

UNIVERSITY OF HERTFORDSHIRE

Faculty of Engineering and Information Sciences

MCOM0177 Computer Science MSc Project

Final Report

January 2009

Word distribution vs. trigrams: A comparative study on
Bayesian-based spam filters

Yousif A. Al Saif

Abstract

A comparative study between a Naïve Bayes spam filter that uses word-based tokenization and a Naïve Bayes spam filter that uses trigram-based tokenization was conducted. A word-based and a trigram-based spam filters were implemented using the same classifier and trainer differing only in their tokenization scheme. The study compared the efficacy of both filters using the same set of emails for training and testing whilst employing performance measures common in previous research in spam filtering. Results of the study were in-favor of word-based spam filtering when the amount of pre-categorized emails available for training are limited and when the resources available for the classification process were limited as well. Given an abundance of resources available and a sufficient amount of emails, results suggest that trigram-based spam filtering is more suitable due to its higher reliability and accuracy.

Acknowledgments

During my studies, there were times when personal circumstances made me believe that I would not be able to finish the journey I started. It is the continued emotional support of my parents and their words of encouragement that gave me the persistence needed to finish the work I started. No words of thanks could express the depth of appreciation and love I hold for my parents Dr. Abdulnabi and Mrs. Batool.

I would also like to take this opportunity to express my gratitude to my project supervisor Dr. Iain Werry. Your valuable constructive criticism, prompt feedback, and patience with my needy nature allowed me to complete this piece of work and for that I am very grateful. Thank you.

Contents	Page
Abstract	
Acknowledgments	
1. Introduction.....	2
2. Related Work.....	10
3. Tokenization.....	12
3.1 Word-based Tokenization.....	14
3.2 Trigram-based Tokenization.....	16
3.3 The Datasets.....	17
4. The Naïve Bayes classifier.....	19
4.1 Training and classification.....	22
5. Evaluation Methodology.....	24
6. Results and Analysis.....	26
7. Discussion and conclusions.....	31
Bibliography.....	38
Appendix A.....	43
Appendix B.....	45
Appendix C.....	48
Appendix D.....	49
Appendix E.....	52
Appendix F.....	55
Appendix G.....	59
Appendix H.....	63
Appendix I.....	65
Appendix J.....	66

1. Introduction

Email is perhaps one of the major selling points of the Internet. With the growing popularity of the Internet and the increase in the adoption rate, the issue of maintaining email systems is – without doubt – causing major concerns to server administrators. Many businesses are starting to depend completely on email for communications with the outside world and there is a great interest in providing efficient infrastructure that is capable of handling the anticipated load increase on mail servers. However, the increasing popularity is not the biggest concern to mail server administrators. Spam – also known as unsolicited electronic mail – constitutes a major hurdle on mail servers with recent reports (Bowers & Harnett, 2008) estimating that roughly %80 of all electronic mail is spam. Spam puts a great strain on the IT infrastructure of businesses consuming precious network resources and affecting the productivity of end-users. This – in effect – raises the budgetary requirements for maintaining an email system as well as affecting the overall throughput of businesses with a recent article (Roberts, 2004) estimating that spam costs business \$874 worth of productivity per employee per year. Moreover, spam has been recently used to spread malicious software and phishing scams causing security concerns regarding the use of email in businesses.

Spam is proving to be a very popular medium to questionable and controversial businesses. This is mostly due to the comparably low cost associated with sending spam. Unlike unsolicited snail mail which costs advertisers printing and postage, spam uses compromised computers – also known as botnets – and other dubious means to send spam lifting the burden of cost from the sender/advertiser to other unsuspecting victims as well as the recipient's email system (Bowers & Harnett, 2008). The extremely limited cost associated with sending spam allows advertisers to ignore the extremely low rate of return on advertisement campaigns – which was estimated by a recent study (Kanich, *et al.*, 2008) to be around 1 sale per 12,500,000 emails or %0.00001 – by sending spam in extremely large numbers.

There is without doubt a great need to fight spam to increase the usability of emails and to prevent the damaging effects of spam on businesses. Some (see

Goodman, *et al.*, 2005) have suggested that a solution to the problem would be to shift some of the burden on to the sender by charging monetary or computational resources. Such suggestion, however, is hard to achieve without introducing a major change in the way email is handled and would subsequently require the compliance of all email server administrators on the Internet which is obviously hard – if not impossible – to achieve given the size of the Internet. However, filtering spam at the destination is becoming an attractive option to email server administrators. Although the burden on network resources is not affected by filtering, the usability of the email system increases which is of a higher importance – given the currently low cost of bandwidth and computational resources. Content-based filtering that distinguished spam messages from legitimate ones is proving to be an attractive alternative solution. Machine learning – in particular – have proven to be an effective approach to identifying and filtering spam especially when considering the fact that the characteristics of spam and the methods used in spamming is continuously evolving to adapt to the limits and restrictions set by email server administrators (Bratko, *et al.*, 2006).

Various machine learning techniques have been proposed to identify and filter-out spam such as rule-learning (Cohen, 1996), Naïve Bayes (Sahami, *et al.*, 1998), Decision Trees (Carreras & Marquez, 2001), Support Vector Machines (Drucker, *et al.*, 1999), Memory-Based Learning (Androutsopoulos, *et al.*, 2000a), or a combination of these techniques which are currently used by real-world filtering software to identify and filter spam. For instance, SpamAssassin¹ uses a combination of IP address blacklists, Bayesian filtering, online databases and checksum-based spam filtering. Other examples include SpamBayes² and Bogofilter³ which both use a Naïve Bayes approach to filtering spam. The basic concept behind these techniques is in the classification of emails using a trained classifier that can automatically predict that a certain message is spam rather than use manual classification which would – in return – result in increased filtering performance and better usability (Lai & Tsai, 2004). The classifier assigns a class label described by a

¹ See <http://spamassassin.apache.org>

² See <http://spambayes.sourceforge.net>

³ See <http://bogofilter.sourceforge.net>

set of attributes using data analysis and pattern recognition. The difference in the above mentioned approaches lies in the different functional representations used which would in the end result in class labeling.

In general terms, spam filtering is a special case of textual categorization where the categories are spam or ham (legitimate). In other words, we can think of spam filtering as a real-world application of automated text categorization (Zhang, *et al.*, 2004). Text categorization can be defined as “the problem of assigning predefined categories to free text documents” (Yang, 1999, pp. 69). However, Carreras and Marquez (2001) argued that because spam and ham are the direct opposites of each other, the problem of spam identification is not an issue of identifying where a message lies in the two classes but rather in identifying whether a message belongs to a single class which is spam. Identifying a single class would allow us to test the efficiency of various filters using common performance measures such as the precision, the recall rate and others. If we think of text categorization in the context of machine learning, we will find that the underlying idea is very similar to latent semantic analysis (LSA) and probabilistic latent semantic analysis (PLSA) in natural language processing (see Wikipedia, 2008a and Wikipedia, 2008b) in that machines are trying to analyze the relationship between documents based on the terms contained within them. LSA uses the Vector Space Model (Salton & McGill, 1983) to represent text documents through the use of vectors of index terms. Similarly, the Naïve Bayes approach makes use of similar vectors as document identifiers. However, whilst Naïve Bayes is used to categorize documents into categories based on the score they achieve from probabilistic comparisons of vectors, LSA and PLSA builds relationships between documents for use in the semantic understanding of text.

Spam is easy to classify. If we consider the content of spam messages we will notice that they revolve around a typical set of topics. According to messages passing through the Symantec Probe Network (Bowers & Harnett, 2008, pp. 4), Internet spam constitutes the majority of spam sent during the month of November of 2008 with %22 of all spam belonging to this category. This includes spam advertising web-hosting, web-design, and spamware. Closely following Internet

spam is spam related to products at %18 which includes advertisements related to general goods and services. Similarly, financial spam constitutes %18 which mainly advertise services that are related to the finance such as credit reports, loans, properties and penny stocks. Surprisingly, the infamous health related spam comes fourth constituting %16 of all spam. Health related spam includes spam that advertises pharmaceutical products and herbal remedies. Scams which use fraud and misguidance such as the Nigerian letter – also known as the Nigerian 419 scam – comprise around %7 of all sent spam. Closely following scams are adult-based spam and leisure spam at %6. Adult-based spam advertise offensive products and services of an adult nature whilst leisure spam advertises awards and prizes that are often associated with online gambling. Phishing attacks which try to fraudulently retrieve sensitive information from the recipients comprise around %4. Finally, political spam comes last at %3 which includes donation pledges and campaign advertisement.

If we analyze the content of spam messages, having considered that they fall around the above mentioned topics, we will notice that differentiating spam from legitimate emails merely involves the analyses of words or phrases commonly used to promote products and services in these topics. In other words, the syntactic analysis of content using Bayesian or other means of statistical analysis could result in the successful categorization of messages into spam or ham. This is mainly due to the fact that European languages follow a Zipfian distribution as noted initially by Zipf (1949) and later by Shannon (1951). For instance, when considering the English language, some words appear frequently in the corpus of English texts such as “the”, “of”, “to”, and “and”. These are mostly prepositions and definite and indefinite articles. However, most English words appear in scarce frequency. This allows messages to have a certain amount of uniqueness associated with it stemming from the use of a limited set of vocabulary that we can associate with topics rated as spam or non-spam. This uniqueness would give us the capability to relate received messages with a certain topic based on common words or phrases found in the text of these messages. For instance, if we receive an email that contain the words “cheap”, “Viagra”, “buy” and “click” we can infer with a great amount of

confidence that this message is an advertisement for a pharmaceutical product which is almost always advertised by spamming millions of email users. Similarly with phrases, phrases such as “click here”, “buy cheap Viagra” and “no prescription required” can be used to identify that the received message is spam. For us to be able to make these assumptions the classifier requires training with email previously classified into spam and ham to allow us to estimate the probability of a message belonging to a certain category based on its vectors.

An often ignored and very important part of spam filtering is the analyses of email headers. Although the spammer community tries hard to forge these headers by including fake data, RFC 5321 (Klensin, 2008) and the now obsolete RFC 2821 (Klensin, 2001) require mail servers to add the “Received” tag describing where the mail originated, what software was used to handle the email, and to whom the message is addressed to. These tags are added as the message travels from one mail server to the other until it reaches the destination server. No matter how forged these tags are, at least the last parts will be legitimate which would be very useful for statistical analysis. Even if we take into account the fact that there is a large number of botnets or zombie mail servers used and abused by spammers, the traces left by the “Received” tag could be used in training the classifier and assisting it in recognizing that a certain message originating from a particular IP address or domain is almost always spam by including the IP address or the domain in the training corpus with a high probability associated with it. Similarly, other tags such as the “Return-Path”, “From”, “Reply-To”, and to a lesser extent even “To” could be used in the identification of spam. Although many spammers use randomly generated addresses in these tags, the possibility that a spammer would use the same address over and over again is not to be overlooked. For instance, certain spam messages that are considered phishing attacks use a legitimate-looking email address and repeatedly use the same address in subsequent attacks in an attempt to fool the user that the origin of the message is legitimate. Moreover, hijacked email accounts are sometimes used to send spam from legitimate mail servers. The use of these tags would help identify the hijacked account and filter out future messages if the filter was trained to treat mail from that account to be spam. The “Priority” tag is an

important indicator that an email might be considered spam. We have noticed that spammers are often using the “Priority” tag and setting it to “high” to alert users that their messages are important in the hope of gaining better exposure. Although some legitimate emails do use the “Priority” tag, it could be used – depending on the user – as an indicator of the possibility that an email is spam. The “Subject” tag is perhaps the most useful in identifying spam. Firstly, it contains key words that are often used in spam to alert users to their products and services. Additionally, the language used in the subject line is human readable, and as such, it would allow us make use of pre-calculated probabilities assigned to words or phrases taken from the body of emails in the categorization of email into spam.

In a paper published by Microsoft researchers (Sahami, *et al.*, 1998), a Bayesian-based approach to spam filtering was discussed and evaluated. The approach made use of the Bayesian theorem by tokenizing words contained in email, classifying them based on a corpus of words into spam and ham emails as feature variables, and predicting the probability of messages being spam or ham based on the combined probability of feature variables in the received message. The approach basically applied the Bayesian network into a classification task. Additionally, Sahami and others (1998) used both words and phrases in the tokenization stage and testing whether the filter improved as a result of using the combination. They then added domain specific properties and evaluated the effectiveness of the resulting spam filter. Domain specific properties – whether textual or non-textual – are added as feature vectors in a similar methodology used with words and phrases. Properties such as a message originating from a domain name ending with a “.edu” or the ratio of non-alphanumeric characters in emails are used for or against the classification of mail into spam or ham. Sahami and others (1998) used the Space Vector Model (Salton & McGill, 1983) to represent messages as feature vectors whereby every dimension corresponds to a word in the entire corpus of seen messages. Moreover, they employed an analysis based on the Zipfian law in which words or phrases appearing less than three times are regarded as having little discriminating power associated with them. Table 1 shows the results of Sahami and others (1998).

Feature Regime	Junk		Legitimate	
	Precision	Recall	Precision	Recall
Words only	97.1%	94.3%	87.7%	93.4%
Words + Phrases	97.6%	94.3%	87.8%	94.7%
Words + Phrases + Domain-Specific	100.0%	98.3%	96.2%	100.0%

Table 1: Classification results using various feature sets (Sahami, *et al.*, 1998)

Although their method showed some positive results, they indicated a rather unacceptable rate of false positives that hurt the real-world usability of their Bayesian-based spam filtering implementation. The experiment showed a great potential for identifying spam, however, the potential to misclassify legitimate messages into spam was large enough to be overlooked. Additionally, when comparing the words only feature regime with the words and phrases feature regime, we could assert that little correlation existed between the two tokenization approaches subsequently questioning whether the use of phrases in tokenization was worthwhile.

Graham (2002) in his essay “A plan for spam” independently researched the use of the Naïve Bayes to predict the probability of an email being spam or ham and came up with a simpler more refined implementation which showed much more encouraging results. However, the underlying idea remains similar to what Sahami and others (1998) have discussed. Rather than the use of publicly available emails, Graham noted that the algorithm implementation used messages from his own mailbox. This allowed the classifier to be trained in a way tailored to the kind of messages he receives ruling out any message about a topic he never receives such as porn and the like. Graham reported that out of 1000 spam emails, his algorithm missed 5 emails with 0 false-positives. In a later essay Graham (2003) refined his implementation whilst maintaining the same general approach to classifying emails using the Naïve Bayes and subsequently reached a higher potential for positively identifying spam and a very marginal level of false positives. Moreover, Graham (2003) brought attention to some of the differences between his approach and Sahami and others’ (1998) approach. He pointed out that whilst the underlying

application of the Naïve Bayes was similar, various methodological steps were questionable such as ignoring the important header part of emails and using a low number of spam in the training and testing of the implemented algorithm. Robinson (2003) discussed the use of the Naïve Bayes in spam filtering and suggested ways to deal with rarely occurring words in the corpus of emails. He pointed out that rare words often carry an over-exaggerated probability rendering them useless in the categorization of email. Graham dealt with the issue of rare words by ignoring them. Robinson, on the other hand, suggested including a calculation of the degree of belief in the statistical analysis of tokens. Both Robinson and Graham mentioned that phrase-based tokenization of email contents might have a higher potential for identifying spam. Graham (2002) concluded that whilst phrase-based tokenization might be an attractive option to increase the reliability of Bayesian-based spam filters, words showed sufficient precision for his needs.

