Good Word Attacks on Statistical Spam Filters

Daniel Lowd
Dept. of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
lowd@cs.washington.edu

Christopher Meek
Microsoft Research
Redmond, WA 98052
meek@microsoft.com

Abstract

Unsolicited commercial email is a significant problem for users and providers of email services. While statistical spam filters have proven useful, senders of spam are learning to bypass these filters by systematically modifying their email messages. In a *good word attack*, one of the most common techniques, a spammer modifies a spam message by inserting or appending words indicative of legitimate email. In this paper, we describe and evaluate the effectiveness of active and passive good word attacks against two types of statistical spam filters: naive Bayes and maximum entropy filters. We find that in passive attacks without any filter feedback, an attacker can get 50% of currently blocked spam past either filter by adding 150 words or fewer. In active attacks allowing test queries to the target filter, 30 words will get half of blocked spam past either filter.

1 Introduction

As spam filters evolve to better classify spam, spammers adapt their messages to avoid detection. For example, as statistical spam filters began to learn that words like "Viagra" mostly occurr in spam, spammers began to obfuscate them with spaces and punctuation, such as "vi@gra." Analogously, as spam filters learned which words occur mostly in legitimate email, spammers learned to add those words to their messages. We refer to these techniques as *attacks* and spammers employing these methods as *attackers*, since their goal is to defeat the normal operation of a spam filter.

One would prefer spam filters that are robust to such attacks, filters that work well against present and future spam. Unfortunately, most empirical evaluations of spam filters ignore this adversarial problem. The problem is a hard one: since spammers are unpredictable, the true effectiveness of a filter cannot be known until deployment. Even so, just as experts analyze the security of networks, operating systems, and cryptographic methods, we can analyze the vulnerabilities of specific spam filters by developing and simulating adversarial attacks.

In this paper, we investigate *good word attacks*, in which a spammer adds extra words or phrases to a spam message that are typically associated with legitimate email. Of the many spam filters in existence, we restrict our attention to the naive Bayes filter, the most popular spam filter, and the maximum entropy filter, one of the most popular text-classification filters. For these filters, a spammer simply needs to identify a list of words considered "strongly legitimate" by the filter to mount an effective good word attack.

We develop and test good word attacks for two scenarios. In *passive attacks*, the attacker constructs a word list without any feedback from the spam filter. Attacks of this type amount to educated guesses regarding which words are "good" and which are "bad." In *active attacks*, the attacker is allowed to send test messages to the filter to determine whether or not they are labeled as spam. While active attacks can yield much better word lists, they may not always be possible since they require the attacker to have user-level access to the spam filter.

In our experiments, we find that current spam filters are extremely vulnerable: in the worst case, adding a few dozen words gets 50% of all currently blocked spam past the filter. Fortunately, preliminary results show frequent filter retraining to be effective, especially when the weight of the most recent training examples is increased by duplication. This underscores the importance of retraining often, not only to adapt to the changing characteristics of email content, but also to mitigate the effectiveness of these attacks.

In Section 2, we describe the statistical filters we used, how we trained them, and some of their characteristics. In Sections 3 and 4, we describe passive and active good word attacks and evaluate them using our filters. We present preliminary work on filter responses in Section 5 and conclude in Section 6.

2 Statistical Spam Filters

Naive Bayes was first applied to spam filtering by (Sahami et al., 1998). Since then, it has been used by many spam filters, including Bogofilter, CRM114, POPFile, SpamAssassin, and SpamBayes. Maximum entropy (maxent) models (Berger et al., 1996) are more popular in the text classification community but have also been applied to spam filtering (Zhang & Yao, 2003).

