Bachelor thesis
The Effects of Different Bayesian Poison Methods on the Quality of the Bayesian Spam Filter 'SpamBayes'

Martijn Sprengers
m.sprengers@student.ru.nl
s0513288
Supervised by
prof. dr. T.M. (Tom) Heskes
tomh@cs.ru.nl

Radboud University Nijmegen

Abstract. As long as e-mail exists, spam will be a part of our society. Nowadays, between 90 and 98% of all e-mail is spam. Fortunately, more and more methods are available to stop spam, including Bayesian spam filtering. This kind spam of filters are widely used in a variety of programs to intercept spam e-mails based on their Bayesian probabilities. However, spammers seek new ways to degrade the effectiveness of these spam filters. Bayesian poisoning is one of those techniques and it is based on adding words that are more likely to appear in non-spam e-mails than in spam, in order to let the spam filter believe the message is legitimate. The goal of this study is to give insights into different Bayesian Poisoning methods and to compare their effects on the performance of a Bayesian spam filter called 'SpamBayes'. We evaluate three poisoning methods in several empirical experiments. It turns out that one method works as expected and drastically decreases the performance of the spam filter. However, the other two, against our expectation, surprisingly improve the performance of the spam filter when the percentage of poisoned training data is increased.

1 Introduction

E-mail spam involves an act where nearly identical messages are sent to numerous e-mail addresses. Spam has grown exponentially since the beginning of the nineties. Spam is a part of the mail system, the Internet and increasingly the society. Nowadays, billions of spam massages are sent a day [1], leading to a huge amount of e-mail users being frustrated or confused. Most spam is automatically sent by bot nets or networks of virus-infected computers. Spam has proved to be effective so it will not disappear easily [2]. Since this problem costs the society a huge amount of time and money, estimated to be 130 billion dollars this year [3], several anti-spam techniques have been developed over time. These techniques proved to be successful, so spammers are in search of new ways to get past the anti-spam techniques. This inspired scientists, companies and governments to develop better spam anti-spam techniques which in their turn inspired spammers to make even smarter anti-anti-spam techniques, and so on. This war will continue, so more research is still needed to defeat the spammers. This research is meant to give more insight into Bayesian poisoning techniques and their influence on Bayesian spam filters.

The main topic of this research is to measure the effects that Bayesian Poisoning has on one single Bayesian spam filter, called 'SpamBayes' [4]. For an answer on the more general question, How does Bayesian Poisoning affect Bayesian Spam filtering?, more spam filters and datasets have to be tested. SpamBayes is an open source project initially started by Paul Graham in a research to defeat spam [5]. The program has been improved a lot, especially by Gary Robinson and Tim Peters. SpamBayes is now widely used because it has a low number of false positives and false negatives. The difference between a conventional Bayesian filter and the filter used by SpamBayes is that there are three classifications rather than two: spam, non-spam (called ham in SpamBayes), and unsure. If the message is classified as unsure, a human decision is needed.

To poison a Bayesian spam filter, several methods have been proposed in the literature. Gregory et al. [6] did this by adding several random words into spam messages. Stern et al. [7] used a method based on adding common English words to the dataset. Lowd and Meek [8] added words that are more likely to appear in no-spam (ham) than spam. It is a part of the research to determine the best way of poisoning. This can the a combination of the three methods mentioned above.

This research is based on a greater dataset than the earlier mentioned researches. The dataset used in this project, is the TREC spam corpora. Initially, the 2007 version [9] was meant to be used. After some investigation, I came to the conclusion that the TREC 2006 corpora [10] have a better structure. Using these corpora, it is easier to test those on a spam filter and to poison it. Also, the 2007 version is much bigger (over 74000 messages) than the 2006 version (about 37000 messages), which will make the test phase significantly longer but would not add a lot of accuracy.

This research solely consists of an evaluation of one Bayesian spam filter and one attack technique. It is not in the scope of this research to test other Bayesian spam filters. In addition to this, it is also not in the scope to test other anti-spam techniques like sender evaluation [11], checksum-based filtering [12] or HELO checking [13]. However, SpamBayes is representative for other Bayesian spam filters so the outcome of the research can be generalized.

This research consists of a comparison between different Bayesian poisoning methods. First, we will introduce the Bayesian spam filtering technique and show how SpamBayes uses that technique for classifications of e-mail messages. After that, we will describe the poisoning methods, followed by a description of the implementations. Finally, we will describe the experimental setup and draw conclusions from the experimental evaluation.

2 Bayesian spam filtering

This section will explain how Bayesian spam filtering works and the mathematical background behind it. Bayesian spam filters use the theorem of Bayes to predict whether a e-mail message is ham or spam. The theorem of Bayes, found by Laplace in 1812 [14] but based on the research by Bayes [15], relates the conditional and marginal probabilities of two random events. It can be expressed as follows:

$$\Pr(A|B) = \frac{\Pr(B|A)\ \Pr(A)}{\Pr(B)}. \tag{1}$$

Here,

- $\Pr(A)$ is the prior probability of A. It is "prior" in the sense that it does not take into account any information about B;
- $\Pr(A|B)$ is the conditional probability of A, given B, also called the posterior probability;
- $\Pr(B|A)$ is the conditional probability of B, given A;
- $\Pr(B)$ is the prior probability of B.

2.1 Using Bayes to classify a word in an e-mail

Sahami et al. [16] were the first to use the Bayesian theorem for classifying e-mails into spam and ham. For example, if you want to classify an e-mail containing the word 'Viagra', the probability that the e-mail is indeed spam can be calculated as follows:

$$\Pr(S|W) = \frac{\Pr(W|S) \cdot \Pr(S)}{\Pr(W|S) \cdot \Pr(S) + \Pr(W|H) \cdot \Pr(H)}. \tag{2}$$

Here,

- $\Pr(S|W)$ is the probability that a message is a spam, knowing that the word 'Viagra' (W) is in it;
- $\Pr(S)$ is the overall probability that any given message is spam;

– $\Pr(W|S)$ is the probability that the word 'Viagra' appears in spam messages;
– $\Pr(H)$ is the overall probability than any given message is not spam (ham);
– $\Pr(W|H)$ is the probability that the word 'Viagra' appears in ham messages.

Between 90% and 98% of all messages is spam [17], thus $\Pr(S)$ and $\Pr(H)$ $\Pr(S)$ should have values 0.9 and 0.1 respectively. However, most spam filters are not biased: they do not make assumptions about the incoming e-mail and consider both events to have the same probabilities (where $\Pr(S)$ and $\Pr(H)$ are both 0.5). Gary Robinson's research on spam explains why [18]:

> *In the real world, a particular person's inbox may contain 10% spam or 90% spam. Obviously, this will have an impact on the probability that, for that individual, an e-mail containing a given word will be spam or ham. That impact is ignored in the calculation* (Equation (2))*. Rather, that calculation, in effect, approximates the probability that a randomly chosen e-mail containing a word w would be spam in a world where half the e-mails were spams and half were hams. The merit of that approach is based on the assumption that we do not want it to be harder or easier for an e-mail to be classified as spam just because of the relative proportion of spams and hams we happen to receive. Rather, we want e-mails to be judged purely based on their own characteristics. In practice, this assumption has worked out quite well.*

The previous assumption on treating an incoming e-mail unbiased, where $\Pr(S)$ and $\Pr(H)$ are both 0.5, leads to a generalization of the formula, which is as follows:

$$\Pr(S|W) = \frac{\Pr(W|S)}{\Pr(W|S) + \Pr(W|H)}. \tag{3}$$

$\Pr(S|W)$ is now called the 'spamicity'. The probabilities of $\Pr(W|S)$ and $\Pr(W|H)$ need to be calculated via a training phase of the model. It is essential for the estimator that it is trained on a set of messages which is representative and has a reasonable size.