As we have previously mentioned, the Zipfian distribution of the English language and the resulting uniqueness associated with rare words in the content of text allows us to identify that a certain message belongs to a particular topic and subsequently classify it as spam or ham. Lyon and others (2006) noted that whilst words could be used as a unique identifier of text, bigrams and trigrams carry a substantially more precise uniqueness and as such would allow a classifier to identify text more accurately. The Ferret program (Lyon, *et al.*, 2006) developed to identify plagiarized text offline showed encouraging results pointing out that trigrams allowed the Ferret to identify plagiarized text more accurately due to the higher level of discriminatory potential associated with trigrams. It is worth mentioning that trigram in the context of this report refers to three consecutive tokens (or words in our case) rather the commonly used definition referring to trigrams as three consecutive letters. The results reported by Sahami and others (1998) concurred with the findings of Lyon and others (2006) showing an increase in the accuracy of the Bayesian spam filter when words are combined with phrases. Although the difference between the results observed with only words and the results observed with the combination of words and phrases was marginal, the difference might point to the possibility that a different implementation taking into

account the assumptions of Lyon and others (2006) might yield more noticeable differences. Additionally, Sahami and others (1998) tokenized words and phrases in a single test-run which would not allow us to confidently make assumptions on the effect of phrases on the accuracy of the Naïve Bayes spam filter.

Given what we have mentioned above, we hypothesize that the use of trigrams in the tokenization of email contents would increase the accuracy and reliability of the classifier due to the higher potential for discrimination associated with trigrams and as a result the efficacy of Bayesian-based spam filters would increase. This project aims to compare the efficacy of a Bayesian-based spam filter when single words were used in tokenization and when trigrams were used. The comparison will be done by implementing a Naïve Bayes classifier using two different tokenizers. The first tokenizer uses single words whilst the second tokenizer uses trigrams (three-word phrases). A suitable approach to trigram-based tokenization will be developed. The classifier would be the same in both instances. The implementations will use a public corpus of spam and legitimate emails and will test the two implementations on a live set of emails from the same corpora of emails. The results will be analyzed and any findings will be discussed thereto.

The remainder of this report is structured as follows. Chapter 2 discusses previous work that compared the efficacy of the Naïve Bayes classifier and other research related to this project. Chapter 3 describes the tokenization of emails and the dataset used as well as any preprocessing steps taken to prepare the data. Chapter 4 describes the Naïve Bayes classifier as well as the training and classification of emails. Chapter 5 discusses the experimental setup of the project and the evaluation methodology used. Chapter 6 reports the analyses of results. Finally, chapter 7 discusses the findings and concludes the report.

2. Related Work

As mentioned before, Graham (2002) and Robinson (2003) pointed out that the use of phrase-based tokenization might yield more effective filters. However, little formal research attempted to test out whether phrase-based tokenization could improve a Naïve Bayes classifier. Sahami and others (1998) compared tokenization

with words only and with a combination of words and phrases. Their approach was not directed towards testing the effect of phrased-based tokenization and as such little could be concluded from the result of their work. However, the slight improvement in the efficacy of the filter might point us to the underlying positive influence of phrase-based tokenization.

Lai and Tsai (2004) compared the performance of various machine learning methods for spam email categorization. Their results showed strong evidence that header information improved the filtering performance of all the tested classifiers whether they were based on a Naïve Bayes methodology or other methodologies such as Term Frequency-Inverse Document Frequency and Support Vector Machine. Additionally, stemming had no effect on the performance of spam filters. Stemming refers to algorithms that reduce words to their base or root and is a concept generally used in search engines and indexing. Moreover, when they tested the filtering ability based only on the subject or the body of emails, poor performance was observed. As such, the results from the study strongly suggested the use of all information contained in the email from the header to and including the body.

A study on web search (Johnson, *et al.*, 2006), investigated the effects of bigrams and trigrams. Although the problem domain of the study is not relevant to our project, the results from the study showed good indicators that bigrams and trigrams are more suitable identifiers of text, and as such, they qualify as high quality discriminators in text categorization. Johnson and others (2006) referred to previous research in natural language processing in favor of the use of phrases due to the higher meaning conveyed in them as opposed to single-words. They concluded that bigram and trigram based search queries improved the efficacy of searching. Moreover, another study by Lyon and other (2006) examined the use of trigrams in the identification of plagiarized text. Based on the assumption that the English language follows a Zipfian distribution (1949), Lyon and others (2006) used trigrams for tokenization. By analyzing corpora of text from TV news, Federalist Papers, and the Wall Street Journal, they showed that trigrams provide a high measure of uniqueness suitable for text identification. Additionally, they noted that this observed phenomenon is not unique to European languages and that trigrams

made from Chinese terms share the same level of uniqueness observed in the English language. Lyon and others (2006) tested the identification of plagiarized programming code using text identifiers based on trigram tokenization. Rather than using English words, they used programming code syntactic structures as the bases and reached encouraging results with zero false-positives.

3. Tokenization

Almost every real-world application of the Naïve Bayes uses word-based tokenization. Tokenization is the activity of constructing a vector representing the content of email. The constructed tokens are considered the vector attributes of messages and their frequency ratios are used in the calculation of probabilities during classification. Deriving tokens from messages is a multi-stage preprocessing task in which text is filtered according to specified criteria and grouped into a dataset accessible to the classification function. Two schemes of tokenization were implemented. A word-distribution based tokenization in which every word is considered a token and a trigram based tokenization in which every 3-word phrase is considered a token.

The two implementations preprocessed the email header in the same way. As we mentioned earlier, the header part of emails includes some strong incriminating evidence that could not be overlooked without affecting the accuracy and reliability of spam filters. Particularly in relation to the *Received* tag, header information contains real identifying and unique information despite the attempts by spammers to forge these headers. For instance, in our preliminary analysis of spam emails, a substantial amount of emails contained a *Received* tag pointing to *localhost* as the source MTA (Mail Transport Agent). Tokenizing the *Received* tag would strongly support the conclusion that an email originating from the *localhost* MTA is spam.

Five header information tags were taken into consideration during tokenization (See Appendix A). They were *Received*, *Message-id*, *From*, *Subject*, *Content-type*, and *Content-charset*. The *Received* tag includes information pertaining to the path the message took until it reached the destination MTA. Interesting information that can be extracted from the *Received* tag is the IP address and

hostname. Some MTAs include both the IP address and the hostname when constructing the *Received* tag. However, there were instances where the IP address of the source MTA does not resolve to a hostname. In these instances, either the hostname was tokenized or the IP address depending on the available information in the tag. Although RFC 2821 require MTAs to include the *Received* tag in relayed messages, malformed headers that do not adhere to the specifications set by RFC 2821 are tolerated by various MTAs. Malformed headers are strong indicators that an email is spam and as such missing *Received* tags are given a special token to indicate that the message did not have a *Received* tag in its header information. In typical circumstances; however, the IP address and/or the hostname was tokenized and given a special prefix “recv-ip:” and “recv-host:” to differentiate these tokens from tokens found in the body of emails.

Similarly, the *Message-id* tag could be used to identify bad or good sources of emails. Most MTAs include a unique generated message id to every message sent from it. The *Message-id* tag is split into two parts. The first part is a generated string unique to the sent message. This will almost always be unique and have a frequency ratio of 1/n and as such would not be useful to us. However, the second part of the *Message-id* includes a string unique to the MTA that sent the message. Typically, the second part of the *Message-id* is the hostname of the MTA, however, this is not always true. What is important for classification is the possibility of including information unique to the MTA that could help us predict whether a message is spam or ham based on the source MTA. The *Message-id* was tokenized and given the prefix “mesg-id:”.

Moreover, the *From* tag was tokenized and given the prefix “from-real:” for the real name of the sender and “from-addr:” for the email address. The *From* tag – and any address in the header part of emails – might include the name of the sender as well as the email address and as such both were taken into account during tokenization. Additionally, some malformed messages might not include the *From* tag and this is almost always an indicator that the message is spam. A special tag “from-addr:none” was given to indicate that the message did not have a *From* tag in the header part of messages. Although the possibility that an email originating from

the same sender is dismal at best, we chose not to ignore such indicator that a message might be from a spammer. We have observed from experience that some spammers use legitimate MTAs with real accounts for sending spam messages. Given a sufficient amount of spam from the same sender in the training corpus, we could predict that this account is used by spammers and classify any future messages from this account as spam. Similarly, a legitimate sender who frequently sends emails to the user could be flagged as an indicator that the message is ham. This concept is very much similar to rule-based blacklisting and whitelisting, however, employing automated statistical methods rather than human-generated rules.

Because of the human-readable nature of text found in the *Subject* tag. Tokenization of this tag was done during the body tokenization stage based on whether we implement a word-based or a trigram-based tokenization scheme. The prefix “subj:” was given to tokens taken from the *Subject* tag. *Content-type* and *Content-charset* are useful indicators especially when legitimate messages are almost always non-HTML formatted. It is a known fact that spammers often use HTML formatted messages, image attachments, and non-ASCII characters to fool text-based spam filters. Including information pertaining to the content-type and the charset used allow us to include some possibly useful indicators into the training corpus and in message classification.

Other header tags such as *To*, *Delivered-To*, *CC*, *BCC*, *Importance*, and *Sender* were ignored. This is mostly due to the fact that they carry little to no statistical significance as noted by the SpamBayes contributors (Meyer & Whateley, 2004). The *Sender* is almost always identical to the *From* tag and the *To* and *Delivered-To* would carry a frequency ratio near 1. Similarly, the *Importance* tag is almost always forged by spammers to various levels to fool current spam filters and as such they lost the once important indication capability.

3.1 Word-based Tokenization

Tokenization of the email body based on word distribution is simply done by collecting words that are separated by white-space – whether the white-space is a space, a tab, a carriage return, a new-line character, a combination of white-space

characters, or multiple white-space characters (See Appendix B). Prior to splitting the body of a message, several preprocessing steps are taken to filter out the irrelevant content which mostly consists of non-alphanumeric characters. Almost all non-alphanumeric characters are replaced by a space character. This in effect removes any non-alphanumeric characters from the body of the email. Some non-alphanumeric characters are kept such as “\$”, “!”, “-”, and “%”. These are kept in the body of emails because of their high association with words and numbers used in spam email. For instance, we have observed that a lot of spam mention the pricing of their products and in an email inbox that receives technical/non-commercial messages, a price might indicate that the received message might not be appropriate for the user’s inbox. Similarly with the exclamation mark “!”, spam often over-use exclamation marks in an attempt to lure the reader into reading their adverts. The percentage sign “%” is also used by spammers to indicate savings offered by them compared to local shops and therefore represent a good indicator that a message containing a percentage sign “%” could be spam. The hyphen “-” is a word connector. Words that are separated by a hyphen are often considered a single word and carry a different meaning from the same words separated, and as such, it would make more sense to keep the hyphen rather than filtering it out.

The implemented tokenizer deals with HTML as well as plain-text content in much the same way. This means that HTML tags are stripped and URLs are torn apart into words. This might limit our ability to use the syntactic information contained within HTML encoded messages. However, we have observed that several tags such as the *IMG* tag as well as HTML attributes such as *HREF* carry a strong indicator that a message is spam as a substantial amount of spam includes external and internal images and links in the body of the email.

Some emails contain multiple parts (called multipart-messages). They were dealt with as a single block of text and are tokenized as such. Multipart-messages that are base64 encoded are not decoded automatically. This means that tokenization of base64 messages is done using unmodified encoded literals. Although little meaning could be derived from an encoded message, the uniqueness associated with the encoded literals remains unaffected and as such we chose to use them for

analysis. Graham (2002) noted that words that are less than 3 characters in length carry little to no statistical significance. As such, our implementation ignored words that are less than 3 characters in length. Moreover, words that are longer than 12 characters were similarly of little statistical significance and were ignored. Robinson (2003) and the SpamBayes project (Meyer & Whateley, 2004) concurred with Graham's conclusion. Additionally, words are automatically converted to lower-case prior to tokenization. This allows words that are capitalized to add to the significance of non-capitalized words rather than treating them as separate tokens with their own frequency ratios. This makes sense because capitalization is merely a grammatical structure that carries no syntactic information.

3.2 Trigram-based Tokenization

In much the same way, trigram-based tokenization follows the same preprocessing steps taken by the word-based tokenizer (See Appendix B). Because we are merely comparing the improvement (or lack thereof) of the spam filter's efficacy when we use trigrams rather than words as tokens, most of the preprocessing steps were replicated in the trigram-based tokenization. This includes separating words based on white-space, stripping non-alphanumeric characters, including the "\$", "!", "-", and "%" characters, stripping HTML encoded messages, dealing with multipart messages, and converting words to lower-case. However, rather than considering every space-separated word as a token, every three-word phrase was considered a token. We dropped the lower limit of word length (3 characters). This is mainly due to the fact that whilst in word-based tokenization words less than 3 characters in length were not significant and carried no significant meaning, phrases that contain words less than 3 characters in length contribute to the meaning carried by phrases. For instance, trigrams such as "buy a watch" and "buy or watch" could appear in spam and ham messages. The former could be attributed to spam messages trying to sell counterfeit watches whereas the latter might be part of a friendly comment by a legitimate user. Stripping the "a" and the "or" words from the above phrase would blur the phrase into something that could be attributed to both categories and as such the phrases would lose the informative qualities associated with them. Having said

that, words that are longer than 12 characters are often either very rare words which would carry no significance anyway or non-human readable textual content such as the remainder of long URLs and such. Therefore, the higher limit was kept in trigram-based tokenization. Additionally, some words less than 4 characters in length are purely non-alphanumeric as a result of not filtering all non-alphanumeric characters. These words often appear after tokenizing HTML formatted messages. Words such as ‘!--’, and ‘%%’ carry no meaning and were filtered out during tokenization. Similarly, multiple occurrences of white-space were stripped out to a single occurrence to allow words to be easily separated programmatically.

No measures to limit the effect of trigram overlapping were taken. A sentence such as “Click here to buy the watch now!” would create 5 tokens “click here to”, “here to buy”, “to buy the”, “buy the watch”, and “the watch now!”. Additionally, phrases at the end of the body which were less than 3 words were not taken into consideration. This is a limitation we are willing to overlook because we have not come out with a way to deal with two-word and single-word phrases. Such phrases – even if taken into account – would not amount enough statistical significance to be considered.