2.1 Background

A naive Bayes filter describes the joint probability over a set of features $\mathbf{X} = X_1, X_2, \ldots, X_n$ and a class C by making the "naive" assumption that each feature is conditionally independent given the class:

$$P(\mathbf{X}, C) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

The class prior, $P(C)$, represents the relative frequency of each class, while the conditional probabilities $P(X_i|C)$ encode the probability of each feature value given the class. In the spam domain, the class prior $P(spam)$ (where $P(legit) = 1 - P(spam)$) represents what fraction of incoming email is spam, and the probabilities $P(word|spam)$ and $P(word|legit)$ give the probability that each type of email contains each word (or other feature). Classification follows from the class posterior odds:

$$\frac{P(spam|\mathbf{X})}{P(legit|\mathbf{X})} = \frac{P(spam) \prod_{i=1}^{n} P(X_i|spam)}{P(legit) \prod_{i=1}^{n} P(X_i|legit)}$$

If the odds exceed some threshold, then the email is classified as spam. By taking the logarithm, log odds can be represented as the following sum:

$$\log \frac{P(spam)}{P(legit)} + \sum_{i=1}^{n} \log \frac{P(X_i|spam)}{P(X_i|legit)}$$

In this way, a naive Bayes binary classifier can be represented as a set of feature weights and a threshold. Features that are more common in legitimate email than spam have negative weights, while those more common in spam have positive weights.

Unlike naive Bayes, which models the joint distribution of all features, $P(\mathbf{X}, C)$, a maxent filter models the conditional probability of the class given the other features, $P(C|\mathbf{X})$. Assuming Boolean features and two classes (spam and legit), the maxent filter is a logistic regression model and the class odds take the following exponential form:

$$\frac{P(spam|\mathbf{X})}{P(legit|\mathbf{X})} = \exp\left(\sum_{i=0}^{n} \lambda_i X_i\right)$$

where X_0 is an artificial feature that has the value one for every message. As with naive Bayes, the classification is determined by comparing this sum with a threshold.

Naive Bayes and maxent share the same representation (weights and a threshold), but are trained to optimize different criteria. The naive Bayes filter is a *generative* model, trained to maximize the likelihood of the training data, while the maxent filter is a *discriminative* model, trained to maximize the likelihood of the class labels conditioned on the other features in the training examples. In this way, maxent filters are similar to support vector machines: both model the class boundary rather than the entire instance space.

2.2 Filter Training

Our experimental data comes from the Hotmail feedback loop, a system in which users of Hotmail volunteer to label some of the messages they receive as spam or legitimate email. Because the number of labeled messages per user is relatively small, we trained global filters using all data rather personalized spam filters for each user. Our training set consists of 500,000 labeled messages from before May 1, 2004. This was split into tuning and validation sets of 490,000 and 10,000 examples, respectively. The test corpus consists of 10,000 messages received between May 1 and 8, 2004. Email in the test set was received by a different set of users than those of the training set in order to ensure generalization across users, not just across time. Our feature set consists of all tokens that appear in the subject or body of the email, as well as many non-textual features based on the message headers. After selecting only those features that occurred at least 10 times in the training data, just under 290,000 features remain.

In a spam filter, the *false positive* rate is the fraction of legitimate email incorrectly classified as spam, and the *false negative* rate is the fraction of spam classified as legitimate. Adjusting the filter threshold allows us to decrease one error rate by increasing the other. For example, using a lower threshold tends to block both more spam and more legitimate email, decreasing the false negative rate but increasing the false positive rate. To simplify the comparison of different filters, we fix the false positive rates at 10% using the validation or test set and compare the false negative rates. A false positive rate of 10% may sound excessive, but some of this is due to labeling error: users may accidentally or maliciously mislabel messages, or simply disagree about what constitutes "spam". (We briefly experimented with a false positive rate of 5% as well, but found the resulting filters more vulnerable to our good word attacks.)

We tuned each algorithm using 490,000 of the training examples, keeping the remaining 10,000 as a validation set. Naive Bayes performs best on the validation set with a Dirichlet prior where $\alpha_i = 0.1$. Maxent performs best with a Gaussian prior for the λ's with zero mean and variance σ^2 set to 0.1. We used these parameters for the remainder of our experiments.

