University of Edinburgh
School of Informatics

The Identification of Unsolicited Electronic Mail

Undergraduate Dissertation
Artificial Intelligence and Computer Science

Kaiesh Vohra[1]

March 2, 2005

Abstract: Electronic mail is now the standard method of communication in corporations, institutions, and social networks. Due to its ubiquitous nature and negligible cost of use, it has fallen victim to advertisers seeking to peddle their products. This has resulted in an overwhelming amount of unsolicited commercial mail being indiscriminantly broadcasted to every address that can be found. This study provides a background into the methods that currently exist to tackle this problem, and analyses their purported effectiveness. These methods motivate the development and introduction of a classifier which aggregates the results from several known text classification techniques by use of an artificial neural network. The aggregation of results from multiple adaptive filters is shown to significantly outperform any individual method employed. Thereby presenting the case for further study into such multi-faceted, adaptive spam classification methods.

[1]Matriculation number: s0126253

Acknowledgements

My brother, for his unrelenting patience, advice, lifetime of guidance, support and love.

My parents, for their love, support, and everything.

My friends, for their support and care.

Thomas French, for his help in $L^{A}T_{E}X$ formatting.

Russell and Norvig, for "*The Bible*."

Steve Renals, for his ongoing help and advice.

Alan Bundy, for keeping me interested in artificial intelligence in the early days.

May the above never receive any spam after today.

Contents

1. Introduction

1.1 A background to electronic mail

Electronic mail is now the de facto method of communication in corporations, institutions, and social networks. E-mail is widely used due to the convenience of its near instant delivery and extremely low cost. The benefits that encourage the use of e-mail as a communication medium also allow for its simultaneous abuse and misuse. The low costs associated with sending e-mail, encourage certain entities to send out unsolicited commercial messages to any and every individual they can reach. The issue of unsolicited e-mail, also known as **spam**, has reached the point where internet service providers such as AOL were blocking 780 million spam messages every day by February 20th 2003[1]. This situation has prompted the research into methods that can be used to block spam with higher accuracy, and more efficiency.

The most pressing issue faced by methods countering unsolicited email, is the issue of a good message being falsely marked as spam. For a user, this type of scenario can be a serious problem. As such, methods employed against spam are required to not yield false positives, and block unsolicited email, whilst still allowing legitimate, new found associates to communicate. Unfortunately, as yet, there is no panacea.

1.2 An introduction to this investigation

In this paper, the accurate categorisation of electronic mail in the English language, through analysis of message body content[2] is explored. The aggregation of results from multiple text analysis methods, using a single layer artificial neural net[1, 2], with a sigmoidal neuron, is explored. The perceptron is shown to enhance the accuracy in categorisation of solicited and unsolicited e-mail, and significantly outperform any individual method used.

Throughout this paper, a single test is referred to as a filter; whereas a classifier is either one filter, or a combination of multiple filters. Additionally unsolicited e-mail is referred to as spam, and *good* e-mail is referred to as ham. The investigation opens in chapter 2 with a study of existing techniques used to address spam, from proposals to modify existing architecture to well known classifiers. Chapter 3 is motivated by this to present the analysis of the e-mail message as a specific text classification problem. Thereby stimulating the argument behind using multiple well known text classification techniques, along with possible methods for interpreting the combination of results the multiple tests yield. In chapter 4 the implementation of these techniques is presented, with chapter 5 following up with the experimental setup and an analysis of the results obtained. Additionally, chapter 5 also presents the comparison of the results yielded by the studied classifier against many of the methods discussed in chapter 2. The investigation is then concluded in chapter 6 with an analysis of the potential future of classifiers and their role in the addressing the spam problem.

[1] http://media.aoltimewarner.com/media/cb_press_view.cfm?release_num=55253034 (last verified 14th Dec 2004)

[2] Message body content is defined as the text written by the user, and not the descriptors added by computer systems.

1.3 What is spam?

The Oxford English Dictionary defines spam to be "*irrelevant or inappropriate messages sent on the Internet to a large number of users*."[3] Although this definition can be accepted as representative of what spam is in the general sense, the words "*irrelevant*" and "*inappropriate*" are highly subjective. This results in problems when filters need to be devised to categorise the unwanted messages, as different individuals are likely to interpret a given message differently to another. Therefore, in this study, spam is defined as unsolicited electronic mail that has been indiscriminately broadcast to a large number of users.

1.4 The importance of the impact of spam

The distributors of unsolicited email presumably find that the benefits from it outweigh the risks of being penalised. The costs associated with sending an individual email are based on the cost of the user's connection to the internet, the numbers of emails that can be sent per second, and the cost of running the machines that send email. This implies that the more messages transmitted per second, the lower the cost of each message.

While the cost of transmitting an email may be low, profit must still be made to make the business viable. The estimated cost per email for the unsophisticated spam distributor[4] is roughly 0.017cents[3][5]. This figure is derived from the fact that a spam sender can pay US$50,000.00/month and transmit approximately 10 million emails per day. It is estimated that on average one email out of 30,000 results in a sale, which means that the profit made by such distributors is in the range of US$400,000.00 to US$700,000.00 per month. It is this monetary gain which is the drive behind sending unsolicited mail, and it is also the drive behind more malicious techniques used by the unscrupulous.

As legislation[4] is put in place to tackle the problem that spam poses, the distributors find ways to transmit messages without being traced. Such methods involve making use of recent, or even revolutionary, communication models. Some of these models make use of, and have detrimental effects on, the average home user's personal computer[6]. These models are manipulated by spam distributors forging alliances with virus writers[7] in order to avoid being found, and reduce their overhead costs for bandwidth. As home user computers are hijacked, and as the spam distributor requires methods of controlling what is sent, the virus is typically a Trojan which yields control of the machine to the spam distributor. Peer-to-peer[8], and "zombie"[5, 6, 7] based spam networks prove to be incredibly cost-effective for the malicious distributors as they do not have to pay for large bandwidth or power costs, yet are still able to send millions of emails everyday.

The issue which arises from large "zombie", and peer-to-peer spam networks, is not only the fact that the controllers are harder to find, but global network congestion is made worse. Furthermore people risk losing private data to the distributors and virus writers, which can have even more serious consequences, such as identity theft[9]. The examples presented are only some of the issues that arise

[3] The Shorter Oxford English Dictionary: Fifth edition, William Little, Lesley Brown, and William Trumble. Oxford University Press, ISBN: 0198605757

[4] An unsophisticated spam distributor is one which incurs bills for the equipment and bandwidth that they use.