3.3 The Datasets

Emails used in testing were collected from the public corpus of SpamAssassin¹. The public corpus is pre-categorized into three categories *easy_ham*, *hard_ham*, and *spam*. We only made use of the *easy_ham* and the *spam* corpora. After examining the emails contained in the *hard_ham*, we found that most of the messages originated from newsletters and product advertisement related emails that had little to no relevancy to the mailing list of SpamAssassin which is the largest contributor to the corpus. To simulate a real-world email user, we chose disregarding emails that are categorically different. Although one might argue that a real email inbox would typically include a variety of emails from different categories, it is not uncommon to find email inboxes mostly belonging to a handful of categories based on the interests and profession of the email user. A software developer – for instance – would

¹ Available at: <http://spamassassin.apache.org/publiccorpus/>

typically receive newsletters relating to recent developments in the software industry, questions and support queries from customers and colleagues, and the occasional social email. Moreover, we found various emails that we judged as obvious spam messages included in the *hard_ham* corpus. This made us hesitant to incorporate the *hard_ham* corpus with the *easy_ham* corpus into a single ham corpus. The total amount of emails used in the training corpus and in testing the classification is summarized in the following table:

Corpus	Number of messages	Distribution
Spam	1896	%32.71
Ham	3900	%67.29
Total	5796	%100

Table 2: The amount of emails used in training and testing the classification

The collected emails were sent between the years 2002 and 2003. Although categorically, emails – and spam in particular – has not changed much since 2003, new ways to circumvent word-based spam filters have been identified in recent spam messages and as such caution should be taken when applying the findings of this study in the real world.

The classifier used and compared the accuracy depending on the amount of messages processed by the trainer. During analysis, an equal ratio of spam to ham was used in training both the word-based classifier and the trigram-based classifier. This was intentionally done to limit the possibility of a statistical α error (Type-I error). An uneven distribution of spam to ham across tests might yield higher accuracy in classifying one class of email against the other resulting in what we may call a false alarm. The following table describes the distribution of messages used in the training corpus:

Corpus	Number of messages	Distribution
Spam	948	%32.71
Ham	1950	%67.29
Total	2898	%100

Table 3: The distribution of spam to ham in the training corpus during result analysis

The emails from both corpora (spam and ham) were randomly selected and sorted into two folders based on their class and were removed from SpamAssassin's pre-categorized corpora of spam and ham so as not to be included in tests relating to the efficacy of the classification task (See Appendix C).

4. The Naïve Bayes classifier

Classification based on a Bayesian network is considered one of the most effective due to the high level of predictive performance observed. The classifier is able to categorize data using pre-fed training data. The probability that a data item with an attribute A is part of the class label C is computed by applying the Bayes rule whereby every instance of A_1, \dots, A_n that is part of C is given a probability and a combined probability is computed resulting in the success or the failure of the categorization. To use the Bayes rule, one must assume that every instance of A is probabilistically independent (i.e. there is no correlation between the attributes). This is obviously not true in many instances. However, the results given by the computations in many applications that falsely assume independence are surprising (Friedman, *et. al.*, 1997).

As mentioned earlier, a Bayesian-based classifier is a Bayesian network applied to a categorization task whereby the categories are either spam or ham. Each message is tokenized and represented by a vector $\vec{x} = \langle x_1, x_2, x_3, \dots, x_n \rangle$ where $X_1 \dots X_n$ are the values of attributes. Researchers in Naïve Bayes classification as in (Sahami, *et. al.*, 1998, Androutsopoulos, *et. al.*, 2000a, and Androutsopoulos, *et. al.*, 2000b), computed a mutual information (MI) of each attribute and selected attributes with the highest MI for use in predicting the message category. For each candidate attribute X with category-denoting variable C , MI is calculated as follows:

$$MI(X : C) = \sum_{x \in \{0,1\}, c \in \{spam, legit\}} P(X = x, C = c) \cdot \log \frac{P(X = x, C = c)}{P(X = x) \cdot P(C = c)}$$

The probabilities are calculated as frequency ratios derived from the training corpus. However, we did not use MI to derive the most significant attributes. Rather than calculating MI , all tokens were taken into account whilst we calculated the

probabilities giving us more representative results but increasing the amount of calculations needed to classify messages. As we will elaborate shortly, frequency ratios were used in calculating the probabilities of each attribute and a combined probability is subsequently calculated using the Bayes' theorem. Graham (2002) and Robinson (2003) used frequency ratios for each attribute (token) in the training corpus to estimate whether a certain message is spam or ham. The Bayesian theorem states that for a given vector $\vec{x} = \langle x_1, x_2, x_3, \dots, x_n \rangle$ of a document, the probability that it belongs to category c is:

$$P(C = c | \vec{X} = \vec{x}) = \frac{P(C = c) \cdot P(\vec{X} = \vec{x} | C = c)}{\sum_{k \in \{\text{spam}, \text{legitimate}\}} P(C = k) \cdot P(\vec{X} = \vec{x} | C = k)}$$

Applying the Bayesian theorem on spam categorization, Graham (2002) and Robinson (2003) used the following formula:

$$p(w) = \frac{b(w)}{b(w) + g(w)}$$

Where $b(w)$ is the number of spam emails containing the word w divided by the total number of spam emails and $g(w)$ is the number of ham emails containing the word w divided by the total number of ham emails. $p(w)$ is the probability that a randomly chosen email containing the word w will be spam. Switching $b(w)$ and $g(w)$ in the above formula gives us the probability that a randomly chosen email containing the word w is ham. The above simplification of the Bayesian theorem considers all messages in the training corpus as part of a long stream of text and thereby every token is part of the training corpus's vector. This in effect requires the user to feed the application implementing the formula an equal and sufficient amount of spam and ham emails prior to calculating the probabilities. However, in practicality a user might have unbalanced amounts of spam and ham in his/her inbox which would subsequently affect the probability calculated. The above formula does not take this into account. Robinson (2003) noted that dealing with the above situation requires that we calculate the degree of belief $f(w)$. When we see a word in a certain message that belongs to spam or ham, the possibility that the same word would appear in the same category might not be absolutely %100. The Bayesian approach allows us to

combine background information with our data resulting in a much accurate calculation. The degree of belief $f(w)$ is calculated as follows:

$$f(w) = \frac{(s \times x) + (n \times p(w))}{s + n}$$

Where s is the strength we give to our background information, x is our assumed probability based on our general background information, and n is the number of emails we received that contains the word w . Robinson (2003) derived the above formula based on the assumption that the classification of an email into spam or ham based on the fact that it contains the word w is a binomial random variable with a beta distribution prior. Based on prior experiments done by the SpamBayes contributors (Meyer & Whateley, 2004), the optimal assumed probability x was observed at 0.5. No improvement was observed when changing the value from 0.5. Additionally, for word-based tokenization the optimal strength s was observed at 0.45. Rather than using $p(w)$ in calculating the probabilities, we used $f(w)$ which as noted by Robinson (2003) would result in more reliable classification. For trigram-based tokenization, the assumed probability x remained at 0.5. However, through trial and error we found that doubling the strength s variable to 0.9 provided the trigram-based classifier with more accuracy than when leaving the variable unchanged at 0.45. To maintain consistency of results across tests, the strength s was left unchanged for trigram-based classifications.

We used Fisher's optimality theorem (Little & Folks, 1971) as noted in (Robinson, 2003) to combine the probabilities of every word w . We assumed that the result of the combination is a chi-square distribution with $2n$ degrees of freedom and used an inverse chi-square function to calculate the combined probability. The combined probability of ham was calculated as follows:

$$H = C^{-1}(-2 \ln \prod_w f(w), 2n)$$

Where C^{-1} is the inverse chi-square function used to derive the probability from a chi-square-distributed random variable (See Appendix D). Because the above calculation is very sensitive to probabilities near 0 which are indicators that the word w is probably part of a ham email, we could not use it to calculate the combined probability of spam. Robinson (2003) noted that to apply the above calculation to the

combined probability of spam, we reverse the probabilities by subtracting them from 1 giving us the inverse and allowing us to do further calculations as we will note shortly. In effect, the combined probability of spam S is calculated as follows:

$$S = C^{-1}(-2 \ln \prod_w (1 - f(w)), 2n)$$

Where C^{-1} is the same inverse chi-square function mentioned earlier. After calculating both S and H we can produce I which indicates whether the evidence is in-favor of concluding that an email is spam or ham. When I is near 1, the evidence provided indicates that the message is spam and when it is near 0 the evidence indicates that the message is ham. I is calculated using the following formula:

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

Where I is the indicator whether the message is ham or spam, H is the combined probability of ham, and S is the combined probability of spam¹.

4.1 Training and classification

Training the Naïve Bayes classifier is the one of the most fundamental and crucial tasks in implementing a classifier. Training comes only after the application is able to fully tokenize and deal with the textual contents of emails. Rather than storing pre-calculated probabilities to each token stored in the training corpus, only the number of occurrences of each token is stored. Probability estimation is postponed until classification (See Appendix D & Appendix E). This allows the training corpus of tokens to be incremented gradually whenever a new email is added as a training item as well as allowing us to examine the effects of adding new emails to the training corpus on the accuracy of the classifier. Two separate hashtables were created. Each hashtable contains the tokens as keys and the number of occurrences as values. Additionally, the number of messages pre-categorized and added to the training corpus is recorded and incremented according to whether new emails were added. This allows us to apply the formulas mentioned above during classification

¹ For a detailed description of how these formulas were derived see (Robinson, 2003) available at: <http://www.linuxjournal.com/article/6467>

using fresh data without jeopardizing the accuracy of the calculations taking into account any new additions to the training corpus.

When a new message is pending classification, the textual contents are tokenized using the tokenizer – creating a hashtable of tokens and number of occurrences. Every token is taken into account during probability estimation. Tokens are given a frequency of one regardless of the number of occurrences in a single email. This is deliberately done to correctly calculate the probabilities according to the above mentioned formulas. Every token is searched for in the training corpus. If the token was found in the training corpus, $f(w)$ is calculated whether it belongs to the spam, ham, or both hashtables. However, when the token does not exist, the presumed probability with the value 0.5 (neutral) is given to the token. When all tokens have been considered, all calculations are temporarily stored and the I value is calculated using the formulas described above. Through trial and error we found that $\pm 1 \sigma$ from the presumed probability 0.5 is the optimal cutoff point whereby when a value less than -1σ (%31.7) was returned, the email was identified as ham and when a value more than $+1 \sigma$ (%68.2) was returned, the email was identified as spam. For any message that returns a value that lies between $\pm 1 \sigma$, an “unsure” classification was returned. Graham (2002) classifier used two categories spam and ham. However, as noted by Robinson (2003) this leads to falsely assuming that an email that only marginally falls below the cutoff point to be regarded as ham when in reality a more correct term describing the email would be “I do not have enough evidence to classify this email” or “unsure” for short. This allows us to distinguish failure of classification from success in classifying ham emails. Additionally, this means that rather than setting a single probability threshold for classifying spam, two thresholds needs to be set – an upper threshold for classifying spam and a lower threshold for classifying ham. Extra care needs to be taken when filtering spam due to the potential disastrous effects of falsely identifying a legitimate email as spam. When the I value drops below %68,2 (or $+1 \sigma$), non-action should be taken by the filter irrespective. Although such approach might yield higher occurrences of false negatives in the users’ inboxes, it is indeed – as noted by many – better than the detrimental effect of producing a false positive result. The usability of spam filters

are greatly affected by false positive results as opposed to the negligible effects of a false negative result. Marking a legitimate email as spam might – depending on the user’s setup – hide it from the inbox resulting in the loss of said email. It should be noted, however, that the cutoff point should be decreased if a large amount of false negative results were observed due to the known inverse relationship between false negatives and false positives. Similarly, when a false-positive was identified, the suggested course of action is to increase the cutoff point.

Slight code optimization was applied to the method used to calculate $f(w)$. Although there is much room for optimization, our main concern was to accurately calculate $f(w)$ and as such optimizations were largely overlooked. However, to speed up the classification, the $f(w)$ values calculated during classification were cached which significantly improved the performance of the implemented classifier. When a new email is added to the training corpus, the cache is reset because any new addition to the training corpus invalidates previous $f(w)$ calculations. Additionally, the script used to test the two classifiers and retrieve the results used concurrent multiple processes to speed up calculations and to take full advantage of multiple processors widely equipped in modern computers.

5. Evaluation Methodology

As in Sahami *et. al.* (1998), two performance measures were calculated: the recall rate and the precision rate. The recall and precision rates were calculated using the following formulas:

$$recall = \frac{n_{spam \rightarrow spam}}{n_{spam \rightarrow spam} + n_{spam \rightarrow ham}} \quad precision = \frac{n_{spam \rightarrow spam}}{n_{spam \rightarrow spam} + n_{ham \rightarrow spam}}$$

Whereby $n_{spam \rightarrow spam}$ is the total number of messages accurately identified as spam, $n_{spam \rightarrow ham}$ is the total number of spam messages identified as ham (false-negative), and $n_{ham \rightarrow spam}$ is the total number of legitimate messages marked as spam (false-positive). Classification in previous studies used two classes spam and ham. When an email fails to accumulate enough probability to supersede a set threshold, it is automatically rejected and classified as ham. According to the filter devised by Robinson (2003) and later used by the SpamBayes team, an “Unsure” category was

added to distinguish failure to classify an email from classifying an email as ham. However, to adjust the above formulas to take into account the third category “unsure”, the number of “unsure” classifications was added to the $n_{spam \rightarrow ham}$ value giving us a recall rate comparable to previous studies using the same performance measure. The recall and precision rates were calculated in both the word-based spam filter test run and the trigram-based spam filter test run. The same collection of emails was used in training the classifier in both instances. Moreover, the collection of emails used in testing was the same. A special test script (See Appendix F & Appendix G) was written to run the two tests and incremental results were collected. The recall and precision rates were calculated every incremental step during which emails were added to the training corpus.

In addition to the recall and precision rates, a total cost ratio (TCR) was calculated. As noted by Androutsopoulos *et. al.* (2000, pp. 6), the performance of classification tasks is measured through the comparison of the accuracy (Acc) and the error (Err) rates. These rates are calculated using the following formulas:

$$Acc = \frac{n_{ham \rightarrow ham} + n_{spam \rightarrow spam}}{N_{ham} + N_{spam}} \quad Err = \frac{n_{ham \rightarrow spam} + n_{spam \rightarrow ham}}{N_{ham} + N_{spam}}$$

Where $n_{ham \rightarrow ham}$ and $n_{spam \rightarrow spam}$ are the number of messages accurately identified as ham and spam respectively. N_{ham} and N_{spam} are the total number of ham and spam messages respectively. However, because we are dealing with emails in which the $ham \rightarrow spam$ error is considered more costly than the $spam \rightarrow ham$ error, a λ variable was introduced representing the penalty multiplier to false-positive classification. A weighted accuracy (WAcc) and weighted error (WErr) taking into account λ are calculated as follows:

$$WAcc = \frac{\lambda \cdot n_{ham \rightarrow ham} + n_{spam \rightarrow spam}}{\lambda \cdot N_{ham} + N_{spam}} \quad WErr = \frac{\lambda \cdot n_{ham \rightarrow spam} + n_{spam \rightarrow ham}}{\lambda \cdot N_{ham} + N_{spam}}$$

Androutsopoulos *et. al.* (2000) compared the rates observed with spam filtering against the baseline rate in which no spam filter was used. The combined formula during which TCR was derived from is the following:

$$TCR = \frac{WErr^b}{WErr} = \frac{N_{spam}}{\lambda \cdot n_{ham \rightarrow spam} + n_{spam \rightarrow ham}}$$

Using the values reported by TCR, we can assert whether the spam filter performed better than the baseline (when TCR is less than 1). The higher the TCR value is, the better the spam filter performed under the experiment's conditions.