2.2 Dealing with rare words

If the calculations of words in e-mails are done in the previously described way, rare words could be a source of problems [18]. Consider the following: if the word 'replica' appears in only one message and that message is spam, the spamicity of the word will be calculated as '1'. This means that every next e-mail containing the word 'replica' will be classified as spam. However, in the real world this is not the case because you do not have enough information to be absolutely certain that all future mail will be spam if it contains the word 'replica'.

Bayesian statistics gives a technique to deal with rare words. It is based on the 'degree of belief' which a person has in a particular event, namely the Bayesian probability of that event. If a person gets an e-mail which contains a particular word and it is spam, the degree of belief of the next time he sees that word in a message and thinks it is spam, is not 100%. This is because the person has his own background information on which the decision is based: he knows that every word can appear in either ham-messages or spam-messages and that only a few training points are not enough to be completely certain about the probability of a word being spam. Bayesian statistics combines the general background information with the collected data for a word so that both aspects have equal importance.

This degree of belief can be calculated as follows:

$$F(S|W) = \frac{sx + n\Pr(S|W)}{s + n}. \tag{4}$$

Here, $F(S|W)$ is the degree of belief, s is the strength we want to give to the background information, n is the number of e-mails we have received that contain word W, $\Pr(S|W)$ is the spamicity of the word as calculated by Equation (3) and x is our assumed probability, based on our general background information, that a word we do not have any other experience of will first appear in a spam.

The values of s and x are implementation specific, but various tests to optimize performance suggest that reasonable values would be 1 and 0.5 respectively. If we do not have enough data or no data at all to do a good prediction, where $\Pr(S|W)$ and n are near 0, $F(S|W)$ is almost the same as our assumed probability based on background information (x).

The derivation of this formula is based on the principles of Bayesian statistics. It assumes that a classification of an incoming e-mail containing the word W is a binomial random variable with a beta distribution prior, as described by [19]. The posterior expectation after incorporating the observed data can be calculated. It is equivalent to a test in which one tosses a coin and counts the heads to see if it is biased. In both cases we have n trials. Each tossing of a coin could be seen as a trial. In the spam classification case, each trial is an event where we look at the next e-mail containing the word W and check if the e-mail is spam. If it is spam, the experiment is considered successful. Because there are only two cases, either spam or ham, and they are independent (the fact that one e-mail contains the word W is not correlated to the fact that the next one will) the experiment is considered binomial. If you have a binomial experiment and you assume a beta distribution for the prior, it is possible to express the expectation that the $n_{th} + 1$ trial will be successful. The expectation of success $E[y]$ can then be calculated as follows (after [18]):

$$E[y] = \frac{\alpha + q}{\alpha + \beta + n}. \tag{5}$$

Here, q is the number of successes, n is the number of trials, α and β are the parameters of the beta distribution and $y \in \{1, 0\}$ (where $y = 1$ means 'success' and $y = 0$ means 'no success').

To prove the equivalence between Equation (4) and Equation (5) perform the following substitutions first:

$$\begin{cases} sx = \alpha \\ s \quad = \alpha + \beta \end{cases}$$

Then, replace $n \Pr(S|W)$ by q. As mentioned before, $\Pr(S|W)$ is only an approximation of the probability that a randomly chosen e-mail containing the word W is spam in a world where there are equal numbers of spam and hams. Thus, $n \Pr(S|W)$ approximates the count of spams containing W. In our experiment it approximates the count of successes and therefore it is equivalent to q.

2.3 Combining the individual probabilities

Until now, we only calculated the spamicity for individual words, whether or not rare, in an e-mail. However, most e-mails consist of multiple words and headers. Thus, every e-mail can be represented by a set of probabilities. To combine these probabilities, several methods have been proposed in the literature.

Naive Bayes combining. This method is based on the fact that the spam filter implementing it sees the words in an e-mail as independent events. This assumption makes this method naive, because for most (natural) languages holds that the appearances of words are not independent events. For example, if you start a sentence with a pronoun, it is more probable that the following word will be a verb than an adjective. If one makes the above assumption, you could see the spam filter as a naive Bayes classifier and combine the individual probabilities as shown below[20] [21]:

$$\Pr(M) = \frac{p_1 \cdot \ldots \cdot p_n}{p_1 \cdot \ldots \cdot p_n + (1 - p_1) \cdot \ldots \cdot (1 - p_n)}. \tag{6}$$

Here,

- $\Pr(M)$ is the probability that message M is spam;
- p_1 is the probability $\Pr(S|W_1)$ or $F(S|W_1)$ for the first word in W as calculated in Equation (3) and (4) respectively;
- p_n is the probability $\Pr(S|W_n)$ or $F(S|W_n)$ for the n-th word in M as calculated in Equation (3) and (4) respectively.

If the classifier has calculated the combined score, it usually compares it with a given threshold to decide whether it is spam or not.

Chi-squared combining. There are more methods to combine individual probabilities. One of them is called the chi-squared combining. This method, due to R.A Fisher [22], is widely used for meta-analysis. It is based on combining the results from a variety of independent tests bearing upon the same overall null hypothesis as if in a single large test. Again, consider $p_1 \dots p_n$ as the individual probabilities from an e-mail as used in Equation (6). If we want to combine those probabilities, mostly with extreme values, into a combined score, we do the following:

$$\chi^2_{2n} = -2 \sum_{i=1}^{n} \ln(p_i). \tag{7}$$

Here, χ^2_{2n} is a chi-squared distribution with $2n$ degrees of freedom (number of independent pieces of information on which the estimate is based). The $p - value$ itself can be found via an interpolation on the chi-square table.

Now consider that all p_i's are interpreted as $p - values$ in independent tests. Using Fisher's method, those single $p - value$'s could then be combined into one single $p - value$. If an e-mail contains a lot of ham words (words that show up in more ham e-mail than in spam), the Fisher calculation is then very sensitive to evidence of hamminess. This is because probabilities near 0 have a great influence on the product of probabilities, on which Fisher's calculation is based [23]. To see this, consider Equation (7) again. Because the sum of logs is the same as the log of products, we could rewrite Equation (7) into a measure for ham scores:

$$H = \chi^{-2}_{2n}(-2 \ln(\prod_{i=1}^{n} p_i)) \tag{8}$$

Here, H is the combined score for ham, χ^{-2}_{2n} is the inverse chi-square function (used to derive a p-value from a chi-square-distributed random variable) with $2n$ degrees of freedom and p_i is an individual probability for one word. For example, the influence of the product $0.01 \cdot 0.5 = 0.05$ is higher if the first term changed to 0.001 then the influence of the product $0.51 \cdot 0.5 = 0.255$ if the first term changed to 0.501.

At this moment we only have a way to find the Ham score, because the Fisher technique is more sensitive to probabilities near 0. However, individual spam scores are much closer to 1 and therefore have less influence on the combined calculation. We could apply the same interpretation of p_i's for the spam score as for the ham score. If we reverse all individual probabilities by subtracting them from 1, $1 - p_i$ then represents the probability that an e-mail containing word i is ham. This will again lead to probabilities near 0 and make them suitable for the Fisher calculation. We could now again interpret all single $1 - p_i$'s as $p - values$ and combine them into one single $p - value$. The formula for the spam score would then become:

$$S = \chi^{-2}_{2n}(-2 \ln(\prod_{i=1}^{n} 1 - p_i)) \tag{9}$$

where S is the spam score.