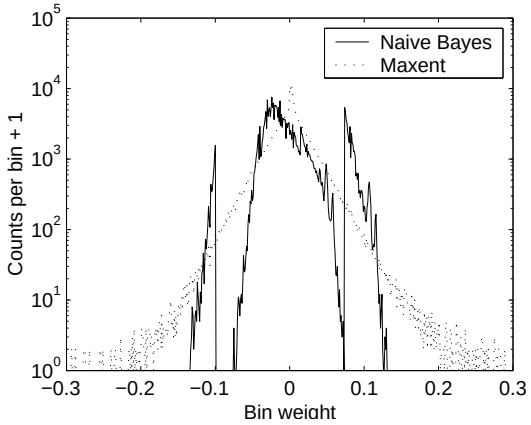

Figure 1: Histograms of normalized feature weights for naive Bayes and maxent filters. Bin width is 0.001. Each bin count was increased by one to permit plotting on a logarithmic scale.

With a fixed false positive rate of 10% on the test set, the maxent filter has 31% fewer false negatives than the naive Bayes filter. (Raw error rates are not meaningful due to labeling error in the test set.)

2.3 Weight Analysis

Overall accuracy is not the only metric of interest: we also wish to determine the susceptibility of each filter to a good word attack. In general, a filter is more susceptible if it has features with very negative weights, since these disguise spam more efficiently. In order to make filter weights comparable, we scaled each filter's weights so that given a false positive rate of 10%, the median score of a true positive is 1 greater than the threshold. This provides a very reasonable interpretation of the weights: a weight of -0.01, for example, provides 1% of the weight necessary to get the median blocked spam past the filter. Put another way, a list of words whose weights sum to -1 is sufficient to get half of the currently blocked spam past the filter.

In spite of its lower error rate, the maxent filter also has large negative weights for many features, suggesting greater susceptibility to good word attacks. Figure 1 shows histograms of naive Bayes and maxent feature weights. The heavier tails of the maxent distribution represent features with relatively large positive or negative weights. The sharp spikes for naive Bayes correspond to features that appear a few times but only in spam or only in legitimate email. The locations of these spikes are determined by the prior weight tuning parameter.

Table 1 summarizes key differences between the two filters: the maxent filter has higher initial accuracy and smaller average weights, but the naive Bayes filter has fewer very positive or very negative weights. The relative importance of these different factors depends on how easy it is for an attacker to find the highest-weight features in each filter, a question we analyze in the next two sections.

Table 1: Filter Weight Summary Statistics

	Naive Bayes	Maxent
Max norm. weight	0.1302	0.5398
Min norm. weight	-0.1329	-0.6346
Avg pos. norm. weight	0.0449	0.0199
Avg neg. norm. weight	-0.0243	-0.0229

3 Passive Attacks

In passive good word attacks, attackers have no direct access to the spam filter. At best, they can make educated guesses as to which words are helpful. In this section, we describe three different passive word selection heuristics and evaluate their effectiveness.

The first and simplest approach is to choose random words from some larger set of words in the hope that such words are more likely to be associated with legitimate email than with spam. The set of words could come from a dictionary or be chosen on the basis of the spammer's domain knowledge. We chose to use dictionary words because these are available electronically. We refer to this type of attack as a *dictionary attack*.

A more sophisticated approach would be to look at what words appear most often in legitimate email and use those. We assume that legitimate email is hard to come by for the average attacker and so use word frequencies from four different English corpora: Reuters news articles, written and spoken English, and USENET messages from 1992. We call this a *frequent word attack*. Wittel and Wu explore a similar attack using a single list of common English words with spammy words removed (2004).

Finally, since spam is easy to come by, we can easily consider word frequencies in spam. Words that appear frequently in legitimate email and never in spam are more likely to be effective than words that are common to both. We refer to this heuristic as a *frequency ratio attack*.

3.1 Public Corpora

Our spam training data consists of all spam on spamarchive.org from the month of April, 2004[1]. Since our filters are trained using data up to May 1, we used spam from the same time period. To convert a set of emails into word frequencies, we selected the message bodies, normalized case, and tokenized them, splitting on non-letters. We

[1] An archive of current spam is available from ftp://spamarchive.org/pub/archives.

further restricted the set of words to lower-case dictionary words.

The Reuters articles are from the Reuters-21578, Distribution 1.0 text categorization test collection, a set of 21,578 documents (but only 19,043 complete articles) from Reuters newswire in 1987[2]. We tokenized all article bodies (splitting on non-letters), normalized case, and generated frequency counts for all dictionary words. Uppercase words, such as proper nouns, are excluded from the list.