Just as we did for calculating the recall and precision rates, the TCR value for the word-based and the trigram based classifiers were incrementally measured using a test script (See Appendix F & Appendix G) that added 100 spam emails and 210 ham emails to the training corpus in each incremental step – maintaining the spam to ham ratio in every incremental step. Pre-categorized emails were fed to the classifier and the returned classification was stored for later analysis. Moreover, λ was considered using three different values that are equal to what Androutsopoulos *et. al.* (2000) used which were $\lambda=1$, $\lambda=9$, and $\lambda=999$. At $\lambda=999$, the penalty for falsely classifying a ham email as spam is 999 times greater than classifying a spam email as ham. Just as we did with the adjusted formula for the recall rate, the number of “unsure” classifications was added to the $n_{spam \rightarrow ham}$ variable to allow us to produce results comparable to previous studies in spam filter analysis. The main reason why three different performance measures were used in assessing the efficacy of word-based and trigram-based spam filters was to allow the reader to easily compare the results reported in this study with results from previous research in the area of spam filtering. Additionally, this allows both readers who are familiar with information retrieval tasks (recall and precision) as well as information classification tasks (accuracy and TCR) to easily comprehend the reported results.

6. Results and Analysis

Data retrieved from the test of the word-based spam filter showed improved recall rates whenever new emails were added to the training corpus. Rapid increases when the training corpus contained less than 961 emails (310 spam and 651 ham) were observed (From Recall = 0.209 until Recall = 0.71). The rate of increase in the recall rate slowed down when emails added to the training corpus exceeded 960 emails (310 spam and 651 ham) at (Recall = 0.71). A noticeable drop in the recall rate was observed between 713 and 868 emails in the training corpus (230 spam and 483 ham until 280 spam and 588 ham) reaching the lowest point of the drop at (Recall =

0.616). At 2170 emails (700 spam and 1470 ham) in the training corpus, the recall rate stabilized and subsequent additions to the training corpus showed insignificant improvements with an average recall rate of (Recall = 0.862). The maximum recall rate achieved was (Recall = 0.888) at 2790 emails in the training corpus (900 spam and 1890 ham) with slightly deteriorating rate when the remaining spam and ham in the collection of training emails were added to the training corpus (Recall = 0.887, 948 spam and 1950 ham). The precision rate for the word-based spam filter remained constant throughout the experiment with insignificant levels of variability as and when emails were added to the training corpus (Average precision = 0.996, maximum precision = 1.0 at 10 spam and 21 ham). The following figure shows the recall and precision rates for the word-based spam filter based on the number of emails in the training corpus:

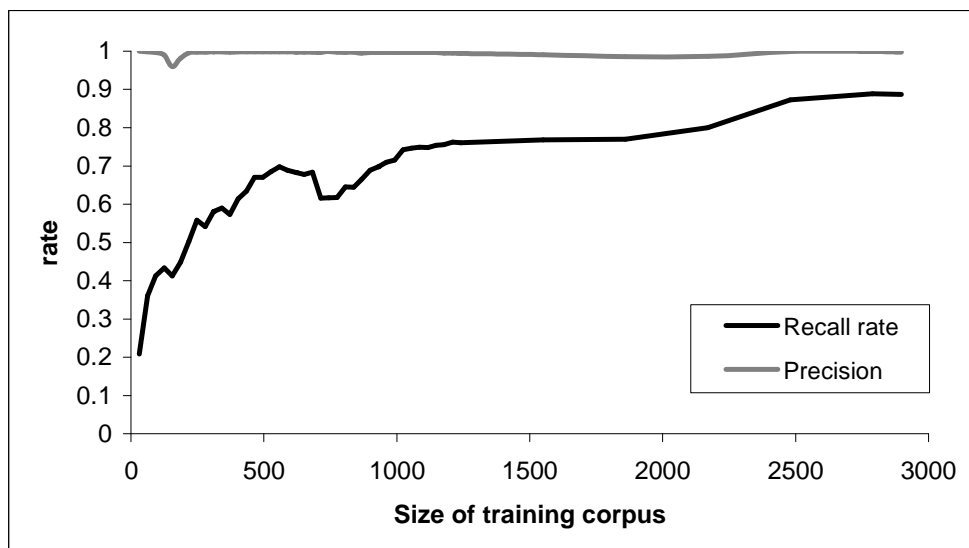


Figure 1: The recall rate and the precision rate of the word-based spam filter.

The TCR values for the word-based spam filter showed gradual increments when emails added to the training corpus were less than 960 emails (310 spam and 651 ham). Similar to the recall rate, a noticeable drop in the TCR value was observed between 713 and 868 emails in the training corpus (230 spam and 483 ham until 280 spam and 588 ham) reaching the lowest point of the drop at (TCR = 1.789). The TCR value stabilized at (TCR = 3.402) showing insignificant improvements between 1023 emails and 1860 emails (330 spam and 693 ham to 600 spam and

1300 ham). Rapid increases in the TCR value were observed when emails added to the training corpus succeeded 2000 emails (600 spam and 1260 ham) peaking at 2790 emails (900 spam and 1890 ham) with a TCR value of (TCR = 8.316) and dropping slightly to (TCR = 8.103) when the remaining spam and ham emails in the collection of training emails were added to the training corpus at 2898 emails (948 spam and 1950 ham). When adjusting the TCR value with ($\lambda=9$) and ($\lambda=999$), similar patterns of improvement were observed. However, the TCR value remained below the baseline (TCR = 1.0) when the TCR value was adjusted with ($\lambda=999$). Whilst at ($\lambda=9$), the rate of improvement showed similar pattern to the TCR value at ($\lambda=1$) peaking at (TCR = 7.77) when 2790 emails were added to the training corpus (900 spam and 1890 ham). When the size of the training corpus was less than 960 emails (310 spam and 651 ham) TCR values were almost equal to the TCR values observed at ($\lambda=1$). A slight decrease on further additions to the training corpus was observed between 1209 emails (390 spam and 819 ham) and 2170 emails (700 spam and 1470 ham). At ($\lambda=999$) the TCR values were insignificant remaining below 1.0 with a maximum of (TCR = 0.853). A rapid increase in the TCR value was observed when emails succeeded 2000 emails (600 spam and 1260 ham) similar to what was observed at ($\lambda=1$)¹. The following figure shows the TCR values at ($\lambda=1$), ($\lambda=9$), and ($\lambda=999$):

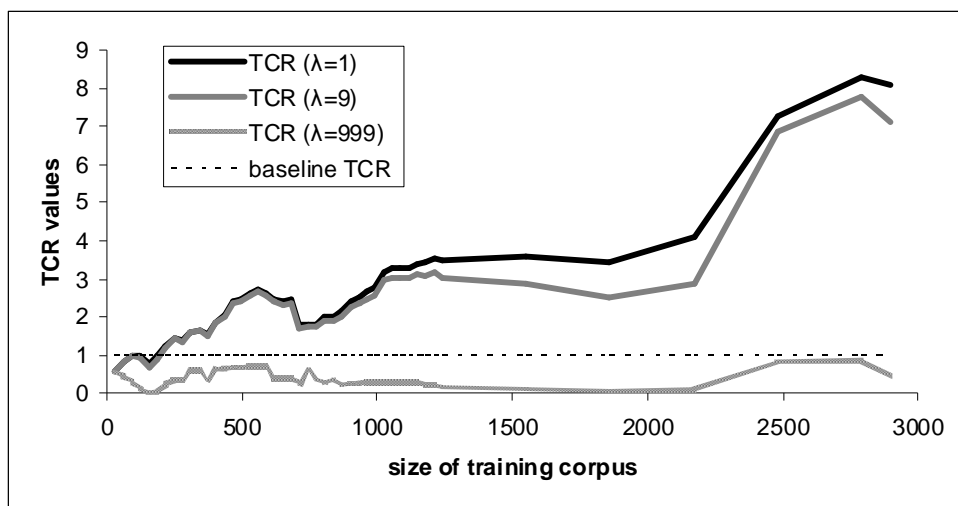


Figure 2: TCR values for the word-based spam filter.

¹ A table containing the raw results from the test on the word-based spam filter is available in Appendix I.

With respect to the test that ran on the trigram-based spam filter, the recall rate remained below what was observed in the test on the word-based spam filter with a maximum recall rate of (Recall = 0.655) at the maximum amount of emails in the training corpus at 2989 emails (948 spam and 1950 ham). The recall rate did not drop at the last increment in the training corpus as what was observed in the test on the word-based spam filter. A different pattern of improvement was observed in the test on the trigram-based spam filter. The trigram-based spam filter showed smoother and gradual increases in the recall rate whenever new emails were added to the training corpus. A slight increase in the rate of improvement was observed at 434 emails in the training corpus (140 spam and 294 ham) when (Recall = 0.28). The trigram-based spam filter showed extreme levels of precision dropping bellow a (Precision = 1.0) many times less than the word-based spam filter. The following figure shows the recall and precision rates for the trigram-based spam filter:

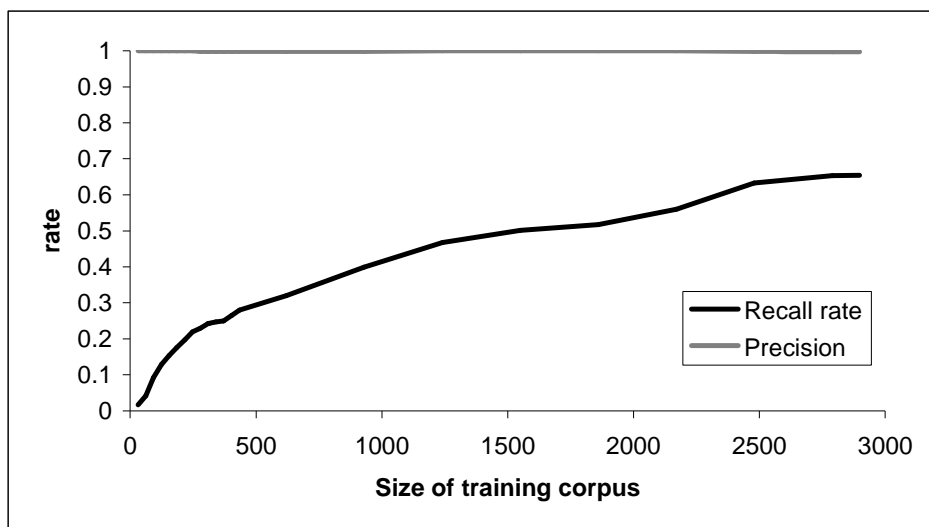


Figure 3: Recall and precision rates for the trigram-based spam filter.

Increments to the training corpus in the trigram-based spam filter test resulted in smooth and gradual improvements to the TCR values when the size was less than 2170 emails (700 spam and 1470 ham) at ($\lambda=1$). Subsequent additions to the training corpus resulted in a rapid increase from (TCR = 1.61) to (TCR = 2.246) peaking at (TCR = 2.312) at the last addition to the training corpus at 2989 emails (948 spam and 1950 ham). The TCR value when ($\lambda=1$) crossed the baseline TCR at

1240 emails (400 spam and 840 ham). Adjusting TCR with ($\lambda=9$) resulted in an almost identical pattern of improvement to what was observed when ($\lambda=1$). A slight deviation from the ($\lambda=1$) pattern was observed when more than 2480 emails (800 spam and 1680 ham) were added to the training corpus. TCR values crossed the baseline TCR at the same point in which the non-adjusted TCR values crossed it at 1240 emails (400 spam and 840 ham). Almost identical TCR values to ($\lambda=1$) and ($\lambda=9$) were observed when less than 248 emails (80 spam and 168 ham) were added to the training corpus. Subsequent additions resulted in a drop to (TCR = 0.39) varying only slightly until the amount of emails added to the training corpus reached 930 emails (300 spam and 630 ham). The TCR values at ($\lambda=999$) crossed the baseline TCR at the same point in which the TCR values at ($\lambda=1$) and ($\lambda=9$) crossed it at 1240 emails (400 spam and 840 ham). Further additions to the training corpus resulted in TCR values similar to the TCR values at ($\lambda=1$) and ($\lambda=9$) up to a training corpus with 2170 emails (700 spam and 1470 ham) with (TCR = 1.61). Even further additions resulted in the deterioration of the TCR values reaching the lowest point at 2790 emails (900 spam and 1890 ham) with (TCR = 0.394). The last increment to the training corpus with 2898 emails (948 spam and 1950 ham) when ($\lambda=999$) resulted in a marginal increase of 0.0003¹. The following figure shows the TCR values at ($\lambda=1$), ($\lambda=9$), and ($\lambda=999$) for the trigram-based spam filter:

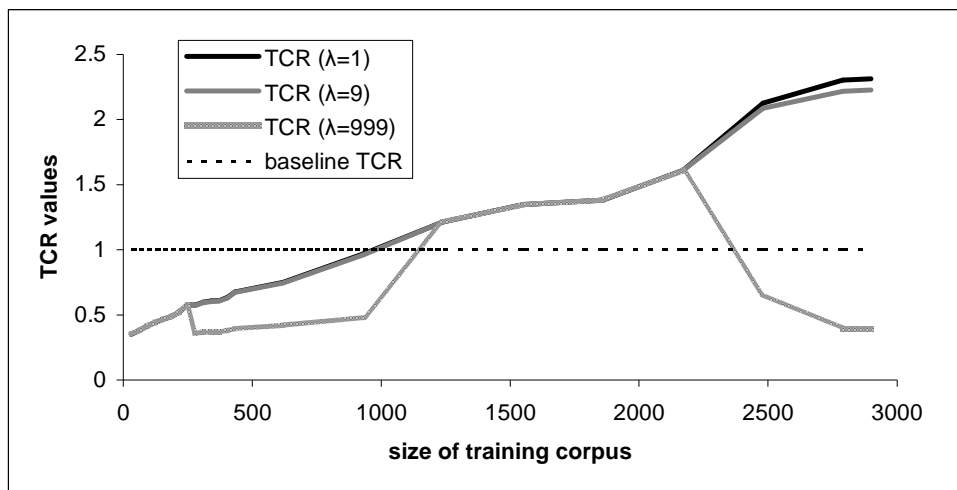


Figure 4: TCR values for the trigram-based spam filter.

¹ A table containing the raw results from the test on the trigram-based spam filter is available in Appendix J

Comparative summary statistics have been computed between the data obtained from the test on the word-based spam filter and the test on the trigram-based spam filter which are summarized in the following tables:

	Word	Trigram		Word	Trigram
Mean Recall	0.651	0.312	Mean Precision	0.996	0.999
Max Recall	0.888	0.655	Max Precision	1.0	1.0
r - Recall	0.819	0.957	r - Precision	0.016	-0.258
σ - Recall	0.134	0.192	σ - Precision	0.006	0.001

Table 4: Summary statistics for the recall and precision rates

	Word	Trigram		Word	Trigram		Word	Trigram
Mean TCR ($\lambda=1$)	2.625	0.914	Mean TCR ($\lambda=9$)	2.425	0.904	Mean TCR ($\lambda=999$)	0.389	0.589
Max TCR ($\lambda=1$)	8.316	2.312	Max TCR ($\lambda=9$)	7.77	2.225	Max TCR ($\lambda=999$)	0.852	1.61
r - TCR ($\lambda=1$)	0.943	0.993	r - TCR ($\lambda=9$)	0.9	0.995	r - TCR ($\lambda=999$)	0.01	0.477
σ - TCR ($\lambda=1$)	1.61	0.616	σ - TCR ($\lambda=9$)	1.438	0.596	σ - TCR ($\lambda=999$)	0.21	0.377

Table 5: Summary statistics for the TCR values at ($\lambda=1$), ($\lambda=9$), and ($\lambda=999$)

7. Discussion and Conclusion

The main concern for this study was to examine the efficacy of word-based Bayesian spam filters as opposed to trigram-based Bayesian spam filters. The hypothesis was addressed by comparing the precision and recall rates as well as TCR in both instances (word-based vs. trigram-based).