To summarize, if the ham score is low, we have a strong evidence to say: 'the message is ham'. Otherwise, if the spam score is low, we have a strong evidence to say: 'the message is spam'. If both are low, we could say: 'mark it as unsure'.

Gary Robinson has written an interesting paper on the theory behind the Fisher calculations and how they can be used for combining individual probabilities [18]. Initially, this theory was presented in this paper. However, it turned out that Robinson's motivations were not as sound as we thought they were. He motivated that the individual probabilities $Pr(S|W)$ and $F(S|W)$ (from Equation (3) and (4) respectively) are assumed to be part of a null hypothesis. This is because they are not real physical probabilities, but more a best guess about those probabilities. He proposed this null hypothesis: *The individual probabilities are accurate, and the actual e-mail is a random and independent collection of words, such that the probabilities do not have an uniform distribution.* However, this null hypothesis is not empirical. Moreover, this null hypothesis is always rejected, because all combined probabilities for both the ham and spam are very close to zero. You then have to assume one of the two alternative hypotheses instead. Because this is all very vague, it is omitted from this research.

3 SpamBayes

This section will describe the working of the spam filter 'SpamBayes', which is used in this research. It is based on the previously described theories about Bayesian spam filtering. On top of that, SpamBayes uses some special developed techniques to generate even better predictions than 'normal' Bayesian spam filters.

SpamBayes [4] is a Bayesian spam filter written in Python which uses techniques laid out by Paul Graham in his paper "A Plan for Spam" [5]. It has subsequently been improved by Gary Robinson and Tim Peters, among others. The most notable difference between a conventional Bayesian filter and the filter used by SpamBayes is that there are three classifications rather than two: spam, ham and unsure. The user trains a message as being either ham or spam; when filtering a message, the spam filters generate one score for ham and another for spam. If the spam score is high and the ham score is low, the message will be classified as spam. If the spam score is low and the ham score is high, the message will be classified as ham. If the scores are both high or both low, the message will be classified as unsure. This approach leads to very good predictions, but it may result in a number of unsures which need a human decision.

3.1 How to combine the spam and ham scores

From Section 2.3 we know that the Fisher calculation could be used to calculate the combined probability and that SpamBayes calculates two scores for each message, a ham and spam score (Equation (8) and (9)) respectively. We want to combine those two scores in one overall indicator for ham or spam. Originally, the following calculation was used to combine these scores:

$$I = \frac{S}{S+H} \tag{10}$$

where I is the overall indicator. If I is below the ham threshold or above the spam threshold, the e-mail is marked as *ham* or *spam* respectively. If I is between those thresholds, the e-mail is classified as *unsure*. However, there is one drawback. The results of the Fisher calculations could end up with S and H values both near 0. This could happen if a friend forwards a spam to another friend as part of a conversation about spam. S could then be on the order of 0.00001 and H could be 0.000000001. Equation (10) would calculate I as $\frac{0.00001}{0.00001+0.000000001} = 0.99990001$ and SpamBayes would classify the e-mail as `spam`. This is called the 'cancelation disease': a spam filter is diagnosed with this disease if it is very sure about its classification, although there are both a lot of ham and spam words. Rob Hooft proposed a new measure for I to deal with this disease [18]:

$$I = \frac{1+H-S}{2}. \tag{11}$$

From the values of the previous example, I would now be calculated as $\frac{1+0.000000001-0.00001}{2} = 0.499995001$ and SpamBayes would classify the e-mail as `unsure`. It now needs a human decision and would not suffer from the effects of a false positive.

3.2 Example of a classification

To show how SpamBayes works, this section contains a small example of a classification. Consider the e-mail below (without the headers), as taken from [24]. The e-mail is `spam`.

```
Dear Sir or Madam:

Please reply to
Receiver: China Enterprise Management Co., Ltd. (CMC)
E-mail: unido@chinatop.net

As one technical organization supported by China Investment and Technical Promotion Office
of United Nation Industry Development Organization (UNIDO), we cooperate closely with the relevant
```

```
Chinese Quality Supervision and Standardization Information Organization. We provide the most
valuable consulting services to help you to open Chinese market within the shortest time:

1. Consulting Service on Mandatory National Standards of The People's Republic of China.
2. Consulting Service on Inspection and Quarantine Standards of The People's Republic of China.
3. Consulting Service for Permission to Enter Chinese Market

We are very sorry to disturb you!
More information, please check our World Wide Web: http://www.chinatop.net

Sincerely yours
```

Now consider Table 1, which contains the individual probabilities for the fifteen most interesting words in the e-mail. These probabilities were calculated by a real training phase. First we calculate $h = \prod_{i=1}^{15} P(i)$

Table 1. Individual probabilities for specific words

Word	Probability
madam	0.99
promotion	0.99
republic	0.99
shortest	0.047225013
mandatory	0.047225013
standardization	0.07347802
sorry	0.08221981
supported	0.09019077
people's	0.09019077
enter	0.9075001
quality	0.8921298
organization	0.12454646
investment	0.8568143
very	0.14758544
valuable	0.82347786

and $s = \prod_{i=1}^{15} 1 - P(i)$ as $1.11659614236 \cdot 10^{-9}$ and $1.20254494649 \cdot 10^{-10}$ respectively. If we choose the Naive Bayes combining method, we could calculate the overall probability for the message as given in Equation (6). So $p(W) = \frac{h}{h+s} = 0.902773632441$ and the e-mail gets classified as `spam`.

If we choose the chi-squared combining method, we first have to calculate H and S as shown in Equation (8) and (9). The outcomes are 0.916780561948 and 0.966693268343 respectively. To combine these score we use Equation (11): $I = \frac{1+0.916780561948-0.966693268343}{2} = 0.524956353197$. Now, the e-mail is classified as `unsure` and thus score the Naive Bayes combining better in this example. However, if we add one word with a low spam probability or if we remove one word, the overall spam probability calculated by the Naive Bayes combining method changes drastically. On the other hand, the overall spam probability calculated by the chi-squared combining method changes just a little bit. For example, if we add the word 'Permission' with a individual probability of 0.11562342, the Naive Bayes and chi-squared methods calculate the combined probabilities as 0.548320089676 and 0.502123986337 respectively.

4 Bayesian poisoning

Bayesian poisoning is a way to attack a Bayesian spam filter. As there are more ways to attack a spam filter, it is nice to have an overview. Most ways that spammers use to attack the spam filters can be grouped into several categories, as described by Gregory et al. [6]. Below is a survey of them.

Tokenization This attack is based on disturbing the feature selection (also called tokenization) of a spam filter on an e-mail. This could be done by splitting or modifying the key features in the messages, like splitting up words with spaces or using (HTML) layout tricks.

Obfuscation In this way of attacking, the spammer obscures the content of the e-mail by using encoding or misdirection. Examples include letter substitution or HTML entity/URL encoding.