[5] http://edition.cnn.com/2004/TECH/internet/11/14/inside.spamming.ap/ (last verified 14th Dec 2004)

[6] http://news.zdnet.co.uk/internet/security/0,39020375,39118252,00.htm (last verified 14th Dec 2004)

[7] http://news.zdnet.co.uk/internet/security/0,39020375,39180203,00.htm (last verified 14th Dec 2004)

[8] http://news.bbc.co.uk/1/hi/technology/3409187.stm (last verified 14th Dec 2004)

[9] http://www.sanluisobispo.com/mld/sanluisobispo/news/9608107.htm (last verified 14th Dec 2004)

out of today’s situation with spam, which make it increasingly important to reduce the incentives behind the practice.

2. Survey of existing anti-spam techniques

As unsolicited e-mail has become such a widespread problem, many corporations and individuals have begun to investigate methodologies to reduce its impact and the costs it incurs. The social and technical problems of unsolicited mail occurs on several different levels, as illustrated in figure 2.1. The multitude of methods that have been developed employ a variety of techniques, addressing different problem areas, and have had varying levels of success. Some of those techniques are presented here along with their benefits and disadvantages.

This chapter opens by analysing techniques proposed to modify the backend mail delivery systems to reduce the flow of spam, followed by applications used at the client side which attempt to stop spam from being displayed to the user. The chapter concludes with a summary of the capabilities, and effectiveness, of all the methods in today's architecture.

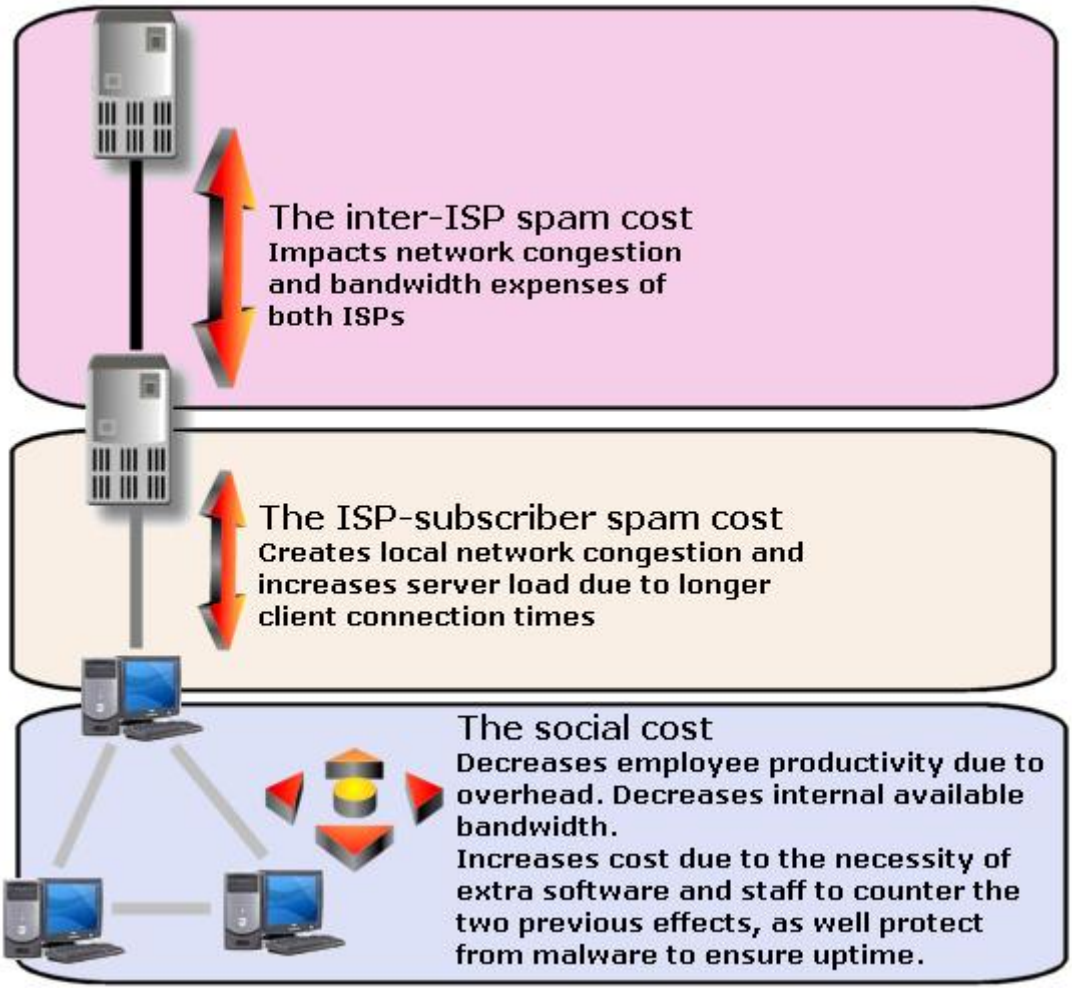

Figure 2.1: Examples of the different levels at where problems can occur

2.1 Mail delivery techniques

Anti-spam techniques which relate to the way mail is delivered between two servers are based on changes to the underlying protocols of mail handling. The implication of adopting any one of these techniques is that globally Mail Transfer Agents[1] (MTAs)[8] need to adopt and implement the same modification. If the modification is not adopted, either two MTAs will not be able to communicate, resulting in the impossibility to transfer mail, or the modification will require to be backward compatible[9, 10]. A backward compatible modification means that communication can still take place with a server that has not implemented the change. Typically, the modifications are engineered to be backward compatible, but to include '*negative ratings*' for servers that have not implemented the change.

These types of methods attempt to directly impact the expenses incurred by Internet Service Providers (ISPs), and bandwidth congestion problems, by prohibiting spam from being relayed between MTAs. In figure 2.1, the primary area tackled is "The inter-ISP spam cost." There are two well known types of proposals: the payment based models, and the server authentication models. Both of them are explored here.

2.1.1 Payment based models

There exist proposals to alter MTAs in a way such that the "physical world" method of paying for sending messages is used as a reference. The proposals vary from using monetary based "micro-payments" for digital stamps[11, 12], to virtual currencies used for requesting resources[13, 14], to "proof-of-work" computational puzzles[15, 16]. The Lightweight Currency Protocol, a virtual currency model, and Adam Back's Hashcash, a computational puzzle model, are both explored here as examples.