The pattern of improvement in the recall rate differs in the two spam filters. Whilst the pattern in the word-based spam filter was of a rapid increase followed by stability, the pattern observed in the trigram-based spam filter was of a gradual increase. The maximum recall rate achieved by the word-based spam filter succeeded the rate achieved by the trigram-based spam filter. However, as we can infer from the results, little improvement in the recall rate was noticed after the

addition of sufficient emails to the training corpus. One explanation could be that an optimum amount of retained attributes in the classifier was achieved and subsequently the classifier reached the optimal possible efficacy considering external variables. The trigram-based spam filter on the other hand showed stronger positive correlation ($r = 0.957$) compared to the word-based filter ($r = 0.819$) with variability somewhat higher than the variability in the word-based spam filter as evident in the standard deviation of the results of the two experiments ($\sigma = 0.134$ vs. $\sigma = 0.192$). These results indicate that there is much room for improvement for the trigram-based spam filter and that the optimal amount of training to the classifier was not reached during the experiment. This raises the question of whether a sufficient amount of training would allow the trigram-based spam filter to achieve higher rates than the word-based spam filter. Although the statistics are in-favor of such assumption – indicating that at some point the recall rate of the trigram-based spam filter would succeed the recall rate of the word-based spam filter, future research should take this point into account and try accumulating enough emails to reach the theoretical optimal amount and test the maximum recall rate achievable by both spam filters. Moreover, the results from the two experiments indicate that the word-based spam filter requires far less amounts of training to reach acceptable recall rates compared to the trigram-based spam filter. Given a limited amount of emails available for training, the word-based spam filter poses a more suitable choice for filtering.

The precision of both spam filters were somewhat constant throughout the experiments. Both filters showed negligible correlation between the size of the training corpus and the precision rate with the trigram-based spam filter showing slight negative correlation. However, the extremely low variability evident in both filters gives little credence to the computed correlations. The mean precision for the trigram-based spam filter almost reached a perfect 1.0 with a mean precision of 0.999 whilst the word-based spam filter achieved a lesser level of precision with a mean precision of 0.996. Although the difference between the two precision levels is negligible at best, the higher variability of the precision rate of the word-based spam filter indicates that the precision of the trigram-based spam filter is comparatively

higher than the precision of the word-based spam filter given a variable size of the training corpus. These results concur with previous research (Sahami, *et al.*, 1998 and Lyon, *et al.*, 2006) that noted that higher levels of uniqueness and discriminatory potential are associated with the use of trigrams in the identification of text compared to single words.

The computed TCR for both spam filters succeeded the base-line TCR when ($\lambda=1$). However, the word-based spam filter was quicker than the trigram-based spam filter – requiring a far less amount of emails in the training corpus for the filter to succeed the base-line. This supports our previous assumption that the word-based spam filter requires fewer emails in the training corpus to reach acceptable efficacy levels than trigram-based filters. It should be noted that whilst the trigram-based spam filter reached a maximum TCR of 2.312, the word-based spam filter reached a maximum TCR of 8.316 indicating that the word-based filter has a potentially higher classification accuracy levels than the trigram-based spam filters. The high level of variability in the TCR values of the word-based spam filter indicates that whilst the word-based spam filter has the potential of reaching high levels of classification accuracy, the reliability of the classifier is far less than what could be achieved with the trigram-based spam filter. Indeed whilst the TCR values resulting from the test on the trigram-based spam filter increased gradually, the observed results from the test on the word-based spam filter showed several drops throughout the increments to the training corpus. In addition, this might indicate that the efficacy of the word-based spam filter is dependant on the quality and type of emails used in training whilst the trigram-based spam filter does not suffer as much and – in a way – more resistant to the effects of the variable content in emails added to the training corpus. Not much could be inferred from the computed TCR values when ($\lambda=9$). Indeed, the word-based spam filter showed similar pattern to the value observed when ($\lambda=1$) dropping only slightly as the amount of emails in the training corpus reached 960 emails (310 spam and 615 ham). However, the trigram-based spam filter sustained a similar TCR value to the one observed when ($\lambda=1$) up-to a point much further than the word-based spam filter. Additionally, whilst the correlation of the TCR values observed during the word-based spam filter test decreased to 0.9, the correlation of

the TCR values for the trigram-based spam filter increased slightly to 0.995 indicating that as the size of the training corpus increases, the TCR values observed when ($\lambda=9$) would increase in an almost equal amount to the increase observed when ($\lambda=1$). In other words, these results indicate that whilst the word-based spam filter was affected by the λ adjustment, the trigram-based spam filter was not affected. This supports our assumption that the trigram-based spam filter results in a much more reliable classification than the word-based spam filter. Additionally, this indicates that when the cost of false-positive classification is high, trigram-based classification is a saner choice than word-based classification. Adjusting the TCR values with ($\lambda=999$) showed interesting effects on both spam filters. The TCR values observed during the test on the word-based spam filter dropped below the baseline TCR reaching a maximum TCR of 0.852. On the other hand, the trigram-based spam filter crossed the baseline TCR at some point of the test reaching a maximum TCR of 1.61 far exceeding the maximum TCR observed on the word-based spam filter at ($\lambda=999$). Additionally, whilst the TCR values for the word-based spam filter had no correlation ($r = 0.01$), the TCR values for the trigram-based spam filter sustained a moderately positive correlation with ($r = 0.477$). At some points throughout the test on the trigram-based spam filter, adjustment with ($\lambda=999$) did not affect the reported TCR values at all indicating evidence in support of our previous assumption that the trigram-based spam filter has a stronger ability to prevent the occurrence of false-positive classification as well as the assumption that trigram-based tokenization results in a stronger potential for spam identification and classification.

The results of the experiment indicated that any addition of pre-categorized spam emails to the training corpus requires a proportionately equal increase in ham to sustain the improvement in the performance of the spam filter. As shown from the last step of the test where the remaining spam and ham emails from the collection of training emails were added to the training corpus, the recall rates as well as TCR value deteriorated as a result of the additions. In the word-based spam filter, the recall rate dropped from (Recall = 0.888) to (Recall = 0.887) and the TCR value dropped from (TCR = 8.316) to (TCR = 8.103). Raw results (See Appendix I & Appendix J) pointed out that whilst the number of *spam->spam* classification

increased (true positive), the number of *ham->spam*, the number of *ham->ham*, and the number of *unsure* decreased indicating deterioration in the accuracy of ham-only classification and improvement in the accuracy of spam-only classification.

Although the correlation between the ratio of spam to ham in the training corpus and the performance of the spam filter might indicate a relationship between the two.

Due to the insignificant decrease and the narrow range we could not confidently attribute the deterioration to the change in the proportion of spam to ham in the training corpus. Moreover, we could not explain the rapid increase in the TCR value when the amount of emails in the training corpus succeeded 2000 emails. One explanation could be that an amount of spam emails with highly incriminating tokens were added to the training corpus towards the end of the test runs. This raises the point of a possible relationship between the quality of emails used in training and the performance of the spam filter. Obviously, high quality spam and ham emails would contribute the overall performance of the spam filter and as such the performance levels indicated in this report should be taken in conjunction with the fact that a special set of emails were used for testing. Using different sets of emails for training and testing would therefore result in different performance levels being reported.

The cutoff point set to classify emails as “spam”, “ham”, or “unsure” were based on the fact that the mean *I* value is 0.5 and the assumption that *I* values are normally distributed. In reality, however, the *I* value could hardly be normally distributed. Often users receive either little amounts of spam or too much spam with the latter being more common nowadays. Because of this, we were hesitant to use the standard deviation as a cutoff point and the decision to do so was slightly arbitrary based on preliminary tests conducted prior to running the experiment. Previous research in spam filtering (e.g. Androutsopoulos *et. al.*, 2000a, Sahami, *et. al.*, 1998, Graham, 2002, Robinson, 2003, and others) used arbitrary values as cutoff points based on the specific circumstances and the observed results from preliminary test runs. Usually a higher level of false-positives and/or false-negatives indicates a need to adjust the cutoff point for a more accurate classification. The cutoff points remained constant throughout the two tests against word-based and trigram-based

classification to sustain consistency of results. The size of the training corpus, the accuracy of the classifier, and the possibility of a tainted training corpus should be taken into account when specifying the cutoff points for classifying emails. As such, in real-world applications, we strongly discourage a fixed cutoff point.

During results collection, one striking observation was that classifying email using the word-based classifier took substantially less time than what was required for the trigram-based classifier. Indeed, during the collection of results, we observed that trigram-based classification took roughly four times as much time than word-based classification. In practically every aspect of the classification process, twice as much processing was required. The training corpus and tokens generated from messages undergoing classification were 7.7 times as much in size (66664 tokens in the word-based corpus vs. 513277 tokens in the trigram-based corpus). This might explain the comparatively substantial amount of process time and memory needed for classification. This observation raises the question of whether substantial increase in the resource requirements is justifiable. Indeed as suggested by the results, given an equal amount of emails added to the training corpus of both classifiers, the word-based classifier outperformed the trigram-based classifier in the recall rates and TCR. Although little attention was put to optimize the code, classifying an email using any of the classifiers took an unexpectedly considerable amount of time. The use of the Python programming language might have contributed to the issue. If classification efficiency is a major concern, an alternative programming language with native to near-native execution speeds needs to be considered. Indeed, besides caching, little could be done to improve the efficiency of the implemented classifiers without introducing change to the classification methodology. For instance, whilst we chose not to make use of a Mutual Information (*MI*) measure as in (Sahami, *et. al.*, 1998), taking only significant tokens into consideration during classification might reduce the processing time whilst maintaining accuracy levels near to what would be observed from the current scheme.

From the results we can conclude that given a limited amount of emails available for training, the word-based spam filter is much more efficient than the

trigram-based spam filter. However, when enough emails are available for training and when the cost of false-positive classification is high, the trigram-based spam filter is a much saner choice than the word-based spam filter. Indeed, as discussed above, the trigram-based spam filter showed higher levels of precision and reliability than the word-based spam filter. Given an abundance of resources available to the classifier and a sufficient amount of pre-categorized emails, trigram tokenization is the better choice.

Future research could try examining in depth the relationship between the ratio of spam to ham in the training corpus and the performance of the Naïve Bayes spam filter. Additionally, a standardized approach to setting a spam threshold could be developed to statistically estimate the optimal threshold level after which we could confidently assume that a certain email is spam or ham. Moreover, examining the difference between using a MI measure as in (Sahami, et. al., 1998) could be evaluated to determine whether the use of a MI is justified and that little difference would be observed. During this study no tests were conducted to determine the optimal values for the strength s and assumed probability x used in the $f(w)$ formula during the trigram-based classification tasks. Equal values were used across the two spam filters, however, as we noted previously, increasing the strength s for the trigram-based spam filter resulted in an increased recall rate. Future research could try examining the effects of changing the value of the strength s variable in an attempt to determine the optimal value for trigram-based classifications. Additionally and as previously discussed, the ability to detect the optimal recall rate after which additions to the training corpus results in the deterioration of the recall rate should be assessed and a method to detect the optimal recall rate could help future developments in creating a more intelligent classifier that could determine when the user needs not to train the classifier. Furthermore, the ability to automatically train the classifier with new emails when the result of the classification are too strong in favor of a spam or ham classification could allow spam filters to substantially reduce the need for user interaction for training activities creating a more autonomous spam filter as the classifier becomes smarter with time.

Bibliography

Androutsopoulos, I., Koutsias, J., Chandrinou, K., Paliouras, G., & Spyropoulos, C., (2000). 'An Evaluation of Naïve Bayesian Anti-Spam Filtering'. Proceedings of 11th European Conference on Machine Learning. May 31 – June 2, 2000. Barcelona, Spain. 9-17

Androutsopoulos, I., Koutsias, J., Chandrinou, K., & Spyropoulos, C., (2000). 'An Experimental Comparison of Naïve Bayesian and Keyword-Based Anti-Spam Filtering with Personal E-mail Messages'. *Proceedings of the 23rd Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, July 24-28, 2000, Athens, Greece.

Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C., & Stamatopoulos, P., (2000). 'Learning to Filter Spam E-Mail: A Comparison of a Naïve Bayesian and a Memory-Based Approach'. *Proceedings of Fourth European Conference on Principles and Practice of Knowledge Discovery*. September 13-16, 2000. Lyon, France.

Bowers, D., & Harnett, D., (2008). *The State of Spam: A Monthly Report – November 2008* [online]. Accessed on 9th November 2008. Available at: http://eval.symantec.com/mktginfo/enterprise/other_resources/b-state_of_spam_report_11-2008.en-us.pdf

Bratko, A., Cormack, G., Filipic, B., Thomas, R., & Zupan, B., (2006). 'Spam Filtering Using Statistical Data Compression Models'. *Journal of Machine Learning Research*. 7, 2673-2698

Carreras, X., & Marquez, L., (2001). 'Boosting Trees for Anti-Spam Email Filtering'. *Proceedings of RANLP-2001*. September 5-7, 2001, Tzigrav Chark, Bulgaria, 58-64.

Friedman, N., Geiger, D., & Goldszmidt, M., (1997). 'Bayesian Network Classifiers'. *Machine Learning*, 29, 131-163.

GFI (2007) *Why Bayesian filtering is the most effective anti-spam technology* [online]. Accessed on 4th November 2008. Available at: <http://www.gfi.com/whitepapers/why-bayesian-filtering.pdf>

Goodman, J., Heckerman, D., & Rounthwaite, R., (2005). 'Stopping spam'. *Scientific American*, 292(4), 42-88, April 2005.

Graham, P. (2002). *A Plan for Spam* [online]. Accessed on 4th November 2008. Available at: <http://www.paulgraham.com/spam.html>

Graham, P. (2003). *Better Bayesian Filtering* [online]. Accessed on 4th November 2008. Available at: <http://www.paulgraham.com/better.html>

Hershkop, S., & Stolfo, S., (2005). 'Combining Email Models for False Positive Reduction'. *Proceedings of the Eleventh ACM International Conference on Knowledge Discovery and Data Mining*. August 21-24, 2005, Chicago, Illinois, USA.

Kanich, C., Kreibich, C., Levchenko, K., Enright, B., Voelker, G., Paxson, V., & Savage, S., (2008) 'Spamalytics: An Empirical Analysis of Spam Marketing Conversion'. *15th ACM Conference on Computer and Communication Security*. October 27-31, 2008, Alexandria, Virginia, USA.

Klensin, J. (ed) (2001). 'Simple Mail Transfer Protocol'. *RFC 2821* [online]. Accessed on 4th November 2008. Available at: <http://www.rfc-editor.org/rfc/rfc2821.txt>

Klensin, J. (2008). 'Simple Mail Transfer Protocol'. *RFC 5321* [online]. Accessed on 4th November 2008. Available at: <http://www.rfc-editor.org/rfc/rfc5321.txt>

Lai, C., & Tsai, M., (2004). 'An Empirical Performance Comparison of Machine Learning Methods for Spam E-mail Categorization'. *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*. December 5-8, 2004, Kitakyushu, Japan.