Statistical This kind of attacks concentrates on the messages statistics and thus on the core of the Bayesian filter. Such attacks attempt to alter the message's statistics such that the filter has trouble telling if a message is spam or not. The statistical category can be divided in two subcategories:

- *Weak Statistical*
  An attack is termed weak if its approach uses purely randomized data. Adding random words, fake HTML tags, or random text extracts are common versions of such attacks.
- *Strong Statistical*
  A strong statistical attack is different then the weak one because it uses other data to add. This data is much more focused than the data used with weak statistical attacks. The chances of a successful misclassification are higher with these sort of data. [25] However, it is also more difficult to develop such an attack than a weak one. Another way of improving the poison could be to get somehow access to the feedback of users. In other words, it could be nice if a spammer knows which spam e-mails got through the filter, so he can found his attacks on those messages.

A Bayesian spam filter can easily be trained on tokenization and obfuscation attacks, because the text need to be readable for humans and thus there are only finite ways to do tokenization or obfuscation. It is harder for a Bayesian spam filter to deal with the statistical attacks.

Bayesian poisoning can be defined as an act where a spammer appends extra data in the spam e-mail to confuse the statistics database of the spam filter. The attack is not only based on adding words that appear more in ham than in spam (so the spam e-mail has a higher probability of being classified as ham), but also on disturbing the individual probabilities as shown in Equation (3). This way, a ham e-mail which contains words that also appear in spam e-mails has a higher probability of being classified as spam.

4.1 Poison methods

This section describes the methods used for the Bayesian poisoning in this research.

Random Words This attack method is based on the research by Gregory et al. [6]. It can be seen as a weak statistical attack, because it uses purely randomized data to add to the spam e-mails.

Common Words This attack method is based on the research by Stern et al. [7]. They added common English words to spam e-mails in order to confuse the spam filter. This attack can be seen as stronger statistical attack than the Random Words method, because the data used is less random and it contains words that are more likely to be in e-mails than the words added with the previous attack.

Ham Phrases This attack is developed in this research and tested against the other two. It is based on a huge collection of ham e-mails. From that collection, only the ham e-mails with the lowest combined probability are used as poison. The ham e-mail is then added at the end of the original spam e-mail. Most people read downwards, so the effectiveness of the message is maintained. This is also a strong statistical attack, maybe even stronger than the Common Words attack, because the words are even less randomized.

5 How to evaluate quality of a spam filter

This section will explain how the quality of a spam filter can be evaluated. In general, the quality of most classifiers is evaluated by the number (or percentage) of false positives and false negatives. In the context of the spam filter Spambayes, and in this research, we define the following:

Null hypothesis Message is *spam*.
True positive A *ham* e-mail *correctly* marked as *ham*.
True negative A *spam* e-mail *correctly* marked as *spam*.
False positive A *ham* e-mail *wrongly* marked as *spam* (Type I error).
False negative A *spam* e-mail *wrongly* marked as *ham* (Type II error).

In the literature several ways to combine those classifications are described[20]:

False positive rate is the proportion of negative instances that were erroneously reported as being positive. So

$$False\ positive\ rate = \frac{\#FP}{\#FP + \#TN}.$$

False negative rate is the proportion of positive instances that were erroneously reported as negative. So

$$False\ negative\ rate = \frac{\#FN}{\#TP + \#FN}.$$

Accuracy the accuracy is the proportion of true results (both true positives and true negatives) in the population. So

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#FP + \#FN + \#TN}.$$

However, in the context of spam filtering, is a false negative even bad as a false positive? As one can imagine, it is far more worse for an user if he does not get his ham mail (because the spam filter decides that a ham should be marked as spam) than when a user gets some spam mail in his mailbox (because the spam filter decides that a spam should be marked as ham). From the spammer's point of view, this may be very interesting. If he can find a way to poison his spam e-mails so that not only the spam e-mails are marked as ham by the spam filter, but also that the ham e-mails get marked as spam, he can benefit from it. In the beginning, if a spam filter trains on the poison, it can filter out the poisoned spam e-mails in the future. However, in the training process the words that are more likely to be ham now also occur in spam e-mails. The spam filter gets confused and starts classifying the ham mails incorrect. The users notices he misses ham e-mail after a while and configures the spam filter to loosen the ham- and spam thresholds. Finally, this may lead to more false negatives, as the spam threshold is less tight.

SpamBayes' internal test suite uses another kind of approach to express performance. For every false positive they think that the user has a loss of $10.00, for every false negative $1.00 loss and for every unsure $0.20 loss. Then, they combine these weights in a test run to one single loss. However, they do not mention the source of these numbers so it looks like they found them intuitively and have less use in scientific research. There is some research done on the cost of false positives and false negatives, but those researches only concern the annual cost per user or per company [26]. So in this research we use the model provided by SpamBayes to combine the numbers of false positives, false negatives and unsures into one single loss. One can argue that the cost of a false positive differs more or less than a factor ten from a false negative and a false negative differs more or less than a factor five from an unsure, but we think these numbers are quite relevant. So the metric used in this research is:

$$totalCost = \#FP \times \$10.00 + \#FN \times \$1.00 + \#Unsures \times \$0.20. \tag{12}$$

It is more clear to use one single metric than to compare only the numbers of misclassifications (#FP and #FN) and unsures. It is now also easier to compare the tests mutually. The only drawback with this metric

is that you can not see which misclassification has the highest share in the cost. For example, 5 false positives and 25 unsures have the same *totalCost* as 52 false negatives and 15 unsures (both $55). So to compare the tests mutually, we will not only compare the outcomes of the *totalCost* but also compare the $False\ Positive$ and $False\ Negative\ rates$.

There is one thing that is not included in the cost model, namely the trust of the users in the spam filter or the general image of spam filters. As one can imagine, it is quite frustrating for users that daily have to delete all the false negatives, search for any false positives in the spam folder or find ham e-mails between the unsures. Eventually, this may lead to less trust in the spam filter and/or spam filtering as a whole.

6 Approach and implementation

This section will describe the different poisoning methods and how they are implemented. Also the dataset used and the data flow within the research will be described. In Figure 1 the flow chart of the data is shown. The different components are explained in the next subsections.

6.1 Dataset

Originally, the TREC 07 corpora [9] was intended to be used because it contains far more e-mails (about 75 000) than the TREC 06 corpora [10] (about 37 000 e-mails). However, compared to the other researches on this topic, 37 000 instances is already a lot and representative for contemporary e-mail. So doubling the number of instances would not produce better results anymore, but also increases the duration of the test runs significant.

The dataset has exactly 37622 e-mails. The number of spam e-mails is 24786 and the number of ham e-mails is 12836. Every e-mail is complete, so all headers, words and HTML-tags are still in. The oldest message in the corpus is from 27 Juli 1992 and the newest from 3 May 2006. The e-mails within the corpus are randomly distributed among 126 folders. There is one *index* file with on every line the {type,path} tuple, where *type* ∈ {spam,ham} and *path* has the format `../data/folderNR/messageNR`.

6.2 Preprocessing

The preprossing is done in three stages, which will be described below.

1. **trec2tree** This is a Python script that reads the lines from the database's *index* file, searches the e-mails and places them the following tree:
   `Data/{Ham,Spam}/{reservoir,Set*}/`
2. **sort+group** This is a Python script that sorts and groups the corpora. When testing with SpamBayes, all e-mails will be processed in sorted order. The e-mails should all have unique names with a group number and an id number separated by a dash (eg. 0123-004556). Sort+group.py sorts the e-mails into chronological order (by topmost Received header) and then groups them by 24-hour period. The group number (0123) is the number of full 24-hour periods that elapsed between the time the current e-mail was received and the time the oldest e-mail found was received. The id number (004556) is a unique auto increment integer over all e-mails, with 000000 given to the oldest e-mail found.
3. **mksets** This is a Python script that distributes the corpora into multiple sets. It will evenly distribute the e-mails across the sets, keeping the groups evenly distributed too. This is done to use the sets with cross-validation.