2.1.1.1 Lightweight Currency Protocol (LCP)

The LCP[14] is based on virtual tokens which represent a type of money. Each server is said to "mint" a certain number of tokens, which are digitally signed[17, 18], and represent access to their resource. Each server is also required to be certified by an authority which the target server trusts. The underlying concept is that a virtual economy will be constructed, and when one server wishes to email another server it pays in a currency which the target server deems valuable. This means that servers which accept mail for a large number of users will have tokens that are extremely valuable to most other servers. Whereas servers which only send out large amounts of mail, will have tokens that are undesirable.

The LCP implementation is necessarily based purely at the level of the MTA, and requires peering MTAs to co-operate within the virtual economy. The motivation is based upon the fact that spammers will have servers which send much more mail than they receive, and so the tokens from spamming servers will very quickly be deemed worthless. The resulting desired effect is that no server will communicate with spam servers as they will not possess useful currency. This means that either the spam servers will need to purchase tokens using real money, or establish a new server with a new trusted certificate; both circumstances achieve the monetary based disincentive for spammers.

[1] Commonly referred to as "Mail servers"

It is theorised that this approach will not stop spam entirely, but will create an overhead larger than one which exists today, and thus will act as a disincentive to send as much spam as is currently done. Ultimately, the goal is that the cost of sending spam will have increased to levels which a spammer finds unattractive and non-profitable. Unfortunately, if there are increased costs for spammers to deploy mail servers, it means there will also be a higher overhead for legitimate users. Additionally, there exist problems for mailing list operators and e-commerce sites, as they need to construe a method to receive e-mails, otherwise they would not have sufficient currency to continue sending messages.

The biggest disadvantages to the LCP proposal are that every MTA administrator needs to accept being signed by "trusted authorities", which usually involves payment, and everyone needs to deploy it at the same time or the implementation will not work. Furthermore, the deployment requires a migration from an existing free, unsupervised structure to a paid, supervised and monitored architecture. This migration may have social acceptance issues, and thus receive widespread resistance.

2.1.1.2 Adam Back's Hashcash

Hashcash[15] is based on the principle that it is relatively expensive for a client to compute a Hashcash tag, also referred to as a stamp, but extremely cheap for a server to validate the stamp. The model does not use real money, but instead is an example of a CPU cycle based payment method for resource utilisation.It provides a potential sender with a computer-based mathematical puzzle to solve. The solution to the puzzle is specific to the target recipient of a message, as well as to the time of sending and is intended to be computed without user interaction. The result is that a sender is forced to spend several seconds waiting for their computer to solve a puzzle before an e-mail is sent.

Hashcash requires a sender to expend a particular amount of computing power in generating an acceptable stamp. The receiving mail server, or end user, can specify how complex the puzzle is, in order to reduce, or increase, the amount of processing time used, before the stamp is considered acceptable. The concept underlying the method is that a sender proves that they have carried out work in order to communicate with the receiver. The implementation of Hashcash is different from most others, as the deployment location is not specific to either a server or an end-user machine, it can be implemented at either level. This flexibly in deployment allows for users to reap the benefits of Hashcash even if their MTAs do not.

Hashcash is a "proof-of-work", denial-of-service countermeasure tool, that aims to reduce the amount of spam sent rather than completely stop it. In this aim, it can only be successful if it is widely accepted, and as the method causes e-mail to be delayed from sending by almost half a minute, businesses and people may not appreciate the perceived loss in time. Hashcash also does not resolve the fact that spam bot networks[2] can be used to generate valid Hashcash based e-mails, but a spam bot which implements Hashcash would visibly slow down a hijacked machine as well as reduce the number of e-mails that it can send per minute.

Unfortunately Hashcash has greater impact on older and slower, or portable, computers than it does on newer computers; the net result means that users of older hardware, or devices like PDAs, will waste more time on puzzle solving than users of new hardware. In this context however, puzzles based on memory-bounds prove effective in having smaller impact on devices of varying calibre[19].

Thus, while Hashcash will not stop unsolicited mail from reaching a users inbox, it has the potential of reducing the number of unsolicited e-mails received. Additionally, puzzle solving mechanisms have the benefit of not requiring everyone to deploy it at the same time, therefore working with gradual

[2] See section 1.4

deployment. However, to achieve the desired potential it must be socially accepted that e-mail can no longer be "instantly" delivered, and such acceptance is likely to meet resistance from the public as the concept behind legitimate users wasting CPU cycles on "*pointless puzzles*" is not extremely appealing.

2.1.2 Server authentication models

Models based on server authentication propose to validate the server being communicated with before a message is allowed to be relayed. The validation is meant to take place on a peer-to-peer level rather than through use of a "common trusted body." The models work on the concept that if a server requires to be authenticated before it is allowed to communicate, then some knowledge of the server must exist. This knowledge about the server allows levels of trust to be applied, which can determine how the messages from the domain are handled. The knowledge also allows for mail to be traced back to its origins, which is useful in legal enforcement of local legislature, if there exists any. There are two well known proposals which are discussed here, Yahoo!'s Domain Keys, and the Sender Policy Framework.

2.1.2.1 Yahoo!'s Domain Keys

The Domain Key solution proposed by Yahoo (YDK) is a cryptographic approach to mail validation[9, 10, 20]. Cryptographic methods of mail verification involve applying a digital signature[17, 18] to a particular portion of the electronic message so that it can be verified by a recipient for legitimacy. YDK requires modifications to be made to the MTAs as well as its Domain Name Service[3] (DNS) entries.

At the time of writing, YDK is still in development and the implementation is not tremendously widespread, and as such its effectiveness once deployed cannot be accurately predicted. However, what is interesting to note from the specification is that the message content is signed along with selected headers. So the signature will guarantee that the mail server being used is authorised to send the message with the supplied `From` field and the originating Internet Protocol (IP) address. Unfortunately, this does not stop spammers repeatedly setting up and taking down their own mail servers which support YDK, nor does it offer support for protection from spam bot networks[4].

As it stands now, YDK proposes to use a peer-based ranking system[20, 10] to protect users from spam sending mail servers, and to make spam senders traceable. Unfortunately, this may not prove entirely effective as the machines of unknowing home users are already being used to send spam[5, 6, 7] rather than dedicated spam relays. Furthermore, not only could a YDK based mail server be temporarily raised on a hijacked machine, but the compromised host could use the mail server belonging to the ISP that the host is subscribed to[21].