Little, R., & Folks, J., (1971). 'Asymptotic Optimality of Fisher's Method of Combining Independent Tests'. *Journal of the American Statistical Association*, 66, 802-806.

Lyon, C., Barrett, R., & Malcolm, J. (2006). Plagiarism Is Easy, But Also Easy To Detect. *Plagiarism: Cross-Disciplinary Studies in Plagiarism, Fabrication, and Falsification*, 1(5): 1-10

Metsis, V., Androustopoulos, I., & Paliouras, G., (2006). 'Spam Filtering with Naïve Bayes – Which Naïve Bayes?'. *Proceedings of the Third Conference on Email and Anti-Spam*. July 27-28, 2006, Mountain View, California, USA.

Meyer, T., & Whateley, B., (2004). 'SpamBayes: Effective open-source, Bayesian based, email classification system'. *Proceedings of the First Conference on Email and Anti-Spam*. July 30-31, 2004. Mountain View, California, USA.

Michelakis, E., Androustopoulos, I., Paliouras, G., Sakkis, G., & Stamatopoulos, P., (2004). 'Filtron: A Learning-Based Anti-Spam Filter'. *Proceedings of the First Conference on Email and Anti-Spam*, July 30-31, 2004. Mountain View, California, USA.

Pantel, P., & Lin, D., (1998). 'SpamCop: A Spam Classification & Organization Program'. *Processing of the 1998 Workshop in Learning for Text Categorization*. July 26-27, 1998, Madison, Wisconsin, USA.

Pollock, S., (1988). 'A Rule-Based Message Filtering System'. *ACM Transactions on Office Information Systems*, 6(3), 232-254

Provost, J., (1999). 'Naïve-Bayes vs. Rule-Learning in Classification of Email'. *Technical Report AI-TR-99-284*, The University of Texas at Austin, Department of Computer Sciences.

Roberts, P. (2003). 'Report: Spam costs \$874 per employee per year'. *Computer World* [online]. Accessed on 4th November 2008. Available at:
<http://www.computerworld.com/softwaretopics/software/groupware/story/0,10801,82705,00.html>

Robinson, G. (2003). 'A Statistical Approach to Spam Filtering'. *Linux Journal* [online]. 1st March. Accessed on 4th November 2008. Available at:
<http://www.linuxjournal.com/article/6467>

Russell, S. & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Prentice Hall.

Sahami, M. Dumais, S., Heckerman, D., & Horvitz, E. A Bayesian approach to filtering junk email., *AAAI Workshop on Learning for Text Categorization*, July 1998, Madison, Wisconsin. AAAI Technical Report WS-98-05

Salton, G., & McGill, M., (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill: New York, NY, USA.

Shannon, C. (1951). Prediction and entropy of printed English. In N. Sloane and A Wyner (Eds.), (1993). *Shannon: Collected Papers*. IEEE Press.

Wikipedia Contributors, (2008a) 'Latent semantic analysis'. *Wikipedia* [online]. Accessed on 4th November 2008. Available at:
http://en.wikipedia.org/wiki/Latent_semantic_analysis

Wikipedia Contributors, (2008b) 'Probabilistic latent semantic analysis'. *Wikipedia* [online]. Accessed on 4th November 2008. Available at:
http://en.wikipedia.org/wiki/Probabilistic_latent_semantic_analysis

Wikipedia Contributors, (2008c) 'Vector space model'. *Wikipedia* [online]. Accessed on 4th November 2008. Available at:
http://en.wikipedia.org/wiki/Vector_space_model

Wittel, G., & Wu., S., (2004). 'On Attacking Statistical Spam Filters'. *Proceedings of the First Conference on Email and Anti-Spam*. July 30-31, 2004. Mountain View, California, USA.

Yang, Y., (1999). 'An Evaluation of Statistical Approaches to Text Categorization'. *Information Retrieval*. 1, 69-90.

Yerazunis, W. (2003). Sparse Binary Polynomial Hashing and the CRM114, *2003 Cambridge Spam Conference Proceedings*, Cambridge, MA.

Zhang, L., Zhu, J., & Yao, T., (2004). 'An Evaluation of Statistical Spam Filtering Techniques'. *ACM Transactions on Asian Language Information Processing*, 3(4), 243-269.

Zipf, G. (1949). *Human Behaviour and the Principle of Least Effort*. Addison Wesley, Cambridge.

Appendix A

“headertoken.py” source-code

```
import re
import email

# Regular expressions to extract data from the header
received_host_re = re.compile(r'from ([a-z0-9._-]+[a-z])[\s]')
received_ip_re = re.compile(r'.*[[ ]((\d{1,3}\.?)\d{4})[ ]]')
message_id_re = re.compile(r'\s*<[^@]+@[^>+>\s*')
# Returns a list of tokens from the header of the message
# We only use Received, Message-id, From, Content-type, and Content-
charset tags
def get_head_tokens(msg):
    """ Returns a list of tokens from the header of the message.

    We only use Received, Message-Id, From, Content-Type, and
    Content-Charset tags.

    msg: a email.Message object representing the message
    """
    if msg:
        h_tokens = []
        # Received: tag
        received_tags = msg.get_all('Received')
        if received_tags:
            for received in received_tags:
                m = received_host_re.match(received)
                if m:
                    h_tokens.append('recv-
host:{0}'.format(m.group(1)))
                m = received_ip_re.match(received)
                if m:
                    h_tokens.append('recv-
ip:{0}'.format(m.group(1)))
            else:
                h_tokens.append('recv-host:none')
                h_tokens.append('recv-ip:none')
        # Message-Id: tag
        msg_tags = msg.get_all('Message-id')
        if msg_tags:
            for msg_id in msg_tags:
                m = message_id_re.match(msg_id)
                if m:
                    h_tokens.append('msg-
id:{0}'.format(m.group(1)))
                else:
                    h_tokens.append('msg-id:none')
            else:
                h_tokens.append('msg-id:none')
        # From: tag
        from_tags = msg.get_all('From')
        if from_tags:
            for from_addr in from_tags:
                p_addr = email.utils.parseaddr(from_addr)
```

```

        if p_addr[0]:
            h_tokens.append('from-
real:{0}'.format(p_addr[0]))
        else:
            h_tokens.append('from-real:none')
        if p_addr[1]:
            h_tokens.append('from-
addr:{0}'.format(p_addr[1]))
        else:
            h_tokens.append('from-addr:none')
    else:
        h_tokens.append('from-real:none')
        h_tokens.append('from-addr:none')
    # Content-type: tag
    if msg.get_content_type():
        h_tokens.append('content-
type:{0}'.format(msg.get_content_type()))
    else:
        h_tokens.append('content-type:none')
    # Content-charset: tag
    if msg.get_content_charset():
        h_tokens.append('content-
charset:{0}'.format(msg.get_content_charset()))
    else:
        h_tokens.append('content-charset:none')
    return set(h_tokens)
else:
    return None

```

Appendix B

“bodytoken.py” source-code

```
import re
import email.utils
import options

def get_multipart_message(msg):
    """ Returns the textual content of a multipart-message.

    This function is meant to be used to recursively retrieve
    the textual contents of emails.

    msg: a email.Message object representing the email.
    """
    body = ''
    for msg1 in msg.get_payload():
        if msg1.is_multipart():
            body += ' ' + get_multipart_message(msg1)
        else:
            body += ' ' + msg1.get_payload()
    return body

def get_body_tokens(msg, is_trigram=options.IS_TRIGRAM):
    """ Returns a list of tokens from the email's body.

    In addition to tokens from the email's body, the Subject
    header
    tag is tokenized here. This is deliberately done so that the
    is_trigram
    argument is not repeated when retrieving the header tokens.

    msg: a email.Message object representing the email.
    is_trigram: True if trigram tokenization is used.
    [Default=False].
    """
    if msg:
        # We are basically stripping words using str.split()
        # specs
        # * Words between 3 and 12 chars inclusive are selected
        # * For trigram tokenization, words less than 12 chars
        # * lower case all words
        # * replace non-alphanumeric chars with \s except $!-@%?

        # * currently no base64 decoding
        # * Discard single/twin non-alphanums in trigram
        tokenization
        b_tokens = []
        #Tokenize the subject here
        subjects = []
        subject_string = msg.get('subject')
        if subject_string:
            subject_string = re.sub(options.FILTER_PATTERN, '
', subject_string)
```

```

        subjects = subject_string.split(' ')
# clear the collection
# we don't want single/twin char non-alphanums
# we don't want whitespaces
for word in subjects:
    if len(word) < 3:
        if not word.isalnum() or word.isspace():
            subjects.remove(word)
while(' ' in subjects):
    subjects.remove(' ')
# Add subject tokens to the list
if subjects:
    if is_trigram:
        for i in range(0,len(subjects)):
            if ((i+1) < len(subjects)) and ((i+2)
< len(subjects)):
                if (options.MIN_WORD_TRIGRAM <=
len(subjects[i]) <= options.MAX_WORD_TRIGRAM) and
(options.MIN_WORD_TRIGRAM <= len(subjects[i+1]) <=
options.MAX_WORD_TRIGRAM) and (options.MIN_WORD_TRIGRAM <=
len(subjects[i+2]) <= options.MAX_WORD_TRIGRAM):
                    b_tokens.append('subj:{0}
{1} {2}'.format(subjects[i].lower(),subjects[i+1].lower(),
subjects[i+2].lower()))
                else:
                    for word in subjects:
                        if options.MIN_WORD_SINGLE <=
len(word) <= options.MAX_WORD_SINGLE:
                            b_tokens.append('subj:{0}'.format(word))

    for msg1 in msg.walk():
        body = ''
        if msg1.is_multipart():
            body = get_multipart_message(msg1)

        else:
            body = msg1.get_payload()

    if body:
        # first remove the non-alphanums
        body = re.sub(options.FILTER_PATTERN,' ',
body)

        # split the body
        words = body.split(' ')
        # remove single/twin non-alphanums
        # remove whitespace
        for word in words:
            if len(word) <
options.MAX_FILTER_NON_ALNUM:
                if not word.isalnum() or
word.isspace():
                    words.remove(word)
        while (' ' in words):
            words.remove(' ')
        # only take words between 3 & 12 inclusive
        if not is_trigram:

```

```

        for word in words:
            if options.MIN_WORD_SINGLE <=
len(word) <= options.MAX_WORD_SINGLE:

                b_tokens.append(word.lower())
            else:
                # take words less than 12 inclusive
                for i in range(0,len(words)):
                    if ((i+1) < len(words)) and
((i+2) < len(words)):
                        if
(options.MIN_WORD_TRIGRAM <= len(words[i]) <=
options.MAX_WORD_TRIGRAM) and (options.MIN_WORD_TRIGRAM <=
len(words[i+1]) <= options.MAX_WORD_TRIGRAM) and
(options.MIN_WORD_TRIGRAM <= len(words[i+2]) <=
options.MAX_WORD_TRIGRAM):
                            b_tokens.append('{0}
{1}
{2}'.format(words[i].lower(),words[i+1].lower(),words[i+2].lower()))

                                # we use set() to remove duplicates
                                return set(b_tokens)
                    else:
                        return None

```

Appendix C

“random-select-emails.py” source-code

```
# This script randomly selects files from the corpus of emails taken
# from Spam Assassin and sorts them into folders in preparation for
# running the testa.py and testb.py test scripts
```

```
import os
import random

file_list = os.listdir('.\\col\\spam')
max_list = len(file_list)

for i in range(0,max_list/2):
    ix = random.randint(0,len(file_list) - 1)
    os.rename('.\\col\\spam\\{0}'.format(file_list[ix]),
'.\\train\\spam\\{0}'.format(file_list[ix]))
    file_list.pop(ix)

file_list = os.listdir('.\\col\\spam_2')
max_list = len(file_list)

for i in range(0,max_list/2):
    ix = random.randint(0,len(file_list) - 1)
    os.rename('.\\col\\spam_2\\{0}'.format(file_list[ix]),
'.\\train\\spam\\{0}'.format(file_list[ix]))
    file_list.pop(ix)

file_list = os.listdir('.\\col\\easy_ham')
max_list = len(file_list)

for i in range(0,max_list/2):
    ix = random.randint(0,len(file_list) - 1)
    os.rename('.\\col\\easy_ham\\{0}'.format(file_list[ix]),
'.\\train\\ham\\{0}'.format(file_list[ix]))
    file_list.pop(ix)

file_list = os.listdir('.\\col\\easy_ham_2')
max_list = len(file_list)

for i in range(0,max_list/2):
    ix = random.randint(0,len(file_list) - 1)
    os.rename('.\\col\\easy_ham_2\\{0}'.format(file_list[ix]),
'.\\train\\ham\\{0}'.format(file_list[ix]))
    file_list.pop(ix)
```

Appendix D

“classifier.py” source-code

```
import math
import tokenizer
import trainer
import options

# GLOBAL CONSTANTS
SPAM = 1
UNSURE = 0
HAM = -1

# -----
# START OF COPYRIGHT NOTICE
# The following code is copyright (c) 2003, Gary Robinson
def chi2P(chi, df):
    """Return prob(chisq >= chi, with df degrees of
    freedom).

    df must be even.
    """
    assert df & 1 == 0
    # XXX If chi is very large, exp(-m) will underflow to 0.
    m = chi / 2.0
    sum = term = math.exp(-m)
    for i in range(1, df//2):
        term *= m / i
        sum += term
    # With small chi and large df, accumulated
    # roundoff error, plus error in
    # the platform exp(), can cause this to spill
    # a few ULP above 1.0. For
    # example, chi2P(100, 300) on my box
    # has sum == 1.0 + 2.0**(-52) at this
    # point. Returning a value even a teensy
    # bit over 1.0 is no good.
    return min(sum, 1.0)
# END OF COPYRIGHT NOTICE
# -----
def calc_I(train_corpus, msg_text, is_trigram=options.IS_TRIGRAM):
    """ Calculate and return the I value based on the provided
    training
        corpus and the textual content of the message.

        train_corpus: a tuple containing the training corpus.
    The tuple must
        consist of spam_tokens, spam_n, ham_tokens, ham_n,
    train_cache.
        msg_text: a string containing the email in it's original
    source.
        is_trigram: True if trigram tokenization is used.
    [Default=False].
    """
```

```

# implement :
#     H = C^-1 (-2 ln (f(w)1...f(w)n, 2n)
#     S = same as H however for f(w) = 1-f(w)
#     I = 1 + H - S / 2

#Tokenize msg_text
msg_tokens = tokenizer.tokenize_from_string(msg_text,
is_trigram)
H_logs = 1.0
S_logs = 1.0
# As we know the sum of logs = the log of products
for word in msg_tokens:
    try:
        H_logs *= trainer.calc_prob(train_corpus, word)
        if H_logs == 0.0:
            break
    # if the key was not found ignore
    except KeyError:
        pass
    try:
        S_logs *= (1 - trainer.calc_prob(train_corpus,
word))
        if S_logs == 0.0:
            break
    # if the key was not found ignore
    except KeyError:
        pass
# Sometimes the logs underflow to 0. We used the smallest
# log possible for a 64 bit floating point in Python 32bit
if H_logs == 0.0:
    H_logs = -743.74692474082133
else:
    H_logs = math.log(H_logs)
if S_logs == 0.0:
    S_logs = -743.74692474082133
else:
    S_logs = math.log(S_logs)

H = chi2P(-2 * H_logs, 2 * len(msg_tokens))
S = chi2P(-2 * S_logs, 2 * len(msg_tokens))
I = ( 1 + H - S) / 2.0
return I

def classify(train_corpus, msg_text, is_trigram=options.IS_TRIGRAM):
    """ Return the classification of a message.

        either classifier.SPAM, classifier.HAM, or
classifier.UNSURE.

        train_corpus: a tuple containing the training corpus.
The tuple must
        consist of spam_tokens, spam_n, ham_tokens, ham_n,
train_cache.
        msg_text: a string containing the email in it's original
source.
        is_trigram: True if trigram tokenization is used.
[Default=False].