The written and spoken English corpora are based on the British National Corpus[3]. We used the word frequencies given but with a certain amount of regrouping. First, we removed number words, since words such as "one" or "five" are more likely to be written as "1" or "5" in an email. We also summed frequencies for words that would be indistinguishable when split on punctuation, (e.g., "can" and "can't"), and removed part-of-speech distinctions that resulted in repeated words (e.g., "no" appeared separately as a determiner and as an interjection).

The word frequencies for the 1,000 most common words on USENET in 1992 are publicly available as well[4]. As with written and spoken English, we split on punctuation and regrouped, ending up with 865 words after the regrouping. Most regroupings were due to distinctions of capitalization – "the" and "The" are treated as two distinct words in this corpus, but we combine their separate frequencies into one.

3.2 Methodology

To determine the effectiveness of a random dictionary word, we averaged the weights of all dictionary words in each filter. Approximately 23,000 of the 290,000 filter features are in our chosen dictionary, the first English ispell dictionary, english.0[5].

For the other passive attacks, we generated explicit word lists and computed the average weight per word according to each filter. Since a set of words whose filter weights sum to -1.0 can disguise the median spam, we use this to compute the expected number of words required to reach this total weight. This number estimates how many words a spammer must add to a typical spam when using a particular attack against a particular type of filter.

[2]The Reuters-21578 text categorization test collection, Distribution 1.0, is freely available for research use at http://www.daviddlewis.com/resources/testcollections/-reuters21578.

[3]English word frequencies were obtained from the companion website for *Word Frequencies in Written and Spoken English: based on the British National Corpus*, http://www.comp.lancs.ac.uk/computing/research/ucrel/bncfreq/.

[4]1992 USENET word frequencies are available from http://www.dcs.shef.ac.uk/research/ilash/Moby/.

[5]English dictionaries are distributed with ispell, available from http://fmg-www.cs.ucla.edu/fmg-members/geoff/ispell.html

Table 2: Passive Attack Results

Experiment	Corpus	NB Words	ME Words
Dictionary	N/A	-132	-555
Freq. Word	Reuters	1,350	1,420
	Written	1,920	2,110
	Spoken	1,470	1,040
	USENET	898	845
Freq. Ratio	Reuters	112	735
	Written	133	305
	Spoken	159	256
	USENET	257	149

For the frequent word attack, we selected the 1,000 most frequent words according to each of the first three English corpora. For 1992 USENET, we used all 865 words.

For the frequency ratio attack, we ranked each word by its frequency in each English corpora relative to its frequency in the spam corpus. Words that never appear in the spam corpus are excluded. (Few words are excluded for this reason, since the spam corpus features over 27,000 distinct dictionary words.) Since the 1992 USENET corpus started out with only 865 words, we selected the top 250 words; for the other corpora, we selected the top 1,000 words.

Note that the frequency ratio method is very similar to how naive Bayes computes weights. In fact, with the exact training data and configuration of the naive Bayes filter, one could discover its exact weights. Lacking those details, an attacker can still approximate the filters using these publicly available corpora.

3.3 Results

Our results are summarized in Table 2. For each experiment and each corpus, we list the expected number of words required to get the median spam past each filter, extrapolated from the median word weight. In some cases, this is more words than in the original word list.

For dictionary attack, the expected number of words is negative, indicating that random dictionary words makes an email look more like spam, not less. As Graham-Cumming (2004) points out, spam is now the majority of email and many spams have random words in them, so a random word is actually more likely to be spammy than good.

Frequent words do much better than random words, but the attack's utility is limited: for most corpora the number of words required exceeds 1,000, a significant increase in message length.

The highest frequency-ratio words do quite well. With the right corpus, fewer than 150 words are needed to get the median spam past either filter. Starting with a less blatant spam, even fewer words may be necessary.