Thus, as YDK is still undergoing development, it is difficult to make any conclusive decisions about how effective its implementation will be. However, traceable, authenticated, and verified e-mails have the possibility of dramatically reducing the amount of spoofed address based e-mails that are received. The boon but also potential problems inherent within the YDK design are that the servers

[3] DNS is an internet service protocol which translates human readable internet addresses into machine understandable addresses.

[4] See section 1.4

are required to do more work while the role of the clients has not changed, and that an Internet-wide software upgrade will need to be carried out on all MTAs.

2.1.2.2 Meng Wong's Sender Policy Framework

The Sender Policy Framework[10] (SPF) proposes to authenticate the sender of an e-mail by comparing the domain the message alleges to be from, against the IP address of the connecting machine. In the return-path of an e-mail there exists an address to which all replies to the message should be sent. An SPF enabled MTA analyses the SPF record associated with the domain in the return path of the proposed e-mail to send. The MTA then performs a reverse DNS query on the IP address of the connecting machine and verifies the domain yielded against the SPF record. If the SPF record does not contain the looked up domain then the message is deemed to be a "forgery", and thus an invalid message. The status determined by the SPF extension to the MTA is inserted in the headers of the e-mail prior to the message being deployed in the target users mailbox. This is illustrated in figure 2.2.

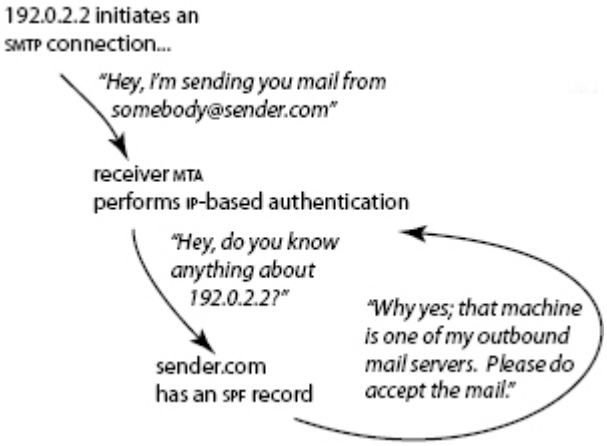

Figure 2.2: Sender Policy Framework method, *image from [10]*

The SPF has a functional implementation today, and has been shown to accurately validate e-mails originating from valid domains. To this effect, SPF claims to have reduced the impact of the particular type of spam known as "phishing"[5][22]. However, as SPF validates the return path of the e-mail and not the `From` field, it only guarantees that the message is coming from a registered mail server with an SPF record, and if it bounces that it will bounce back to the correct domain. As the `From` field remains unprotected, phishers are still using it to socially engineer their attacks on unsuspecting users[23]. Additionally, SPF fails to function properly when mail forwarding[8, 10] is used, which is another significant caveat in its design.

Overall, SPF appears to be more of a network administrator's tool rather than one for the end user. This is due to the fact that the user is not explicitly notified about the return paths of e-mails or their validity. Additionally, SPF technically only has the capability to prevent one class of spam, which is phishing. Furthermore, for a user to yield any direct benefit from the information , changes would be required within the Mail User Agent (MUA), or *user's mail client.* This modification to the MUA is proposed in Microsoft's Sender ID, which is further explored in section 2.2.1.

[5]"Phishing" is where an e-mail alleges to be from an organisation it is not actually from, and requests for personal information. The most common phishing e-mails masquerade as e-mails from large banking corporations requesting for account information. "Phishing" has also been referred to as "Web spoofing".

2.2 Mail filtering techniques

Mail filtering techniques attempt to combat the expenses incurred by corporations, as well as the social problem spam presents through time wasted by people, by prohibiting the undesired messages from displaying in the MUA. In figure 2.1, the area tackled by them is described as “The social cost.” There are a plethora of options available as to which filter to use today, but they all make use of some well known basic techniques. The well known techniques that are discussed in this section are Microsoft’s Sender ID, address black and white listing, challenge-response systems, naïve Bayesian filtering and rule based filtering.

2.2.1 Microsoft’s Sender ID

Microsoft’s Sender ID builds on Meng Wong’s Sender Policy Framework[10] (SPF) (section 2.1.2.2). It essentially takes the implementation of SPF and embeds it into Mail User Agents (MUAs). This results in the ability of the mail client to automatically discard mail that falsely claims to be sent from a particular address. Sender ID enhances SPF by attempting to validate the `Mail From` field, which is the field the user sees, rather than just the `Return Path`, using the same method as SPF.

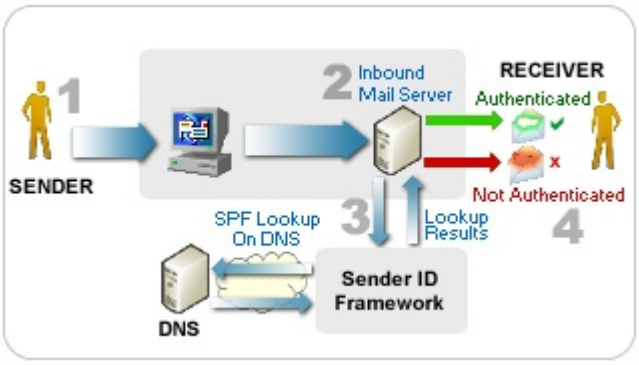

Figure 2.3: Outline of the Microsoft Sender ID method

Figure 2.3[6] outlines the procedure in which Sender ID verification takes place, and includes a *cloud* in stage 3, which is representative of a server having implemented SPF. The implications of this are that for Sender ID to function, it falls victim to the same problem as SPF, in the sense that it must be adopted by most mail-servers for it to be effective. Sender ID also suffers from an additional social problem because it is a patented protocol by Microsoft[7], which means that it violates open source[8] ethics. Consequently, it will fail to be implemented by proponents of open source, which may prove a significant problem in its adoption. Furthermore, SenderID will not be able to reduce spam, as much like Sender ID it will only be able to stop one class of unsolicited e-mail, known as “phishing.”