6.3 Poisoning

After the preprocessing is done, the data is copied three times, so every poison method has it's own dataset. Below an overview on how the poisoning is done.

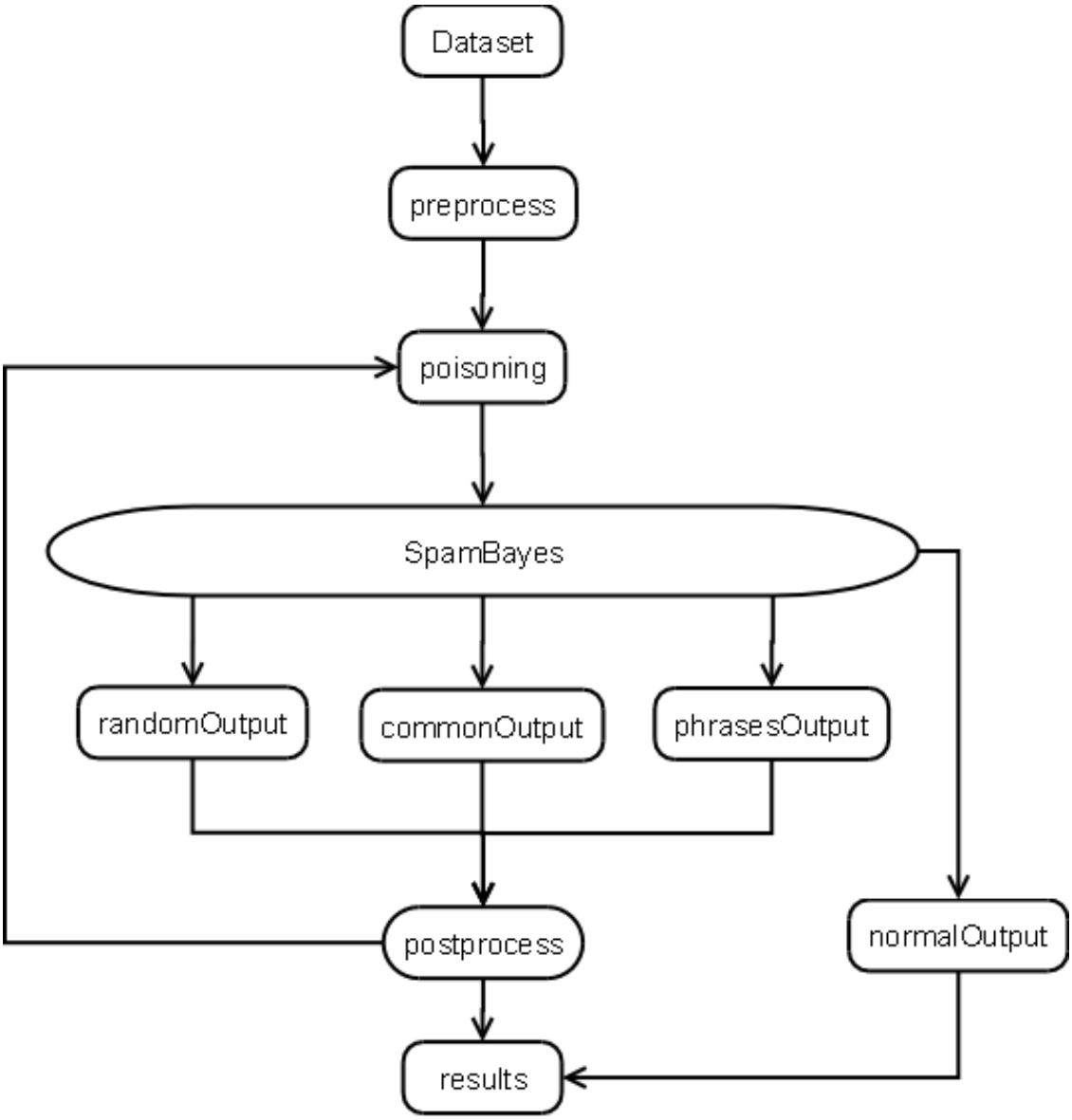

Fig. 1. General overview of the implementation and the data flow

Random Words. As described earlier, this method uses random English words to insert. The source of the words is the Unix dictionary (*Unix.dict*, available on every Linux machine in the folder /usr/share/dict/words/). Every line contains a word, making a total of 25104 English words. The user specifies which set should be poisoned and with how many words. Algorithm 1 contains the pseudo code for the poisoning.

Algorithm 1 Pseudo code for Random and Common words implementations

```
for i ∈ spamMessagesInSet do
    for j ∈ nrOfWordsToInsert do
        Randomly choose word from dictionary;
        Append word to e-mail i;
    end for
end for
```

Common Words. Like the 'Random Words' method, this method also uses a kind of dictionary to insert words. The difference with this method is the composition of the dictionary. This dictionary only consists of the most common English words. Every line is a tuple of {sort-order, frequency, word, word-class} and there are 6318 words. The list is composed by a research on the British National Corpus (BNC), which is 'a 100 million word collection of samples of written and spoken language from a wide range of sources, designed to represent a wide cross-section of current British English, both spoken and written' [27]. The insertion of the poison words goes the same as with the Random words method (Algorithm 1).

HamPhrases. As the implementation of the first two poison methods is very straightforward, this one is more complex. Now, the dictionary does not consist of words, but of e-mail messages or phrases. To generate the phrases and insert them into the training- and test sets the pseudo code shown in Algorithm 2 is executed. The algorithm only stores the messages which have a test value ($P(i)$) lower than the predefined *threshold*.

Algorithm 2 Pseudo code for the hamPhrases implementation

```
for i ∈ hamMessagesInSet do
    train(i);
end for
for i ∈ hamMessagesInSet do
    if P(i) ≤ threshold then
        Save i in poisonSet;
    end if
end for
for i ∈ spamMessagesInSet do
    Randomly choose phrase from poisonSet;
    Append phrase to e-mail i;
end for
```

The test value is the combined probability of the message, as calculated by Equation (6) or (11). From those messages in the poisonlist, the algorithm picks one randomly and adds it to the current spam e-mail.

6.4 Spambayes

This is the core component of the research. When the preprocessing is done and the dataset is placed in the right tree, SpamBayes can execute various tests on the data. The software comes with a huge test

suite to do all kinds of tests. In this research only a few tests are relevant: the incremental test and the cross-validation test. The incremental test simulates a real integrated spam filter, so it starts without any training data and it tests and trains e-mails one by one. The cross-validation test suite demands that all the data is spilt up in to several sets. Then, 80% of the sets is used for training and 20% of the sets for testing. This repeats till every set is used for testing: after the algorithm has tested set i, it trains set i, untrains set $i+1$ and then tests set $i+1$.

6.5 Postprocess

The postprocessing done depends on which way the poisoning was tested. If the incremental test (test and train messages one by one) was done, `mkgraph.py` is used to post process the test output. `mkgraph.py` is a Python script that converts the input to a csv-file which can be by most spreadsheet tools. It displays the cumulative counts per day: day 1 is the timestamp in the oldest message and the last day the newest message in the corpus. Thus every day is seen as a group of messages.