Figure 2: FINDWITNESS(M_{spam}, M_{legit})

$M_{curr} \leftarrow M_{legit}$
repeat
  $M_{prev} \leftarrow M_{curr}$
  if some word w is in M_{curr} but not M_{spam} **then**
    remove w from M_{curr}
  else if some w is in M_{spam} but not M_{curr} **then**
    add w to M_{curr}
  end if
until M_{curr} classified as spam
return (M_{curr}, M_{prev})

Figure 3: FIRSTNWORDS(M_{spam}, M_{legit})

$L \leftarrow \emptyset$
$(M_{\epsilon+}, M_{\epsilon-}) =$ FINDWITNESS(M_{spam}, M_{legit})
for each word $w \in W$ **do**
  if $M_{\epsilon+}$ with w classified as legitimate **then**
    add word w to L
  end if
  if L contains at least n words **then**
    exit loop
  end if
end for
return L

The frequency ratio attack works very well against both naive Bayes and maxent, but the best corpus for naive Bayes is the worst for maxent, and vice versa. Overall, naive Bayes is somewhat easier to defeat, requiring fewer than 300 words from any corpus.

4 Active Attacks

The basic theory of active attacks is that through active experimentation, an attacker can discover a word list that is guaranteed to be effective against a particular filter. One simple experiment is to send thousands of email messages, each with a few random words added, as proposed by Graham-Cumming (2004). If the attacker can detect which ones are marked as spam, then the attacker knows which sets of random words are effective and can use them again in the future. However, this method depends heavily on finding random groups of words that are sufficient for getting the message through. If the message is always blocked, then the attacker learns very little. In this section, we present two methods that avoid this problem and provide more information about which words are helpful.

See (Lowd & Meek, 2005) for a provably effective algorithm that tackles a related adversarial problem: getting a single message past a spam filter with minimal modification.

4.1 First-N Words

We first consider a method for finding fixed-size lists of words with negative weights (i.e., good words). An algorithm to find a list of good words using minimal queries is listed in Figure 3.

Our algorithm begins by calling the subroutine FINDWITNESS listed in Figure 2. Starting from two messages, one classified as legitimate and one classified as spam, it finds another legitimate/spam message pair that differ by only one word. This is accomplished by removing words from the legitimate message, then adding ones from the spam message, until the message is classified as spam. Both messages in the new pair are likely to be near the boundary between spam and legitimate messages because they only differ by a single word. The FIRSTNWORDS algorithm tests each dictionary word by adding it to the barely-spam message. If the message is classified as legitimate with the addition of the word, then it must be a good word. This process halts after n good words have been found.

However, the result of the FindWitness subroutine in Figure 2 could be a pair of messages that differs by the word with the largest weight. It is entirely possible that no, or very few, other word changes will suffice to make the spam message legitimate again. In practice, this is rare, especially when the set of features is large. If this should happen, one can simply attempt to rerun the algorithm starting from a different message pair or using a larger word list.

4.2 Best-N Words

The First-N Words algorithm can be refined to identify a list of good words with the largest weights, enabling more effective good word attacks. Our modified algorithm is listed in Figure 4.

As before, our algorithm uses the FINDWITNESS subroutine to identify a barely-spam/barely-legitimate message pair. As in the First-N attack, it builds a list of negative words by adding each dictionary word to the barely-spam message and sending it through the filter. However, it also builds a list of positive words in an analogous procedure starting from the barely-legitimate message.

Each positive word partitions the negative words into two sets: words of greater magnitude than the positive word, and the words of lesser magnitude. Starting from the barely-spam message, the algorithm adds a positive word and then each negative word in turn. If a resulting message is classified as spam, then its negative word's weight has greater magnitude than the positive word's weight.