In summary, while Sender ID is a good initiative to stop spoofed and fraudulent e-mails, and has a fairly simple implementation for administrators and developers, its deployment is likely to be hindered by corporate agenda. Microsoft’s attempt to deploy the protocol while still maintain the right to license deployment is conceivably what has choked the speed at which it could have been adopted, especially considering the large adoption of the SPF protocol[10].

[6] Image extracted from http://www.microsoft.com/mscorp/safety/technologies/senderid/technology.mspx (last verified 27 Feb 2005)

[7] http://www.microsoft.com (last verified 27 Feb 2005)

[8] http://www.opensource.org (last verified 27 Feb 2005)

2.2.2 Blacklists and whitelists

Blacklisting is the storing of an internet address that is thought to condone the relaying of spam messages. There are several publicly available blacklists[9] which can be queried by MTAs[10] or user client applications[11]. The publicly available blacklists are regularly updated by the internet community as well as proprietary testing done by the organisations. If a message is found to have traversed through, or be from, an address that is blacklisted then its legitimacy is considered questionable.

Whitelisting is the storing of an address so that any message found to match that address is considered legitimate without question, unlike blacklists which typically cover whole domains, whitelists typically cover individual addresses. Many MTA[12] and client-side filtering tools[13]make use of whitelisting to allow for users to bypass testing on addresses they expect to receive mail from.

Blacklists and whitelists alone do not perform very well[24], not only because e-mail headers can be forged which can result in a false negative, but also because legitimate users can be wrongly blacklisted resulting in a false positive. However, when whitelists are combined with other features like YDK[14] or SPF[15] the problems with false negatives can be dramatically reduced. Unfortunately, using blacklists, there is no completely reliable way to ensure that email from non-whitelisted senders is legitimate email, and so other filters need to be used to aid the decision.

2.2.3 Challenge-Response systems

Challenge-Response systems, in the context of e-mail, are tests designed to distinguish legitimate correspondents from spammers by the address of the sender. There are two types of well known challenge-response tests, those which verify the entity sending the message actually exists and intended to send you an email, and those which verify that the sender is actually human. Challenge-Response systems typically reside between the user's MTA and MUA. Both systems typically make use of whitelists and blacklists to expedite the process of receiving mail and reduce overhead for both correspondents. Any message received from an address not in the white or blacklist is held in storage with an "unlock key" and a challenge is issued to the sender which contains a link which is connected to the "unlock key" in some way, as is shown in figure 2.4

Sender existence and intention verification systems[16] typically respond to the unknown sender with an e-mail message. The message contains instructions on how to reply to the verification message if the sender wants the message to be fully delivered. The instructions normally request the user to "*hit the reply button*" and send the message back, as the address or subject of the message contains the unlock key.

[9]Examples: `http://www.dnsbl.org`, `http://dsbl.org` and `http://bl.spamcop.net` (last verified on 14th Feb 2005)

[10]Example: SpamAssassin (`http://www.spamassassin.org`) with details at `http://wiki.apache.org/spamassassin/dnsBlocklists` (last verified on 14th Feb 2005)

[11]Examples: SpamPal at `www.spampal.org` and MailWasher at `http://www.mailwasher.net` (last verified on 14th Feb 2005)

[12]Example: SpamAssassin (`http://www.spamassassin.org`) with details at `http://wiki.apache.org/spamassassin/AutoWhiteList` (last verified on 14th Feb 2005)

[13]Example: Thunderbird MUA at `http://www.mozilla.org/products/thunderbird` (last verified on 14th Feb 2005)

[14]See section 2.1.2.1

[15]See section 2.1.2.2

[16]Example: Opensource, "Active Spam Killer (ASK)" available at `http://www.paganini.net/ask` (last verified on 14th Feb 2005)

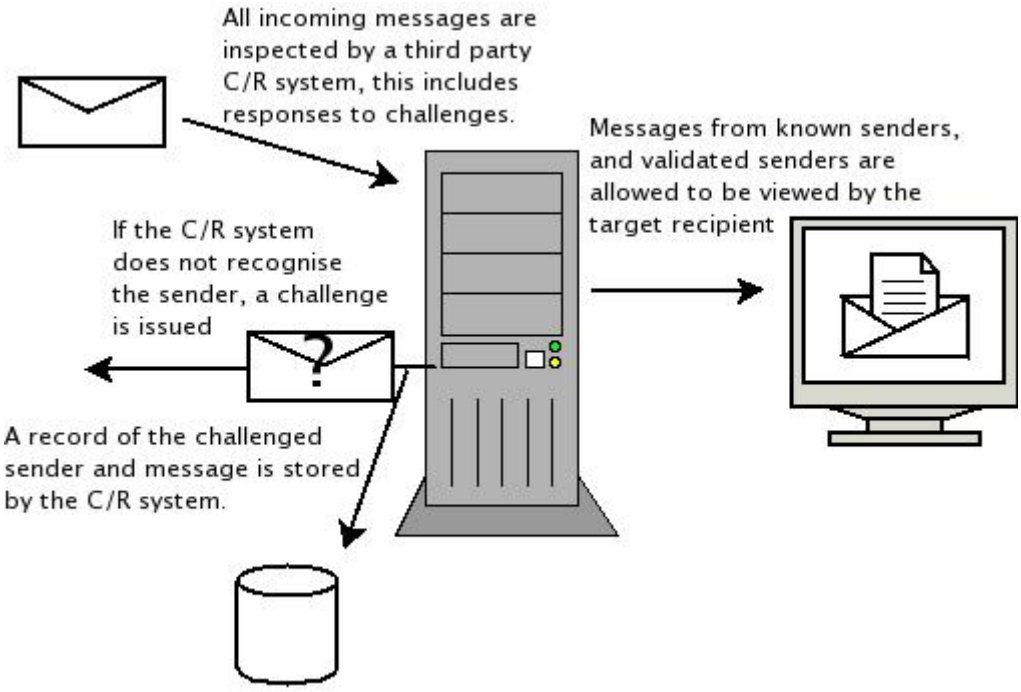

Figure 2.4: Outline of the Challenge-Response concept

Human verification systems[25][17] typically respond to the unknown sender with an e-mail message which points the receiver to a website[18] and asks them to complete a test which are easy for humans but difficult for computers[19]. The website then performs some action to allow the suspended message to be delivered.