If the poison was tested with `timcv.py` (the cross-validation testing module), `rates.py` is used to post process the test output. This Python script scans the output for summary statistics and writes them to a new file. Two of those files can later be fed to `compare.py` to compare the results. The file contains a kind of statistics based on the options. In this research, the numbers of false positives, false negatives and unsures are most important. The file also displays which e-mails are wrongly classified or classified as unsure.

In this research, a *test run* is defined as an action where a percentage of the test- or training data is poisoned and then is tested by the SpamBayes' test suite. From every test run we collect the outcomes. Then, a new percentage of poison is inserted and tested again. The overall results are later compared against the outcomes of a normal run (a test run with no poison inserted).

7 Experimental evaluation

This section contains the results of the experimental study. First, the options of SpamBayes and the experimental setup will be described. After that, the results will be shown and discussed.

7.1 Parameters

SpamBayes has a lot of options that configure the spam filters. The most important of those parameters are described below. All parameters and options in the list are kept constant throughout all the tests except the *Probability combine method*. The values of the parameters were determined by intensive testing, carried out by the developers and the community of SpamBayes.

Probability combine method Originally, the current version of SpamBayes has only one implementation for combining the individual word probabilities: `chi-squared combining`. Almost all tests carried out in this research use this combining scheme. However, some tests are done with the `Gary combining`, but that will be explicitly mentioned. The chi-squared combine method appears to be the best combining method so far. One systematic benefit is its immunity to the in Section 3.1 mentioned 'cancelation disease'.

Max discriminators The maximum number of extreme words to look at in a message, where 'extreme' means that words have an individual spam probability farthest away from 0.5; so where the outcome of $|Probability - 0.5|$ is the highest.

Slurping This option is set to `false`. If this option is enabled, when a message normally scores in the 'unsure' range, and has fewer tokens than the maximum looked at, and contains URLs, then the text at those URLs is obtained and tokenized. If those tokens result in the message moving to a score outside the 'unsure' range, then they are added to the tokens for the message. This should be particularly effective for messages that contain only a single URL and no other text. However, this option is still experimental and therefore not used in this research.

Cut offs The ham cutoff is set to `0.20` and the spam cutoff to `0.90`. If the combined probability for the actual message is lower than the ham cutoff, it is classified as 'Ham'. If the probability is higher than the spam cutoff, it is classified as 'Spam'. If the probability falls between those boundaries, the e-mail is classified as 'Unsure'.

Basic header tokenizing This option is set to `false`, so the headers are not tokenized. If true, the tokenizer will tokenize the contents of each header field just like the text of the message body, using the name of the header as a tag. Tokens look like "header:word". The basic approach is simple and effective, but also very sensitive to biases in the ham and spam collections. For example, if the ham and spam were collected at different times, several headers with date/time information will become the best discriminators.

Long skip trigger length This option is set to `12`. Words longer than this value trigger a skip, so the tokenizer will neglect them.

Use mixed uni/bi-grams scheme This option is set to `false`. If it is turned on, it the tokenizer generates both unigrams (words) and bigrams (pairs of words). Because turning on this option will at least double the database size and it is also experimental, it is not used in this research.

Unknown word probability & unknown word strength These two parameters control the prior assumption about word probabilities. The unknown word probability is essentially the probability given to a word that has never been seen before. This option is set to 0.5. The unknown word probability adjusts how much weight to give the prior assumption relative to the probabilities estimated by counting. All previously conducted researches showed that a value near 0.4 worked best, so this does not seem to be corpus-dependent. The value of this option is set to 0.45.

7.2 Experimental setup

To test the different poison methods, it is not only needed to do tests with different quantities of poison, but also with different training- and test percentages. As described in Section 6, every poison method has its own dataset, plus one to test SpamBayes on the dataset without poison. Every dataset will be divided into ten smaller sets, five for the spam e-mails and five for the ham e-mails. So every dataset will look like this: `Data/{Ham,Spam}/{Set1,...,Set5}/`.

All experiments are performed on a Single-Core Pentium 4 Processor 2000 MHz with 512 MB of RAM, running Windows XP Professional SP3 x86_32. The Python programs are executed with Python version 2.6. Every test run uses four sets (80%) for training and one set (20%) for testing.

For the Random and Common words methods we have two test parameters: the number of words to insert and the percentage of poison in the training data. The HamPhrases method has also two test parameters: the minimum threshold for ham e-mails and the percentage of poison in the training data. To determine the best values for the *number of words* to insert and the *minimum threshold*, we first conduct two experiments.

Figure 2 shows the performance (in $, as calculated by Equation (12)) of the spam filter when different numbers of words are used for poisoning. We only poison the test set in this experiment. The performance of the spam filter is inversely proportional with the total cost. It turns out that and increasing number of words leads to an increasing cost (and decreasing performance). For the Random Words method, the cost starts to stabilize with 600 words as poison. On the other hand, the cost of the Common Words method still increases with the number of words, until the maximum number of spam messages in the test set is reached (there are 4958 spam messages in the test set). Both methods have opposite tendencies. If the number of words increases with the Random Words method, the number of false negatives decreases and the number of unsures increases (23 and 2707 respectively, with 2000 words poison), which is the opposite of the Common Words method, where the number of false negatives increases and the number of unsures decreases (1309 and 1437 respectively, with 2000 words poison). The number of false positives stays the same for both methods, which is trivial because no poison was inserted in the training sets yet.

The outcomes of the second experiment, to determine the best threshold for the HamPhrases method, are shown in Figure 3. This figure shows the number of ham messages that are beneath the given threshold. In the whole dataset, there are 12836 ham e-mails. We decreased the threshold until the total number of ham messages beneath the threshold has about the same size as the dictionary used for the Common Words method. A threshold of $1.0 \cdot 10^{-13}$ leaves 7584 ham e-mails (about 59% of the total) to be used for poisoning.

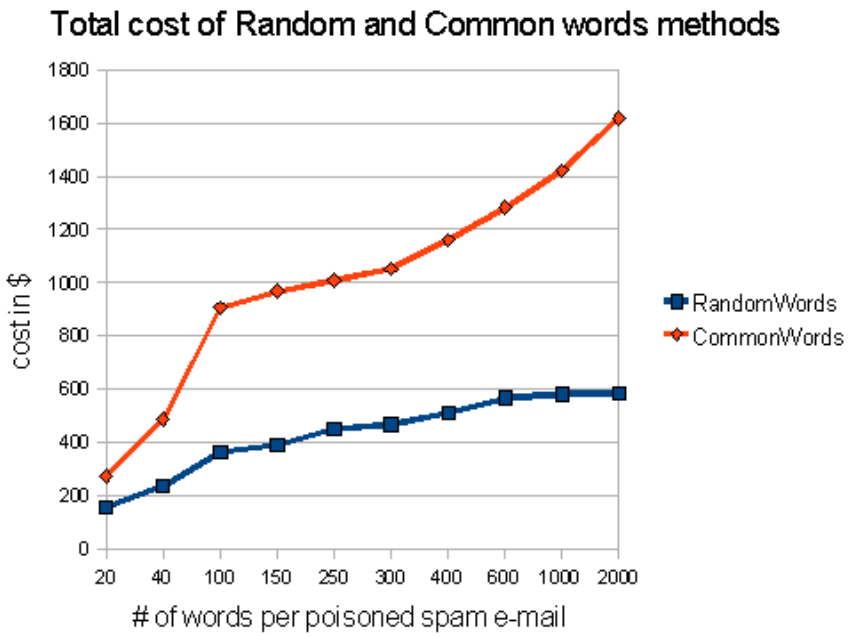

Fig. 2. Performance of SpamBayes when inserting different numbers of words as poison