Figure 4: BESTNWORDS(M_{spam}, M_{legit})

$(M_{\epsilon+}, M_{\epsilon-}) =$ FINDWITNESS(M_{spam}, M_{legit})
$S \leftarrow \{w \in W | M_{\epsilon-}$ with w is classified as spam$\}$
$L \leftarrow \{w \in W | M_{\epsilon+}$ with w is classified legitimate$\}$
$L_{best} \leftarrow \emptyset$
for each spammy word $w_+ \in S$ **do**
  $L_{small} \leftarrow$
  $\{w_- \in L | M_{\epsilon+}$ with w_+, w_- classified as spam$\}$
  $L_{large} \leftarrow$
  $\{w_- \in L | M_{\epsilon+}$ with w_+, w_- classified legitimate$\}$
  if $|L_{best}| + |L_{large}| < n$ **then**
    remove all words in L_{large} from L
    add all words in L_{large} to L_{best}
  else
    remove all words in L_{small} from L
  end if
  if L remains unchanged for 10 iterations **then**
    augment L_{best} with $n - |L_{best}|$ words from L
    exit loop
  end if
end for
return L_{best}

Since the goal is to find the best n words, the algorithm then reduces the set of negative words under consideration. If the set of greater magnitude words is larger than n, then it never needs to consider any of the lesser magnitude words. On the other hand, if the set of greater magnitude words is smaller than n, then this entire set is a subset of the best n words, so the algorithm focuses future iterations on the less-negative set. If all negative words have greater magnitude than the current positive word, or all have lesser magnitude, then the algorithm learns nothing. The algorithm halts when it has found the n best words, or when 10 positive words in a row yield no progress. In the latter case, its list of n words is a combination of the best words found and random negative words still under consideration.

As with the First-N attack, this algorithm can fail. It is possible for the spammy words to all be of greater magnitude than the legitimate words, in which case no ordering is possible. Even so, this algorithm will never produce a less effective word list than the First-N attack.

The number of queries can probably be reduced further by incorporating passive attack heuristics. For example, the Best-N attack could be run with only the words with best and worst frequency ratios. While this approach might miss a few of the best words, it would also cut down on queries by skipping most of the low-weight words.

The relative cost of making queries versus adding more words to an email will depend on the spammer's specific economic model. If the word list obtained is used to get 10 million emails past a spam filter, then 100,000 query emails is a reasonable cost. On the other hand, if the particular spam filter being attacked only guards 1,000 email accounts, then even 10,000 queries might be too expensive. While the procedures for sending (and checking) query emails may be more complicated than for spam, both the First-N attack and the Best-N attack can be parallelized to a large extent by testing many candidate words at once.

4.3 Results

We applied both the First-N and Best-N attacks to each filter 100 times. Our dictionary consists of the 23,000 words from the first English ispell dictionary, english.0, that have non-zero weight in our filters. We ran each trial until we found a set number of words which we varied from 100 to 5,000. Average word weights and query counts are shown in Figure 5.

In the active attacks, we see that Best-N attacks can be extremely effective against maxent. In fact, even First-N attacks against maxent are more effective than Best-N attacks against naive Bayes. This suggests that the larger feature weights found in maxent filters make them more susceptible to active attacks.

To reduce maxent feature weight magnitudes, we trained a new filter using a smaller variance for the Gaussian prior, $\sigma^2 = 0.01$ instead of 0.1. Against the new filter, each good word attack required 60%-100% more words – a substantial increase in adversary difficulty, but still more vulnerable to attack than the naive Bayes filter. More queries were required as well, but never more than three times as many. More robust filters could perhaps be trained using even smaller variances, but this would reduce accuracy in the absence of an attack.

While the Best-N attacks are more effective, they come at a price: the Best-N attacks require over 60,000 queries on average, while the First-N attacks require under 5,000. Often, the First-N attacks took even fewer queries: over 60% of the time, the attack found 100 features in under 1,000 queries.

Table 3 compares the most effective passive and active attacks. The table presents the number of words required to get half of blocked spam labeled as legitimate for different good word attacks. The number of queries required to identify those good words is presented in parentheses. The better attacks require more queries, but drastically reduce the number of words required.