The concept behind such challenge-response systems is appealing and they provide low overhead for correspondents but they fail to address the issues behind false header information, mailing lists, and correspondents using a different, proprietary Challenge-Response system. False header information, commonly used in spam, can result in uninvolved people being *spammed* by confirmation messages; or in further load on mail servers being forced to deliver an error message if the address does not exist. Also, a challenge-response subscriber who does not whitelist their mailing list is likely to spam the subscribers of the mailing list with a challenge-response email every time someone sends a message to the mailing list. Additionally, a legitimate pair of correspondents that have not yet whitelisted each other can enter challenge-response deadlock if one user's system challenge is caught and challenged by the other user's system. This susceptibility to deadlock by challenge-response systems make them unsuitable for global deployment as it can cause an exceptionally large number of false positives.

Furthermore, systems which use websites as part of the verification process fail to address privacy concerns as the correspondence of two legitimate users is monitored by a third party agency which some users are uncomfortable with. Another issue, experienced primarily with human verification systems, is that they do not take into account visually impaired users, or if they do, it typically involves a method to circumvent the system[20], which would logically also be available to the spammers.

[17]Example: Carnegie Mellon Research, "Captcha Project" at http://www.captcha.net (last verified on 14th Feb 2005)

[18]Example: Commercial provider, "MailBlocks" at http://www.mailblocks.com (last verified on 14th Feb 2005)

[19]Research into this area by the Carnegie Mellon CAPTCHA Research Team has shown that advances in machine vision reduce the difficulty of these puzzles for computers, whereby the accuracy now reached by a machine is roughly 92%, with very little overhead. The details of this research is beyond the scope of this paper, but are available in [26]

[20]Example: The commercial provider "MailBlocks" (http://www.mailblocks.com) provides information on the website

Overall, challenge-response systems tend to create more mail than they stop. This does not address the economic issue of spam, nor does it seem to benefit the wider community as a whole.

2.2.4 Checksum based filtering (CBF)

Checksum based filtering techniques are based on the content of message bodies; the message is represented as a long checksum which acts a signature for the message[27, 24]. The signatures are resilient to minor changes like personalisations and a few random words at the end of the message[27]. They are computed for each message and compared against a peer-to-peer network of servers[21] to report that the message has been seen, and question how many others have seen it too. Additionally, *honeypot addresses*[28][22] are used to help build the signature database and expedite detection. The resulting effect is that an incredibly large signature database exists, which reflects a large portion of known spam.

The CBF technique which is currently widely deployed is an algorithm developed by Rhyolite Software[23], known as the `Distributed Checksum Clearinghouse`, and is unique as there exists support for the algorithm to be deployed on both MTAs and within MUAs. The ability to have the software deployed on the client-side and server-side means that the problem has the ability to address the "inter-ISP spam cost" as well as "the social cost". The method has the ability to stop a large portion of spam that is simply copied to any email address that a spammer can find, and has proven to be quite effective in doing so.

On the other hand however, the method has several caveats. CBF relies on the signature algorithm being used to be robust enough to support random modifications to messages by spammers, but still maintaining a large enough space for individual user e-mail messages to be received and sent without hindrance. In this scenario, the weak point is the signature algorithm, and if methods to subvert the algorithm are found then the system fails. It is known that spammers already randomly add extracts from various news sources and books, to their messages in order to bypass signature checks. Furthermore, if honeypot addresses are avoided, through use of honeypot detection techniques[29], then a fair number of users, which are part of the network, still need to receive and flag the message as spam before other users will not be hindered by it.

Therefore, while CBF techniques have the ability to, and actively do, counter a large portion of spam, they are currently unable to achieve average "detection rates higher than 70%, but still produce false positives with the severity of around 2%"[24]. Unfortunately, even if the signature algorithm were to be robust enough to not produce any false positives, but still catch variations of the same message, it is possible that the detection rate will not increase. However, the method has proven to provide MTAs with the ability to reduce the number of spam messages they relay, and thus reduce their bandwidth costs and impact on network congestion.

---

for visually impaired users. The instructions tell the users to forward the email received to `support@mailblocks.com`

[21] http://www.rhyolite.com/anti-spam/dcc/ (last verified 26 Feb 2005)

[22] Honeypot addresses are e-mail addresses that have no human behind them, are not distributed to people, and set up exclusively to receive spam and no real e-mail. As a result, any messages that arrive at those address are known a priori to be spam.

[23] http://www.rhyolite.com (last verified 26 Feb 2005)

2.2.5 Naïve Bayesian techniques

Naïve Bayesian filtering techniques are statistical filters that use probabilistic techniques based upon a variation of the Bayesian probability rule, to determine if a combination of words in a message define it as spam or ham. The filter is typically tailored for use by one person, as the filter requires to learn which words occur, and where they occur, in a user's ham as well as spam. The learning process is carried out with the aid of the user using the filter, by their participation in manually classifying messages. The training process continues through the life of the filter by the user flagging any wrongly marked message with the appropriate classification. The filter is widely deployed in many MUAs[24] today and has experienced widespread success[30, 31, 24].

$$\forall \text{ word } \varepsilon \text{ email } \exists\ P(spam) = \frac{\frac{|word \cap db_{spam}|}{|db_{spam}|}}{\frac{|word \cap db_{spam}|}{|db_{spam}|} + \frac{|word \cap db_{ham}|}{|db_{ham}|}},\ P(ham) = \frac{\frac{|word \cap db_{ham}|}{|db_{ham}|}}{\frac{|word \cap db_{ham}|}{|db_{ham}|} + \frac{|word \cap db_{spam}|}{|db_{spam}|}}$$

Figure 2.5: Naïve Bayesian classification algorithm

The algorithm implemented by the Naïve Bayesian filters is extremely simple, as shown in figure 2.5, and thus extremely computationally efficient. Proposals like P. Graham's to increase the accuracy of the filter by using only the 15 most significant attributes in each message[30], further decrease the computation time taken, and increase classification performance. As Bayes classifiers require to learn profiles, they tend to be deployed in MUAs so that they are more effective. There are implementations of Bayes classifiers at corporate server levels, but accurate statistics on their performance is not well known as they are typically deployed in filters like SpamAssassin[32] which use multiple other methods against spam (discussed in section 2.2.6), however it has been shown that using naïve Bayes alone may not be the optimal solution for server-side mail filtering[24].

