Most of the literature considers false spam positives to be objectively worse than uncaught spams, and weights the precision accordingly. This is not done here because boosting is a "bucket-blind" algorithm, in that the semantic meaning of the two classifications is irrelevant. This "blindness" is one reason boosting has been used more successfully in multi-class problems (Schapire and Singer, 2000).

Precision was applied to both buckets equally; the script will simply report the percentage of items in the test set that were classified correctly, whether ham or spam. Recall and F_1 are redundant for this purpose and have been omitted.

Since Popfile's original Naive Bayes engine remains in the code, a comparison between the two is straightforward and is incorporated into the run script.

3.2 Speed

The AdaBoost algorithm is of $O(Tmwf)$, where T is the number of boosting iterations, m the number of training samples, w the number of unique words used, and f the number of folds in cross-validation. Not surprisingly, this does not even come close to being scalable; a full pass through the data set takes nearly 60 minutes on a Sun Blade workstation, even with a relatively low value of T. For that reason, relatively few tests were carried out on the full data set.

Even on the "tiny" data set, and with values considered inadvisable in the literature, such as $T = 3$ and $f = 3$, the script took 41 s (16 s for NB and 19 s for AdaBoost). Increasing f to 10 raised execution time to 35 s for Naive Bayes, but 79 s for AdaBoost. Using the "small" instead of "tiny" corpus resulted in a similarly larger increase in time for AdaBoost (414 s) than Naive Bayes (142 s).

Use of the "full" corpus was beyond the capabilities of the hardware and time available. With an earlier iteration of the code, the "full" corpus took 60 minutes to execute and yielded only 72 percent accuracy for AdaBoost. In the interest of time, no further tests were carried out on this data set.

3.3 Accuracy

With $f = 3$ and $T = 3$ on the "tiny" data set, AdaBoost obtained only 86.7 percent accuracy, compared with 95.3 percent for Naive Bayes. With $f = 10$, *ceteris paribus*, Naive Bayes boasted 100 percent accuracy; AdaBoost showed no improvement, still reporting 86.7 percent. On the "small" corpus, Naive Bayes' accuracy rose to 97.4 percent, even with $f = 3$; AdaBoost fell to 82.6 percent.

Carreras and Marquez (2001) used values of T greater than 500. Using $T = 100$ here resulted in only marginally better accuracy (88.9 percent), but execution time grew exponentially, taking nearly 30 minutes even on the tiny corpus.

4 Discussion

The evidence found here suggests that the AdaBoost algorithm, using decision stumps, is inferior in terms of both accuracy and speed to the Naive Bayes algorithm for this problem.

It is possible that speed and performance improvements can be obtained with more development effort. Better heuristics for choosing the weak learner are one place to start; possible schemes include POS tagging the words and selecting more semantically significant ones. n-fold validation could be dropped in favor of disjoint training and test sets.

However, there there is little incentive to do so without clear signals that markedly superior results over existing Naive Bayes systems is possible. This research does not point to any.

5 References

1. Androutsopoulos, I., et al. (2000) Learning to filter spam e-mail: a comparison of a Naive Bayesian and a memory-based approach. In *4th PKDD's Workshop on Machine Learning and Textual Information Access.*
2. Carreras, X., and Marquez, L. (2001) Boosting trees for anti-spam email filtering. In *Proceedings of RANLP-01, 4th International Conference on Recent Advances in Natural Language Processing.*
3. Drucker, H., Wu, D., and Vapnik, V. (1999) Support vector machines for spam categorization. *IEEE Transactions on Neural Networks.* 10(5):1048-1054.
4. Graham-Cumming, J. (2003). The Spammer's Compendium. In *Proceedings of the 2003 Spam Conference.* http://www.spamconference.org/proceddings2003.html.
5. Graham, P. (2002) A Plan for Spam. http://paulgraham.com/spam.html.
6. Joachims, T. (1999) Transductive inference for Text Classification using Support Vector Machines. In *Proceedings of ICML-99, 16th International Conference on Machine Learning*, p. 200-209.
7. Mitchell, A. "Boosting" Stumps from Positive Only Data. Technical Report, University of New South Wales.
8. Schapire, RE and Singer, Y. (1999) Improved boosting algorithms using confidence-based predictions. *Machine Learning*, 37(3):297-335.
9. Schapire, RE and Singer, Y. (2000) BoosTexter: A boosting-based approach to text categorization. *Machine Learning*, 39(2/3).
10. Schapire, R. (2001) The boosting approach to machine learning: an overview. In *MSRI Workshop on Nonlinear Estimation and Classification.*
11. Wu, D. and Vapnik, V. (1998) Support Vector Machine for Text Categorization.
12. Zhou, F., et al. (2003) Approximate Object Location and Spam Filtering on Peer-to-peer Systems. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference.*