```

```
"""
# Calculate I
I = calc_I(train_corpus, msg_text, is_trigram)
# Return classification based on cutoff
if I >= options.SPAM_CUTOFF:
    return SPAM
elif options.SPAM_CUTOFF > I > options.HAM_CUTOFF:
    return UNSURE
elif I <= options.HAM_CUTOFF:
    return HAM
```

Appendix E

“trainer.py” source-code

```
import tokenizer
import sys
import pickle
import options

# GLOBAL CONSTANTS
SPAM = 1
UNSURE = 0
HAM = -1

def create_train_corpus(spam_path, ham_path,
is_trigram=options.IS_TRIGRAM):
    """ Create a training corpus tuple from a collection of spam
and ham emails.

    spam_path: the path to the collection of spam emails
    ham_path: the path to the collection of ham emails
    is_trigram: True if trigram tokenization is used.
[Default=False].
    """
    spam_tokens, spam_n = tokenizer.tokenize_from_dir(spam_path,
is_trigram)
    ham_tokens, ham_n = tokenizer.tokenize_from_dir(ham_path,
is_trigram)
    train_cache = {}
    return (spam_tokens, spam_n, ham_tokens, ham_n, train_cache)

def load_train_corpus(file_path):
    """ Load a serialized training corpus from a file.

    file_path: the path to the serialized training corpus
    """
    f = open(file_path, 'rb')
    if f:
        train_corpus = pickle.load(f)
        f.close()
    #check if the training corpus is trigram tokenized
    word = train_corpus[0].keys()[0]
    words = word.split(' ')
    assert((len(words) == 1) or (len(words) == 3))
    if len(words) == 1:
        options.set_option('trigram', False)
    elif len(words) == 3:
        options.set_option('trigram', True)
    return train_corpus

def save_train_corpus(file_path, train_corpus):
    """ Serializes and saves the training corpus to file.

    file_path: where to save the serialized training corpus
```

```

    train_corpus: a tuple containing the training corpus. The
tuple must
    consist of spam_tokens, spam_n, ham_tokens, ham_n,
train_cache.
"""
    f = open(file_path, 'wb')
    if f:
        pickle.dump(train_corpus, f)
        f.close()
        return True
    return False

def add_to_train_corpus(source_train_corpus, msg_type, msg_text,
is_trigram=options.IS_TRIGRAM):
    """ Adds an email to the specified training corpus and returns
a new training corpus

    source_train_corpus: the source training corpus.
    msg_type: the type of the message. Either trainer.SPAM or
trainer.HAM.
    msg_text: a string containing the email in it's original
source.
    is_trigram: True if trigram tokenization is used.
[Default=False].
"""
    # dettach the tuple
    spam_tokens, spam_n, ham_tokens, ham_n, train_cache =
source_train_corpus
    if msg_type == SPAM:
        msg_tokens = tokenizer.tokenize_from_string(msg_text,
is_trigram)
        for word in msg_tokens:
            if word not in spam_tokens:
                spam_tokens[word] = 1
            else:
                spam_tokens[word] += 1
        spam_n += 1
    elif msg_type == HAM:
        msg_tokens = tokenizer.tokenize_from_string(msg_text,
is_trigram)
        for word in msg_tokens:
            if word not in ham_tokens:
                ham_tokens[word] = 1
            else:
                ham_tokens[word] += 1
        ham_n += 1
    # reset the cache
    train_cache = {}
    return (spam_tokens, spam_n, ham_tokens, ham_n, train_cache)

def calc_prob(train_corpus, word):
    """ Calculate the f(w) value of the specified word given the
training corpus.

    train_corpus: a tuple containing the training corpus. The
tuple must

```

```

        consist of spam_tokens, spam_n, ham_tokens, ham_n,
train_cache.
    word: the word that will be tested
    """
    # dettach the tuple
    spam_tokens, spam_n, ham_tokens, ham_n, train_cache =
train_corpus
    # set f_w to the default value
    f_w = options.DEFAULT_F_W
    # must lower case
    word = word.lower()
    # First check if f_w is cached
    if word not in train_cache.keys():
        # not cached so we calculate the value
        # apply f(w) formula
        # Is the word in the spam tokens
        if word in spam_tokens.keys():
            s = options.S
            x = options.X
            s_w = float(spam_tokens[word]) / spam_n
            h_w = 0.0
            if word in ham_tokens.keys():
                h_w = float(ham_tokens[word]) / ham_n
            p_w = s_w / (h_w + s_w)
            n = float(spam_tokens[word])
            if word in ham_tokens.keys():
                n += float(ham_tokens[word])
            f_w = ((s * x) + (n * p_w)) / (s + n)
        # No? then is it in the ham tokens?
        elif word in ham_tokens.keys():
            s = options.S
            x = options.X
            h_w = float(ham_tokens[word]) / ham_n
            s_w = 0.0
            if word in spam_tokens.keys():
                s_w = float(spam_tokens[word]) / spam_n
                print "This shouldn't happen"
                sys.exit()
            p_w = s_w / (h_w + s_w)
            n = float(ham_tokens[word])
            if word in spam_tokens.keys():
                n += float(spam_tokens[word])
            f_w = ((s * x) + (n * p_w)) / (s + n)
        # cache the value (if the word is not in the training
corpus use the default value)
        train_cache[word] = f_w
    else:
        # retrieve value from from cache
        f_w = train_cache[word]
    return f_w

```

Appendix F

“testa.py” source-code

```
# This script tests the word-based classifier and then stores
# the results to a csv file for later analysis.

# ----[ WORD WORD WORD ] ----
import sys
import csv
import os
import multiprocessing
import tokenizer
import trainer
import classifier
import options

class Worker(multiprocessing.Process):
    """ This class spreads the classification task into two
    processes
    to take advantage of multi-processor PCs.
    """
    data1 = None
    data2 = None
    pipe = None
    def __init__(self,data1,data2,pipe):
        multiprocessing.Process.__init__(self)
        self.data1 = data1
        self.data2 = data2
        self.pipe = pipe
    def run(self):
        cr1 = classifier.classify(self.data1, self.data2, False)

        self.pipe.send(cr1)
# We are measuring the following:
# * Spam -> Spam S_S
# * Ham -> Ham H_H
# * Ham -> Spam H_S
# * spam -> Ham S_H
# * Unsure U
# Emails are taken from .\\train\\spam and .\\train\\ham for
training
# and from .\\test\\spam and .\\test\\ham for testing
# 948 spams for training
# 1950 hams for training
# 948 spams for testing
# 1950 hams for testing
# steps are the following:
# A. 100X spam training 210X ham training

if __name__ == '__main__':
    train_corpus1 = ()
    cr1 = 0
    cr2 = 0
```

```

train_spam_list = os.listdir('.\\train\\spam')
train_ham_list = os.listdir('.\\train\\ham')
test_spam_list = os.listdir('.\\test\\spam')
test_ham_list = os.listdir('.\\test\\ham')
assert(len(train_spam_list) == 948)
assert(len(train_ham_list) == 1950)
assert(len(test_spam_list) == 948)
assert(len(test_ham_list) == 1950)
f = open('results-a-a.txt','w')
fp = open('results-fp-a.txt','w')
cwriter = csv.writer(f)
cwriter.writerow(['S_S1','S_H1','H_S1','H_H1','U1','i+10'])
progress = 0
total_progress = 0
# the range tests i to i+99
print 'Starting Step A\nTraining the initial corpora...'
p1_c, p1_s = multiprocessing.Pipe()
p2_c, p2_s = multiprocessing.Pipe()
train_corpus1 = trainer.create_train_corpus(None, None, False)
print '...Done'
for i in range(0, 948, 100):
    #increment spam corpus
    print 'Adding 100 spam, 210 ham emails to the training
corpus...'
    for p in range(0,210):
        if len(train_ham_list) > int(p+i*2.1):
            msg_f1 =
open('{0}{1}{2}'.format('.\\train\\ham',os.sep,train_ham_list[int(p+
i*2.1)]), 'r')
            msg_txt1 = msg_f1.read()
            msg_f1.close()
            train_corpus1 =
trainer.add_to_train_corpus(train_corpus1, trainer.HAM, msg_txt1,
False)
            for p in range(0,100):
                if len(train_spam_list) > p+i:
                    msg_f1 =
open('{0}{1}{2}'.format('.\\train\\spam',os.sep,train_spam_list[p+i]
), 'r')
                    msg_txt1 = msg_f1.read()
                    msg_f1.close()
                    train_corpus1 =
trainer.add_to_train_corpus(train_corpus1, trainer.SPAM, msg_txt1,
False)
                    trainer.save_train_corpus('train-corpus-a-i-
{0}.dat'.format(i), train_corpus1)
                    print '...Done'
                    #test the spam
                    print 'Testing Spam...'
                    progress = 0
                    S_S1 = 0
                    S_H1 = 0
                    U1 = 0
                    for p in range(0,948,2):
                        msg_f =
open('{0}{1}{2}'.format('.\\test\\spam',os.sep,test_spam_list[p]),
'r')

```

```

        msg_text = msg_f.read()
        msg_f.close()
        msg_f2 =
open('{0}{1}{2}'.format('.\\test\\spam',os.sep,test_spam_list[p+1]),
'r')

        msg_text2 = msg_f2.read()
        msg_f2.close()
        c5 = Worker(train_corpus1,msg_text,p1_c)
        c6 = Worker(train_corpus1,msg_text2,p2_c)
        c5.start()
        c6.start()
        cr1 = p1_s.recv()
        cr2 = p2_s.recv()
        c6.join()
        c5.join()
        if cr1 == classifier.SPAM:
            S_S1 += 1
        elif cr1 == classifier.HAM:
            S_H1 += 1
        elif cr1 == classifier.UNSURE:
            U1 += 1
        if cr2 == classifier.SPAM:
            S_S1 += 1
        elif cr2 == classifier.HAM:
            S_H1 += 1
        elif cr2 == classifier.UNSURE:
            U1 += 1
        progress += 2
        print '\rStep progress: %{0:4.2}\tTask progress:
%{1:4.2}
(float(progress)/948)*100),
        #test the ham
        print '\r...Done
,

        print 'Testing Ham...'
        progress = 0
        H_S1 = 0
        H_H1 = 0
        for p in range(0,1950,2):
            msg_f =
open('{0}{1}{2}'.format('.\\test\\ham',os.sep,test_ham_list[p]),
'r')

            msg_text = msg_f.read()
            msg_f.close()
            msg_f2 =
open('{0}{1}{2}'.format('.\\test\\ham',os.sep,test_ham_list[p+1]),
'r')

            msg_text2 = msg_f2.read()
            msg_f2.close()
            c7 = Worker(train_corpus1,msg_text,p1_c)
            c8 = Worker(train_corpus1,msg_text2,p2_c)
            c7.start()
            c8.start()
            cr1 = p1_s.recv()
            cr2 = p2_s.recv()
            c8.join()
            c7.join()

```

```

        if cr1 == classifier.SPAM:
            H_S1 += 1

fp.write('.\\test\\ham\\{0}\\r\\n'.format(test_ham_list[p]))
fp.flush()
        elif cr1 == classifier.HAM:
            H_H1 += 1
        elif cr1 == classifier.UNSURE:
            U1 += 1
        if cr2 == classifier.SPAM:
            H_S1 += 1

fp.write('.\\test\\ham\\{0}\\r\\n'.format(test_ham_list[p+1]))
fp.flush()
        elif cr2 == classifier.HAM:
            H_H1 += 1
        elif cr2 == classifier.UNSURE:
            U1 += 1
        progress += 2
        print '\rStep progress: %0:4.2}\tTask progress:
%{1:4.2}
(float(progress)/1950)*100),
        print '\r...Done
'

        # Writer results to file
        cwriter.writerow([S_S1,S_H1,H_S1,H_H1,U1,i+10])
        f.flush()
        print 'Results written to file...'
        total_progress += 100
print 'Experiment complete! :)'
f.close()
fp.close()

```

Appendix G

“testb.py” source-code

```
# This script tests the trigram-based classifier and then stores
# the results to a csv file for later analysis.

# ----[ TRIGRAM TRIGRAM TRIGRAM ] ----
import sys
import csv
import os
import multiprocessing
import tokenizer
import trainer
import classifier
import options

class Worker(multiprocessing.Process):
    data1 = None
    data2 = None
    pipe = None
    def __init__(self,data1,data2,pipe):
        multiprocessing.Process.__init__(self)
        self.data1 = data1
        self.data2 = data2
        self.pipe = pipe
    def run(self):
        cr1 = classifier.classify(self.data1, self.data2, True)

        self.pipe.send(cr1)
# We are measuring the following:
# * Spam -> Spam S_S
# * Ham -> Ham H_H
# * Ham -> Spam H_S
# * spam -> Ham S_H
# * Unsure U
# Emails are taken from .\\train\\spam and .\\train\\ham for
training
# and from .\\test\\spam and .\\test\\ham for testing
# 948 spams for training
# 1950 hams for training
# 948 spams for testing
# 1950 hams for testing
# steps are the following:
# B. 100X spam training 210X ham training

if __name__ == '__main__':
    train_corpus1 = ()
    cr1 = 0
    cr2 = 0
    train_spam_list = os.listdir('.\\train\\spam')
    train_ham_list = os.listdir('.\\train\\ham')
    test_spam_list = os.listdir('.\\test\\spam')
    test_ham_list = os.listdir('.\\test\\ham')
    assert(len(train_spam_list) == 948)
```

```

assert(len(train_ham_list) == 1950)
assert(len(test_spam_list) == 948)
assert(len(test_ham_list) == 1950)
f = open('results-a-b.txt','w')
fp = open('results-fp-b.txt','w')
cwriter = csv.writer(f)
cwriter.writerow(['S_S1','S_H1','H_S1','H_H1','U1','i+10'])
progress = 0
total_progress = 0
# the range tests i to i+99
print 'Starting Step A\nTraining the initial corpora...'
p1_c, p1_s = multiprocessing.Pipe()
p2_c, p2_s = multiprocessing.Pipe()
train_corpus1 = trainer.create_train_corpus(None, None, True)
print '...Done'
for i in range(0, 948, 100):
    #increment spam corpus
    print 'Adding 100 spam, 210 ham emails to the training
corpus...'
    for p in range(0,210):
        if len(train_ham_list) > int(p+i*2.1):
            msg_f1 =
open('{0}{1}{2}'.format('.\\train\\ham',os.sep,train_ham_list[int(p+
i*2.1)]), 'r')

            msg_txt1 = msg_f1.read()
            msg_f1.close()
            train_corpus1 =
trainer.add_to_train_corpus(train_corpus1, trainer.HAM, msg_txt1,
True)

    for p in range(0,100):
        if len(train_spam_list) > p+i:
            msg_f1 =
open('{0}{1}{2}'.format('.\\train\\spam',os.sep,train_spam_list[p+i]
), 'r')

            msg_txt1 = msg_f1.read()
            msg_f1.close()
            train_corpus1 =
trainer.add_to_train_corpus(train_corpus1, trainer.SPAM, msg_txt1,
True)

    trainer.save_train_corpus('train-corpus-b-i-
{0}.dat'.format(i), train_corpus1)
    print '...Done'
    #test the spam
    print 'Testing Spam...'
    progress = 0
    S_S1 = 0
    S_H1 = 0
    U1 = 0
    for p in range(0,948,2):
        msg_f =
open('{0}{1}{2}'.format('.\\test\\spam',os.sep,test_spam_list[p]),
'r')

        msg_text = msg_f.read()
        msg_f.close()
        msg_f2 =
open('{0}{1}{2}'.format('.\\test\\spam',os.sep,test_spam_list[p+1]),
'r')