5 Spam Filter Responses

Under a siege of good word attacks, what hope does a spam filter have? If the addition of a few dozen words increases

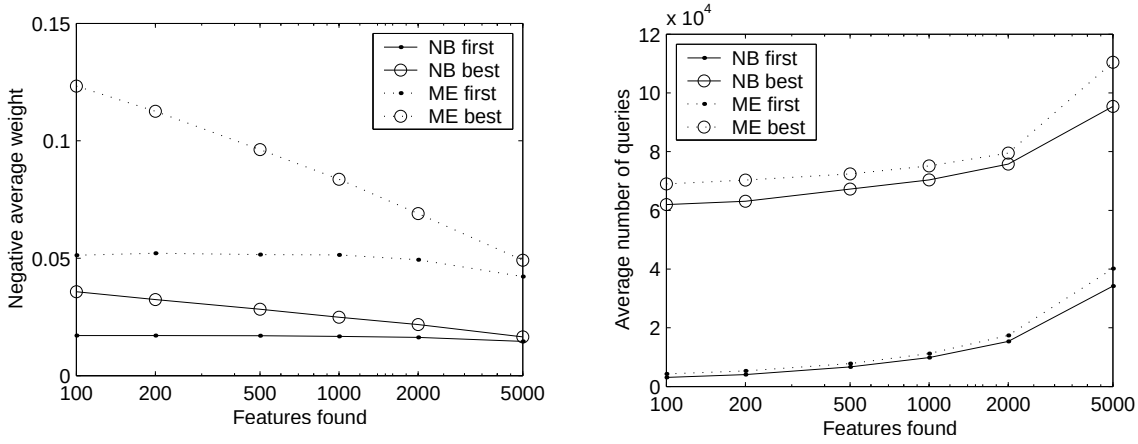

Figure 5: Average negative feature weight and average number of queries required for each type of active attack against each filter.

Table 3: Comparison of Good Word Attacks

Attack type	Naive Bayes words (queries)	Maxent words (queries)
Passive	112 (0)	149 (0)
Active (first)	59 (3,083)	20 (4,262)
Active (best)	29 (61,982)	9 (68,987)

the false negative rate to over 50%, is filtering even worthwhile? To answer these questions, we conducted a preliminary study of possible defenses against these attacks.

If each good word attack is merely a list of words added to a spam, then it should be possible to detect an attack by looking for such lists of unrelated words. Unfortunately, an attacker can easily disguise a good word attack by turning the word list into plausible text. A simple way to do this is to search the internet for paragraphs that are rich in these words. A more sophisticated method could use a language model for automatically generating sentences from words.

Alternatively, one might hope to foil active attacks by providing a noisy filter that gives inconsistent responses. While this would defeat the algorithms we present, such an approach does not seem effective in general. For instance, if the messages are misclassified with a fixed probability, then an adversary can easily compensate by issuing more test messages to average out the misclassified examples. A more interesting approach to adding misclassification noise is to increase the misclassification rate for messages near the classification boundary. However, such an approach potentially provides additional information to the attacker about the proximity to the boundary and again might only force the attacker to use more test messages in identifying good words. Furthermore, while this approach might make a filter more difficult to attack, it directly increases the error rate through intentional misclassification.

Dalvi et al. (2004) present a proactive approach that anticipates attacks and adjusts the filter to compensate for them. However, they make the unrealistic assumptions that the attacker has complete knowledge, an optimal strategy, and no knowledge of the filter adjustments. It remains to be seen if this approach works in more realistic scenarios with imperfect yet tenacious adversaries.

The simplest approach, and the only one we have found to be effective, is frequent retraining. With this method, good words that are overused by spammers soon cease to be good. The actual effectiveness of retraining depends on the relative responsiveness of the attacker and the filter. We assume (and hope!) that the filter can be retrained faster than spammers can find new word lists.

5.1 Preliminary Retraining Experiments

We conducted preliminary experiments to test how well retrained spam filters fare against active attacks, the most dangerous of the good word attacks. The precise effectiveness of retraining will inevitably depend on many factors, such as the frequency of the good word attacks and the rate at which you acquire labeled training data. Our study does not explore all dimensions of this problem; instead, it uses one plausible scenario to demonstrate the potential impact of retraining. A more thorough investigation is left for future work.

To augment a spam message with a good word attack, we added random words one at a time from a list of 1,000. We used 1,000 words because that allows for some variance in the attacks, presenting a greater challenge for the classifier. Once a modified spam message got past the filter, we stopped adding words. If a spam was still blocked after adding 100 words, we gave up and used the original unmodified message. By adding only as many words as necessary, we gave the spam filter less useful information.