7.3 Results

Figure 4 shows the performance of SpamBayes when the training sets get poisoned too. For both the Random and Common Words methods, the number of poisonwords is set to 1000. The threshold for the HamPhrases method is set to $1.0 \cdot 10^{-13}$. The yellow dotted line in the figure is the total cost ($141) when no poison is inserted (derived from 2 false positives, 76 false negatives and 225 unsures). The outcomes of the tests show that the HamPhrases method indeed works as expected: the number of false negatives decreases when more training data is poisoned, but then the number of false positives start to increase, leading to a much higher cost. When all training data is poisoned, the total cost is $962, derived from 80 FP, 72 FN and 450 unsures. On the other hand, it seems to be that the Random and Common Words techniques do not work as expected, but the opposite of it. It looks like that the performance of SpamBayes gets better when only a small percentage of the training data is poisoned. With only 25% poison in the training data, SpamBayes' classifications get even better than the normal classifications. Then applies: how more training data is poisoned, how higher the cost (and how less the the performance). However, even with 100% of the training data poisoned, the Random and Common Words techniques lead to better predictions than normal. If we look to the exact numbers, it seems that the number of false positives remains the same for all training percentages. This is against our expectation, because we expected that SpamBayes would decrease the number of false negatives at the cost of an increase in the number of false positives (when the percentage of poison in the training data is increased).

To really check if the performance of SpamBayes increases when the percentage of poison in the training data increases, we did some additional tests. These tests only focused on the Common Words method, as it is the same type of attack as the Random Words method. First, we tried another combining scheme. In Section 2.3 we described several contemporary combining schemes. However, SpamBayes initially used another combining scheme: *Gary combining*. This combining scheme is not used anymore, but it was relatively easy to implement it again. However, when we did some testing with this scheme, it turned out that it is of no use: every e-mail gets classified as 'unsure'.

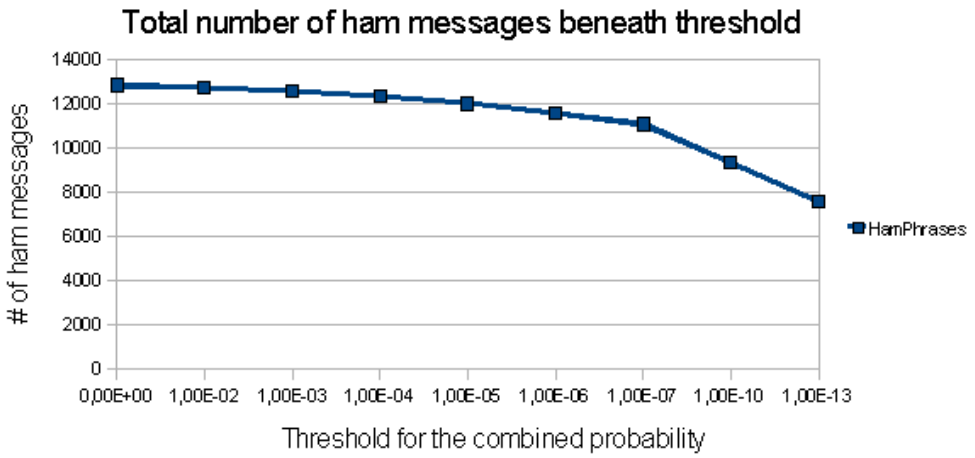

Fig. 3. Performance of SpamBayes when inserting different numbers of words as poison

Another factor could be the number of words used as poison. In Figure 5, different number of words are tested against the performance. The figure shows that when every tested spam message is poisoned with 1000 words, the total cost decrease is huge when only 25% is tested (to $44). When more training is done on the poison, the total cost increases, but it is still better than the 'normal setting' (dotted yellow line). To check if the number of words has influence on the behavior of the spam filter, we also repeated the test with less words (100 and 20 respectively). However, when training on both numbers of words this still gives an increase in performance, where a 100 words works best (only a total cost of $39,80, which is an decrease of 72% over the normal setting). We also checked the performance of the spam filter when there is no poison inserted in the test set, only in the training sets. With 20 words as poison, the red line in Figure 5 shows that the total cost decreases when there is no poison inserted in the test set (if 100% of the training set is poisoned, a total cost of $106.40 is reached, which is an decrease of 25% over the normal setting). When looking closer at the test results, it seems that SpamBayes better classifies the spam e-mails which are very long and have a lot of common words in it. This could be spam e-mails that offer a job, to promote products or services or want you to do an investment, the so-called 'pump and dump stock' spams. Most of this kind of spams are now classified correctly.

8 Conclusions, discussion and future work

From the results described in Section 7.3 we can draw several conclusions about the proposed poison methods. From a spammer's point of view, the 'HamPhrases' technique seems to work best. It does decrease the performance of the spam filter (with an average total cost increase of 600% over the normal performance) independent of the percentage of poison in the training data. The 'Random' and 'Common Words' techniques seem to score worse from a spammers point of view. Initially, when the spam filter is not trained on the poison, these techniques score very good. The total cost is then fairly high (an increase of 1000% for the Common Words technique and an increase of 400% for the Random Words technique). However, when we train the spam filter on those poison methods, the performance gets even better than normal (with an average total cost decrease of 20%). So we can conclude that training on poison generated by random or common words definitely helps to increase the performance of a spam filter and reduces the total cost.

However, the HamPhrases method used in this research is a little bit cheating. This is because both ham and spam e-mails that the spam filter uses for testing and training are available for the algorithm. Real spammers do not have the ham e-mails of real users. However, they could try to build a database of ham e-mails from open datasets. Moreover, if spammers could find a way to trace the ham e-mails or interests of

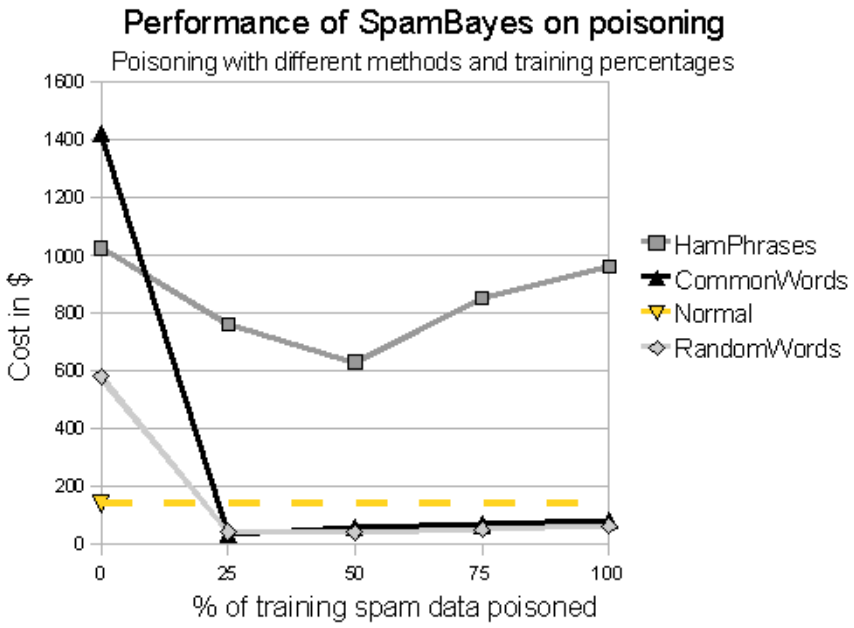

Fig. 4. Performance of SpamBayes on different poison methods

an user, they could personalize the spam e-mails. As shown in this research, those kind of spams have less trouble to get through the spam filter as others.