```

```

msg_text2 = msg_f2.read()
msg_f2.close()
c5 = Worker(train_corpus1,msg_text,p1_c)
c6 = Worker(train_corpus1,msg_text2,p2_c)
c5.start()
c6.start()
cr1 = p1_s.recv()
cr2 = p2_s.recv()
c6.join()
c5.join()
if cr1 == classifier.SPAM:
    S_S1 += 1
elif cr1 == classifier.HAM:
    S_H1 += 1
elif cr1 == classifier.UNSURE:
    U1 += 1
if cr2 == classifier.SPAM:
    S_S1 += 1
elif cr2 == classifier.HAM:
    S_H1 += 1
elif cr2 == classifier.UNSURE:
    U1 += 1
progress += 2
print '\rStep progress: %{0:4.3}\tTask progress:
%{1:4.3}
(float(progress)/948)*100),
(float(total_progress)/948)*100),
    #test the ham
    print '\r...Done
'

    print 'Testing Ham...'
    progress = 0
    H_S1 = 0
    H_H1 = 0
    for p in range(0,1950,2):
        msg_f =
open('{0}{1}{2}'.format('.',os.sep,test_ham_list[p]),
'r')

        msg_text = msg_f.read()
        msg_f.close()
        msg_f2 =
open('{0}{1}{2}'.format('.',os.sep,test_ham_list[p+1]),
'r')

        msg_text2 = msg_f2.read()
        msg_f2.close()
        c7 = Worker(train_corpus1,msg_text,p1_c)
        c8 = Worker(train_corpus1,msg_text2,p2_c)
        c7.start()
        c8.start()
        cr1 = p1_s.recv()
        cr2 = p2_s.recv()
        c8.join()
        c7.join()
        if cr1 == classifier.SPAM:
            H_S1 += 1

fp.write('.',os.sep,test_ham_list[p])
fp.flush()

```

```

        elif cr1 == classifier.HAM:
            H_H1 += 1
        elif cr1 == classifier.UNSURE:
            U1 += 1
        if cr2 == classifier.SPAM:
            H_S1 += 1

fp.write('.\\test\\ham\\{0}\\r\\n'.format(test_ham_list[p+1]))
fp.flush()
        elif cr2 == classifier.HAM:
            H_H1 += 1
        elif cr2 == classifier.UNSURE:
            U1 += 1
        progress += 2
        print '\rStep progress: %0:4.3}\tTask progress:
%{1:4.3}          '.format((float(total_progress)/948)*100,
(float(progress)/1950)*100),
        print '\r...Done
'

        # Writer results to file
        cwriter.writerow([S_S1,S_H1,H_S1,H_H1,U1,i+10])
        f.flush()
        print 'Results written to file...'
        total_progress += 100
print 'Experiment complete! :)'
f.close()
fp.close()

```

Appendix H

“options.py” source-code

```
import pickle

IS_TRIGRAM = False # Are we using trigram tokenization
FILTER_PATTERN = r'\s|[\^\w\$\!\-\%\`']' # filtering pattern to remove
from words
MIN_WORD_SINGLE = 3 # the minimum char length of words in word-based
tokens
MAX_WORD_SINGLE = 12 # the maximum char-length of words in word-
based tokens
MIN_WORD_TRIGRAM = 0 # the minimum char length of words in trigram-
based tokens
MAX_WORD_TRIGRAM = 12 # the maximum char length of words in trigram-
based tokens
MAX_FILTER_NON_ALNUM = 3 # the maximum char length of words in
trigram-based tokens to filter out
DEFAULT_F_W = 0.5 # the default f(w) value when a word does not
exist in the corpus
S = 0.45 # the strength of the f(w) calculation
X = 0.5 # the assumed probability of f(w) must match DEFAULT_F_W
SPAM_CUTOFF = 0.683 # the spam cutoff value currently set at +1 S.D.
from DEFAULT_F_W
HAM_CUTOFF = 0.317 # the ham cutoff value currently set at -1 S.D.
from DEFAULT_F_W

def load_options():
    """ load options from presistant storage
    """
    f = open('options.dat', 'rb')
    if f:
        opt = pickle.load(f)
        f.close()
    IS_TRIGRAM, FILTER_PATTERN, MIN_WORD_SINGLE, MAX_WORD_SINGLE,
    MIN_WORD_TRIGRAM, MAX_WORD_TRIGRAM, MAX_FILTER_NON_ALNUM,
    DEFAULT_F_W,
    S, X, SPAM_CUTOFF, HAM_CUTOFF = opt

def save_options():
    """ Saves options to presistant storage
    """
    f = open('options.dat', 'wb')
    if f:
        opt = IS_TRIGRAM, FILTER_PATTERN, MIN_WORD_SINGLE,
    MAX_WORD_SINGLE,
        MIN_WORD_TRIGRAM, MAX_WORD_TRIGRAM,
    MAX_FILTER_NON_ALNUM, DEFAULT_F_W,
        S, X, SPAM_CUTOFF, HAM_CUTOFF
        pickle.dump(opt, f)
        f.close()
    return True
return False

def set_option(option, value):
```

```

""" Sets specific options.

option: the option to set. Possible options are "trigram",
"filter pattern",
"min single word", "max single word", "min trigram word", "max
trigram word",
"max non-alnum filter", "default f(w)", "s", "x", "spam
cutoff", "ham cutoff"
value: the value to set
"""
if option == 'trigram':
    IS_TRIGRAM = value
if option == 'filter pattern':
    FILTER_PATTERN = value
if option == 'min single word':
    MIN_WORD_SINGLE = value
if option == 'max single word':
    MAX_WORD_SINGLE = value
if option == 'min trigram word':
    MIN_WORD_TRIGRAM = value
if option == 'max trigram word':
    MAX_WORD_TRIGRAM = value
if option == 'max non-alnum filter':
    MAX_FILTER_NON_ALNUM = value
if option == 'default f(w)':
    DEFAULT_F_W= value
if option == 's':
    S = value
if option == 'x':
    X = value
if option == 'spam cutoff':
    SPAM_CUTOFF = value
if option == 'ham cutoff':
    HAM_CUTOFF = value

```

Appendix I

Raw results from the word-based spam filter test

S_S1	S_H1	H_S1	H_H1	U1	spam	ham	total	recall	precision	TCR	TCR/9	TCR/999
429	0	0	844	1625	10	21	31	0.2089	1	0.5834	0.5834	0.58338
640	3	1	1123	1131	20	42	62	0.3608	0.99844	0.8352	0.8294	0.44444
666	13	2	1283	934	30	63	93	0.4129	0.99701	0.9989	0.9824	0.3219
726	12	7	1215	938	40	84	124	0.4332	0.99045	0.9906	0.9358	0.11935
802	4	33	924	1135	50	105	155	0.4132	0.96048	0.8089	0.6602	0.0278
779	5	15	1148	951	60	126	186	0.449	0.98111	0.9763	0.8689	0.05947
767	8	3	1371	749	70	147	217	0.5033	0.9961	1.2474	1.2092	0.25253
812	4	2	1442	638	80	168	248	0.5585	0.99754	1.472	1.4364	0.35909
813	4	2	1394	685	90	189	279	0.5413	0.99755	1.3719	1.3409	0.35281
810	5	1	1501	581	100	210	310	0.5802	0.99877	1.615	1.5933	0.59811
810	5	1	1501	581	100	210	310	0.5802	0.99877	1.615	1.5933	0.59811
815	6	1	1516	560	110	231	341	0.5902	0.99877	1.672	1.6487	0.60575
819	5	2	1467	605	120	252	372	0.5731	0.99756	1.549	1.5096	0.3635
806	5	1	1585	501	130	273	403	0.6143	0.99876	1.8698	1.8408	0.6299
803	6	1	1631	457	140	294	434	0.6343	0.99876	2.0431	2.0085	0.64843
792	4	1	1715	386	150	315	465	0.6701	0.99874	2.4246	2.3759	0.68251
781	3	1	1732	381	160	336	496	0.6704	0.99872	2.4623	2.4122	0.68547
785	4	1	1751	357	170	357	527	0.685	0.99873	2.6188	2.5622	0.69706
796	4	1	1756	341	180	378	558	0.6976	0.99875	2.7399	2.678	0.70536
803	4	1	1731	359	190	399	589	0.6887	0.99876	2.6044	2.5484	0.69604
814	3	2	1704	375	200	420	620	0.6829	0.99755	2.4947	2.3939	0.39899
814	3	2	1704	375	200	420	620	0.6829	0.99755	2.4947	2.3939	0.39899
830	3	2	1672	391	210	441	651	0.6781	0.9976	2.3939	2.301	0.39632
821	3	2	1695	377	220	462	682	0.6836	0.99757	2.4817	2.3819	0.39865
845	3	3	1524	523	230	483	713	0.6163	0.99646	1.7921	1.7143	0.26909
852	3	1	1516	526	240	504	744	0.6169	0.99883	1.7887	1.7621	0.62042
850	3	2	1520	523	250	525	775	0.6177	0.99765	1.7955	1.7426	0.37559
853	3	3	1574	465	260	546	806	0.6457	0.9965	2.0127	1.9152	0.27359
860	3	2	1562	471	270	567	837	0.6447	0.99768	1.9916	1.9268	0.3835
865	2	4	1593	434	280	588	868	0.6649	0.9954	2.1545	2.0085	0.2139
859	4	3	1647	385	290	609	899	0.6883	0.99652	2.4184	2.2788	0.27998
857	4	3	1667	367	300	630	930	0.6979	0.99651	2.5348	2.3819	0.28147
857	4	3	1667	367	300	630	930	0.6979	0.99651	2.5348	2.3819	0.28147
863	4	3	1679	349	310	651	961	0.7097	0.99654	2.6629	2.4947	0.28299
856	4	3	1698	337	320	672	992	0.7151	0.99651	2.7558	2.5761	0.284
844	5	3	1758	288	330	693	1023	0.7423	0.99646	3.2027	2.9625	0.28815
841	5	3	1768	281	340	714	1054	0.7462	0.99645	3.2803	3.0288	0.28876
845	5	3	1766	279	350	735	1085	0.7484	0.99646	3.3031	3.0482	0.28894
844	4	3	1767	280	360	756	1116	0.7482	0.99646	3.3031	3.0482	0.28894
844	4	3	1775	272	370	777	1147	0.7536	0.99646	3.3978	3.1287	0.28964
840	4	4	1782	268	380	798	1178	0.7554	0.99526	3.4348	3.0779	0.22212
844	4	4	1787	259	390	819	1209	0.7624	0.99528	3.5506	3.1706	0.22259
844	4	5	1784	261	400	840	1240	0.761	0.99411	3.5111	3.0581	0.18023
848	5	8	1786	251	500	1050	1550	0.7681	0.99065	3.5909	2.8902	0.11494
875	3	13	1748	259	600	1260	1860	0.7696	0.98536	3.4473	2.5013	0.07155
878	2	12	1788	218	700	1470	2170	0.7996	0.98652	4.0862	2.8902	0.07765
888	1	1	1880	128	800	1680	2480	0.8732	0.99888	7.2923	6.8696	0.84043
898	1	1	1886	112	900	1890	2790	0.8882	0.99889	8.3158	7.7705	0.85252
899	1	2	1882	114	948	1950	2898	0.8866	0.99778	8.1026	7.1278	0.44865

Appendix J

Raw results from the trigram-based spam filter test

S_S1	S_H1	H_S1	H_H1	U1	spam	ham	total	recall	precision	TCR	TCR/9	TCR/999
44	0	0	154	2700	10	21	31	0.016	1	0.3511	0.3511	0.35111
107	15	0	300	2476	20	42	62	0.0412	1	0.3806	0.3806	0.38057
233	20	0	378	2267	30	63	93	0.0925	1	0.4145	0.4145	0.41452
318	20	0	431	2129	40	84	124	0.1289	1	0.4411	0.4411	0.44114
371	15	0	492	2020	50	105	155	0.1542	1	0.4658	0.4658	0.46585
415	15	0	533	1935	60	126	186	0.1755	1	0.4862	0.4862	0.48615
448	13	0	625	1812	70	147	217	0.1971	1	0.5195	0.5195	0.51945
462	13	0	795	1628	80	168	248	0.2197	1	0.5777	0.5777	0.5777
489	12	1	764	1632	90	189	279	0.2293	0.99796	0.5763	0.5735	0.35868
506	10	1	807	1574	100	210	310	0.2421	0.99803	0.5981	0.5951	0.36702
506	10	1	807	1574	100	210	310	0.2421	0.99803	0.5981	0.5951	0.36702
514	10	1	818	1555	110	231	341	0.2472	0.99806	0.6054	0.6023	0.36973
518	10	1	822	1547	120	252	372	0.2496	0.99807	0.6085	0.6054	0.37089
538	10	1	860	1489	130	273	403	0.2641	0.99814	0.632	0.6286	0.3795
544	10	1	953	1390	140	294	434	0.2798	0.99817	0.6767	0.6728	0.39516
593	9	1	1041	1254	200	420	620	0.3195	0.99832	0.75	0.7453	0.4191
648	14	1	1270	965	300	630	930	0.3983	0.99846	0.9673	0.9595	0.47927
686	12	0	1430	770	400	840	1240	0.4673	1	1.2123	1.2123	1.21228
707	12	0	1487	692	500	1050	1550	0.5011	1	1.3466	1.3466	1.34659
735	5	0	1476	682	600	1260	1860	0.5169	1	1.3799	1.3799	1.37991
751	3	0	1558	586	700	1470	2170	0.5604	1	1.6095	1.6095	1.60951
767	1	1	1685	444	800	1680	2480	0.6328	0.9987	2.1256	2.0881	0.65651
773	1	2	1713	409	900	1890	2790	0.6534	0.99742	2.301	2.215	0.39369
773	0	2	1715	408	948	1950	2898	0.6545	0.99742	2.3122	2.2254	0.39401