Table 4: Percent Decrease in Blocked Spam

	First-N Orig.	First-N Attack	Best-N Orig.	Best-N Attack
Orig. NB	0.00%	62.52%	0.00%	87.22%
ME	0.00%	99.76%	0.00%	99.94%
Retrain NB	-0.02%	-0.02%	-1.38%	-1.38%
ME	-0.60%	-0.30%	-0.90%	0.15%

We randomly split the test set into two sets of 5,000 messages each, one for retraining and one for retesting. Spam messages in the retraining set were modified using both First-N and Best-N word lists, against both naive Bayes and maxent spam filters. (Legitimate messages from the retraining set remained unmodified.) The resulting modified versions of the retraining set were used as additional data for retraining their respective filters. We gave the 5,000 modified messages higher weight relative to the 500,000 original training examples by duplicating them before retraining. We tried numbers from one to 100 and settled on the number of duplicates that maximized performance on both modified and unmodified test data, giving equal weight to each. We then measured the accuracy of the original and retrained filters on the retesting set, with and without good word attacks.

5.2 Retraining Results

Table 4 lists the relative decrease in blocked spam for each type of filter under each scenario. For example, a value of zero means that the same amount of spam is blocked as in the baseline case, with no attack and no retraining. Negative values represent an increase in accuracy. For example, a value of -10% means that 10% more spam are blocked than in the baseline case. The first two rows show the degradation of the original spam filters under each attack. The second two rows show how well the retrained filters perform, both with and without each good word attack. Both the "First-N Orig." and "Best-N Orig." columns show performance on the unmodified retesting set; the only difference is in the retrained filters, which were trained using First-N or Best-N attack data, respectively.

After retraining, both naive Bayes and maxent demonstrate good accuracy against both attacks, often exceeding their original accuracies. While maxent was originally more susceptible to the attacks, it does just as well after retraining. This suggests that maxent depends more on frequent retraining than naive Bayes.

6 Conclusion

By adding a relatively small number of easily found words, an attacker can get 50% of currently blocked spam past a typical spam filter. While current good word attacks may be less sophisticated, any weakness of current spam filters will eventually be exploited. Active attacks are the most effective, but good words can still be found without issuing a single query. This means that no spam filter is safe. The only remedy we know of is frequent retraining: if we cannot prevent attacks, we can still seek to limit their impact. Future work includes characterizing other spam attacks (e.g., word obfuscation); exploring the relative vulnerability of other types of spam filters; finding better defenses against good word attacks; assessing the effectiveness of retraining in more varied scenarios; and further analyzing the effect of priors on the feature weight distributions of naive Bayes and maxent filters.

Acknowledgements

The authors thank Mary Lowd and Janet Schmidt for detailed comments on an earlier draft, and Geoff Hulten for helpful discussions. This research was partially done while the first author visited Microsoft Research, and was partially funded by an NSF Graduate Research Fellowship awarded to the first author.

References

Berger, A., Pietra, S. D., & Pietra, V. D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics, 22*.

Dalvi, N., Domingos, P., Mausam, Sanghai, S., & Verma, D. (2004). Adversarial classification. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004)* (pp. 99–108). Seattle, WA: ACM Press.

Graham-Cumming, J. (2004). How to beat an adaptive spam filter. *MIT Spam Conference*. Cambridge, MA.

Lowd, D., & Meek, C. (2005). Adversarial learning. *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2005) (to appear)*. Chicago, IL: ACM Press.

Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). A Bayesian approach to filtering junk E-mail. *Learning for Text Categorization: Papers from the 1998 Workshop*. Madison, Wisconsin: AAAI Technical Report WS-98-05.

Wittel, G. L., & Wu, S. F. (2004). On attacking statistical spam filters. *Proceedings of the First Conference on Email and Anti-Spam (CEAS 2004)*. Mountain View, CA.

Zhang, L., & Yao, T. (2003). Filtering junk mail with a maximum entropy model. *Proceedings of the 20th International Conference on Computer Processing of Oriental Languages (ICCPOL 2003)* (pp. 446–453). ShenYang, China.