Naïve Bayesian classifiers have had the highest success at the client side with upwards of 99% accuracy and 0.03% false positives[30]. However, spammers are slowly finding ways to circumvent these classifiers[29], and one of the mechanisms is to attach large amounts of text from legitimate e-mails. Additionally, due to the lack of the number of messages that are currently getting through Bayesian filters, spammers are sending more e-mails in the hope that one does. Furthermore, while naïve Bayes systems do not use much processor power in their computations, the longer a user uses one, the more intensive on disk space usage, and file I/O load the filter becomes. The increased load on disk space and I/O is due to the size of the "dictionary" that the filter generates with time.

In summary, the naïve Bayesian classifier is by far the best client-side classifier currently available, and addresses "the social cost" exceedingly well, with what has shown to be consistency over time. The problems with the naïve Bayes method are that it does not address the "inter-ISP cost", and has proven to be quite a personal mail filter, which means that for everyone to reap the benefits, everyone that reads e-mail needs to install it. Additionally each user must constantly manually, and consciously, train the filter with examples of spam and ham, otherwise the performance of the filter degrades. The installation of the Bayesian classifier has not met with social resistance however, as people do not perceive an immediate difference in performance as they would with Hashcash (see section 2.1.1.2) for example. Also, people do not seem to mind having to wait for the training period of the filter before the amount of spam they see is reduced. Fortunately, most implementations of naïve Bayes filters do not require much user-competency with technical matters.

[24] http://www.eudora.com's Eudora, and http://www.mozilla.org's Thunderbird are examples of commercial mail clients which implement Naï Bayesian filters

2.2.6 SpamAssassin

SpamAssassin[25] is a well known rule-based filter that is typically deployed on corporate or residential mail-servers. The filter compares an entire message against an exceedingly large ruleset[32], which is manually updated by the administrator of the server running the filter. SpamAssassin now uses a single layer neural net[26], or perceptron, to tailor the impact of each test to the community using the mail-server, therefore minimising the time required by the administrator.

SpamAssassin is developed to be used "out of the box" with no training necessary, which is a consequence of the use of rules, rather than statistical techniques like naïve Bayes. Rule based filters also support the addition of custom rules, so that if an administrator notices a particular type of spam arriving at their mailserver, it can be choked at the server with the integration of a necessary rule. An example of the type of rules that can be used in SpamAssassin are shown in figure 2.6, and illustrates how varied the tests can be.

Area tested	**Description of test**	**Default scores** *(local, net, with Bayes, with Bayes+net)*
body	Message only has text/html MIME parts	1.204 , 1.158, 1.156, 0.177
rawbody	Quoted-printable line longer than 76 chars	0.000, 0.000, 0.105, 0.039
rawbody	MIME filename does not match content	0.100, 0.100, 0.100, 0.100
body	HTML and text parts are different	1.837, 1.505, 1.823, 0.066
body	Character set indicates a foreign language	3.200, 3.200, 3.200, 3.200

Figure 2.6: Sample of some rules in SpamAssassin

SpamAssassin has had highly varied levels of success between deployments[33, 24], and it has been suggested that it is the wrong approach to take to spam[33] as it is a *non-automatic reactive* system that requires manual updating. As spam evolves, the administrator of SpamAssassin is typically required to modify the rules slightly, or download larger rulesets. Additionally, SpamAssassin necessarily runs on a mail-server which distributes mail to particular portion of users, such as a company, or possibly even a corporate division; which means it is not suitable for the average home-user and thus only tackles a small part of "the social cost." Furthermore, SpamAssassin typically requires a dedicated machine to act as a mail-server with it running, as it can be quite processor intensive; and it requires a system administrator to be comfortable in managing the software. Despite these caveats however, it is known to be well deployed, and trusted, across many corporate, and even personal, mail-servers.

In summary, SpamAssassin is an advanced user's tool to filter spam out from a larger population of users as opposed to one person, and requires a fair level of management. It has the ability to reduce the amount of spam a large number of users receive, but to do so requires experienced administration and testing or the filter may not perform as desired. It is not viable for an inexperienced user to deploy or manage, and it requires a dedicated machine to function on. Unfortunately it does not automatically adapt to trends in spam, but out of the solutions available it is the most effective method at protecting large pools of users. It has been recommended that the mail server a user collects mail from should run SpamAssassin, while the client runs a statistical naïve Bayes filter themselves, that way the least of amount of spam should penetrate a user's inbox[24].

[25] http://www.spamassassin.org (last verified 26 Feb 2005)

[26] Discussed later in section 3.3.1

2.3 IBM's SpamGuru

IBM's SpamGuru[34] is a classifier, and also a modification to architecture. It uses multiple filters and modifies the way mail is handled. Unlike traditional proposals to architectural changes (see section 2.1), SpamGuru provides support for the change in architecture, with payment based models as a preference. It also provisions for clients and servers that have not yet implemented the changes by filtering the messages using several sophisticated algorithms. This section will provide a brief overview of the techniques employed by SpamGuru and its purported effectiveness, but the interested reader should consult [34] for more details. Once a message is received, if SpamGuru is unable

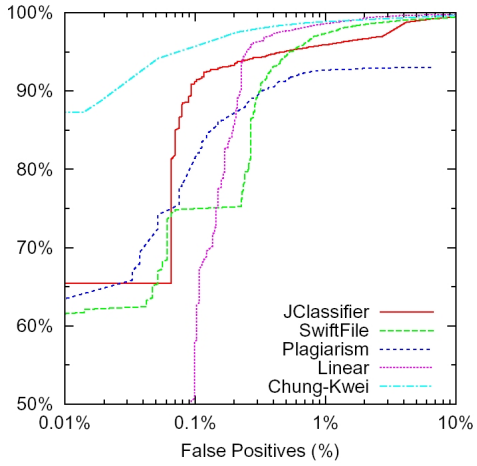

Figure 2.7: The textual analysis filters used in IBM's SpamGuru

to detect valid payment for the message, then it proceeds to analyse the message's DNS stamps and their validity. This is followed by user, and corporate, whitelists and blacklists, followed by a rendering engine. The rendering engine parses the HTML in the message to determine how it is actually displayed in the MUA. The rendering engine, for fear of loss in information, does not only pass on the rendered version of the message, instead it passes the raw version and the rendered version to the remaining filters. The filters after the renderer are shown in figure 2.7, and consist of the following:

a "plagiarism detector" which uses `k-grams`[27] to detect for message similarity

JClassifier which is based on Graham's naïve Bayes classifier[30][28]

a linear discriminant document classifier based on the findings in [35] which discusses logistic regression methods and support vector machines[29] in text classification