In this research, we only evaluated the performance of one spam filter on one dataset. In the future we could check more spam filters and more datasets. Although the dataset in this research is very big and contains different kinds of spam messages, it is not always representative. The oldest e-mail in the dataset is from 1992 and the newest from 1996, so it is based on reasonable old messages. One could argue that spam has evolved and the dataset does not reflect the contemporary e-mail anymore. For example, image spam is now rising and developers of anti-spam techniques have to evolve in the same way as the spam does [28] [29]. However, spam is only effective if it can convey its message. Most spammers find new ways to transfer that message, but it mostly comes down to getting the receiver to read text. If a spammer puts text in an image and the text could be extracted by an optical character recognition algorithm [30], one could still apply a Bayesian spam filter on the extracted text. Even so, the Bayesian technique could be used to train on images or collections of pixels and classify image spam in that way. So Bayesian spam filters could still be used in the future.

However, as long as spammers will gain revenue, spam will exist. The real solution to the spam problem does not lie in the anti-spam methods, but in the definition of e-mail. E-mail was designed to send messages between two or more persons. Over time, it became more and more a substitute for real mail. And that is where the difference lies: real mail is not free. As long as e-mail is free, spam will exist because spammers can gain a lot of money with only few financial assets.

This study started from the idea to give some insights into Bayesian spam filtering and poisoning. We presented three methods to poison a Bayesian spam filter and showed experimentally how the spam filter reacts on those methods and several percentages of poisoning. We conclude that one of the three poison methods decreases the performance of the spam filter, but the other two just increase the performance of the spam filter.

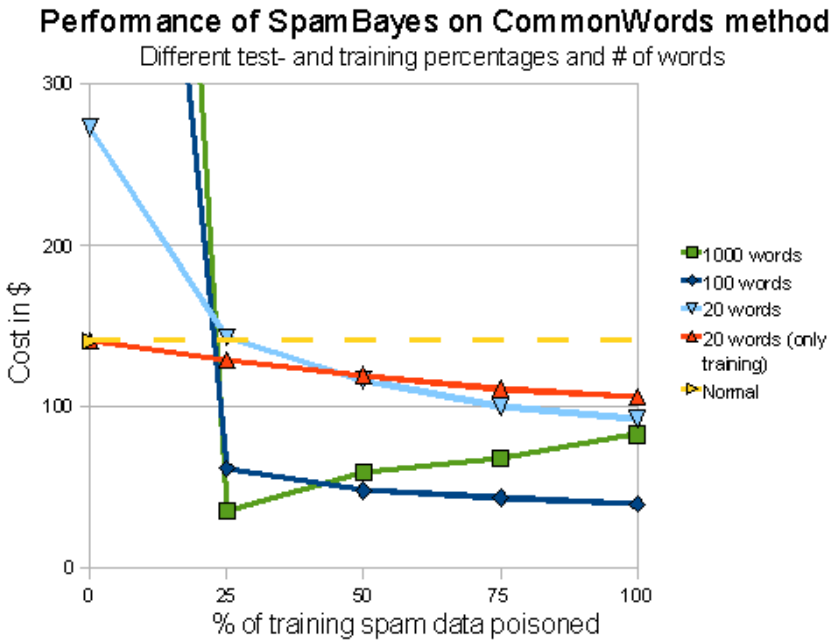

Fig. 5. Performance of SpamBayes on different training and test percentages of poison

Acknowledgement

I would like to thank prof. dr. Tom Heskes for his advice and guidance throughout this research.

References

1. SpamUnit, A.S.: Spam statistics (2008)
2. Marshal: Sex, Drugs and Software Lead Spam Purchase Growth (2008)
3. Ferris: Industrial statistics, cost of spam (2009)
4. Graham, P.: SpamBayes (2002)
5. Graham, P.: A plan for spam. URL http://www. paulgraham. com/spam. html accessed **22** (2002)
6. Wittel, G., Wu, S.: On attacking statistical spam filters. In: Proceedings of the First Conference on Email and Anti-Spam (CEAS). (2004)
7. Stern, H., Mason, J., Shepherd, M.: A Linguistics-Based Attack on Personalised Statistical E-mail Classifiers, Technical Report CS-2004-06, Dalhousie University, 2004. Submitted to the 1st Conference on E-mail and Anti-Spam (CEAS)
8. Lowd, D., Meek, C.: Good Word Attacks on Statistical Spam Filters. log **1** (2005) 1
9. Cormack, G.V., Lynam, T.R.: TREC Public Spam Corpora (2007)
10. Cormack, G.V., Lynam, T.R.: TREC Public Spam Corpora (2006)
11. Ramachandran, A., Feamster, N., Vempala, S.: Filtering spam with behavioral blacklisting. In: Proceedings of the 14th ACM conference on computer and communications security, ACM New York, NY, USA (2007) 342–351
12. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P., di Tecnologie dell'Informazione, D., Crema, I.: P2P-based collaborative spam detection and filtering. In: Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on. (2004) 176–183
13. Clayton, R.: Stopping outgoing spam by examining incoming server logs. In: Second Conference on Email and Anti-Spam (CEAS). (2005)
14. Laplace, P.: Theorie analytique des probabilites. Courcier (1820)
15. Bayes, T., Bayes, T.: An essay towards solving a problem in the doctrine of chances. 1763. MD Comput **8**(3) (1991) 157–71
16. Sahami, M., Dumais, S., Heckerman, D., Horvitz, E.: A Bayesian approach to filtering junk e-mail. In: AAAI-98 Workshop on Learning for Text Categorization. Volume 460. (1998)
17. XS4ALL: Spam Statistics 2009 (2009)
18. Robinson, G.: A statistical approach to the spam problem. Linux Journal **2003**(107) (2003)
19. Heckerman, D., et al.: A tutorial on learning with Bayesian networks. NATO ASI SERIES D BEHAVIOURAL AND SOCIAL SCIENCES **89** (1998) 301–354
20. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining, (First Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2005)
21. Rish, I.: An empirical study of the naive Bayes classifier. In: IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence. (2001) 41–46
22. Fisher, R.: Combining independent tests of significance. American Statistician **2**(5) (1948) 30
23. Littell, R., Folks, J.: Asymptotic optimality of Fisher's method of combining independent tests. Journal of the American Statistical Association (1971) 802–806
24. Graham, P.: Spam Example (2002)
25. Graham-Cumming, J.: How to beat an adaptive spam filter. In: Presentation at the MIT Spam Conference. (2004)
26. Williams, C., Ferris, D.: The cost of spam false positive. Ferris Analyzer Information Service. Report **385** (2003)
27. Kilgarriff, A.: Putting frequencies in the dictionary. International Journal of Lexicography **10**(2) (1997) 135–155
28. Goodman, J., Heckerman, D., Rounthwaite, R.: Stopping spam. Scientific American **292**(4) (2005) 24–31
29. Wu, C., Cheng, K., Zhu, Q., Wu, Y.: Using Visual Features for Anti-Spam Filtering. In: Image Processing, 2005. ICIP 2005. IEEE International Conference on. Volume 3. (2005)
30. Fumera, G., Pillai, I., Roli, F.: Spam filtering based on the analysis of text information embedded into images. J. Mach. Learn. Res. **7** (2006) 2699–2720