Swiftfile which is another variant on Bayesian classification, further discussed in [36]

Chung-Kwei a proprietary, biologically inspired, algorithm based on the Teiresias pattern discovery technique, more information is available in [37]

Results aggregator which weights the results of the raw outputs from the individual classifiers, and applies a threshold to the combined sum of the weighted module outputs, somewhat like a perceptron[30]

[27] Discussed later in section 3.2.1.3
[28] See section 2.2.5
[29] Discussed later in section 3.3
[30] Discussed later in section 3.3.1

As SpamGuru is an active research project at IBM, there is not much data available to evaluate it. However, the data released by IBM indicates that combining several test results using a simple linear aggregator yields a remarkable performance increase. This is even with several of their methods not exhibiting exceptional abilities at capturing spam. IBM's tests also show that SpamGuru's performance, on a massive test corpus produces results where their combined classifier achieves spam accuracy of over 95% with only 0.01% false positives. The results of this test are remarkable as it is against possibly the largest test corpus out of most other studies.

Overall, SpamGuru's implementation appears to attempt to address every cost factor specified in figure 2.1, allowing for phased deployment and evolution of methods against spammer techniques. The data released by IBM suggests that the concept behind aggregating multiple filters is successful in increasing the overall effectiveness in identifying spam. Unfortunately as SpamGuru is a private commercial project with proprietary methods, there is not much data available for analysis and so a critical evaluation of their proposed classifier is not possible.

2.4 Summary

From this survey it is possible to see that there are several approaches that can be taken to addressing the problems spam poses. Each proposal is not without it's caveats, and more often than not only address one of the problem areas depicted in figure 2.1.

The alterations to mail delivery techniques through establishing new protocols, may not necessarily face protocol issues, but rather social acceptance issues. People may be reluctant to begin paying for e-mail, or wait longer for it to be delivered, because they are familiar with a particular quality of service. Furthermore, these types of solutions generally only work when the majority of MTAs support the same protocol.

The mail-filtering techniques presented have generally had fewer social acceptance issues, as they do not visibly impact the user experience of the medium. Unfortunately the mail-filtering techniques fail to address much more than "the social-cost". This results in spammers still being able to congest global networks without paying for it; and with the confidence that at least a percentage of their messages will penetrate some filters.

It has been shown that the filters generally tend to function independently of each other; and those that do make use of other methods, with the exception of IBM's SpamGuru, attempt to extend the functionality of them. Microsoft's Sender ID with SPF, and Challenge-Response with black and whitelists, are examples of such methods. IBM's SpamGuru is the exception amongst these methods as it applies several different tests, which address different problem areas (figure 2.1), and then attempts to use the information from these tests together. It is SpamGuru's approach to handling the problem that spam poses that appears the most promising and interesting.

3. Combining filters

This chapter begins by exploring the reasoning behind using multiple filters to obtain higher accuracy in classifying spam. It draws upon information presented in chapter 2, and examines the possible characteristics that can be expected from genuine user e-mail. After understanding what can be expected, possible methods that may highlight these characteristics are presented. The chapter closes by analysing the possible methods that can be used to successfully aggregate the results from these multiple methods.

3.1 The concept behind aggregating various results

This section begins by analysing why distinctions between different parts of an e-mail need to be made. Following the examination of e-mail structure, the characteristics of users and their use of e-mail is presented; which explains why multiple filters are likely to help in classification.

3.1.1 Selecting body content for analysis

It is generally accepted that there are technically two components to an e-mail message[38], one part is called the header section while the second part is called the body section. Both sections can be divided further as they both contain different types of information. The analysis of filters in section 2.2 revealed that most classifiers make a distinction between the header and body components of an e-mail message. This distinction is made by each filter's function varying depending on the component being addressed. This section presents the anatomy of the e-mail, and discusses the motivation for selecting body content for analysis.

3.1.1.1 The header section

The header of an e-mail contains descriptors designed for machine readability. The descriptors are meant to contain information regarding the recipient of the message, the sender, the route the message has traversed, and anything else deemed appropriate for one machine to relay to another[1]. The header section can be further broken down into what each header is meant to represent. Headers are generally not intended to be read by users, excepting the subject, date and sender fields which are typically rendered by the MUA.

The header component in solicited e-mail generally conforms to particular standards, and can be used to trace the path, and sender of the message[38]. However, in unsolicited e-mail, it is unknown which portion of the headers can be trusted and which of those cannot, as the relay/server sending the spam message may have added an unknown number of headers of questionable integrity[24]. Classifiers like SpamAssassin try to make use of this by checking the headers for syntactic validity as well as checking for I.P. address stamps from known open relays.

[1] Please refer to appendix section A.1 for an example of e-mail headers

The alterations to MTAs, as exemplified in section 2.1, are designed to enforce stricter regulations on header fields, and incite trust in communication between them. With the proposed changes implemented, header analysis would be a useful indicator. However if spam were to originate from home user machines, and thus through legitimate ISP mail servers, the analysis is much more difficult as the headers are exactly like those in a legitimate e-mail.

The sender of the message is also located in the header, but identifying messages based on sender names, addresses, or originating ISPs is not particularly easy. The caveats in such an implementation have been summarised in section 2.2.2.

3.1.1.2 The body section

The body of an e-mail message is the part of the message which is constructed by the sender. It is the part of the message that conveys information from a sender to a recipient, and thus to the user it is arguably the most important part of the message. A message can theoretically be divided up into a salutation, several paragraphs related to the information to be conveyed, and a sign-off component.

Unfortunately, the body component in solicited e-mail does not necessarily have to comply with any structure at all. An example of when such a case may arise is when communication takes place outside of e-mail and a message is sent in context of another conversation taking place. In such a scenario the message may not contain anything except a few words and a link or an attachment. This fact is well known by spammers and so is one of their mechanisms for attack[2]. This is also one of the facets of e-mail which make it difficult to categorise. Additionally, HTML formatting can now be used in messages, and as a consequence, spam messages are making increased use of HTML to obfuscate their messages[33] from the filters. HTML provides the ability to present data in a manner which is difficult to relate directly to the programming code, unless the HTML is actually rendered. This is shown in figure 3.1.

Code	Output
< table border=0 > < tr > < td > B < /td > < td > < font color=#FFFFFF > ... < /font > < /td > < td > V < /td > < /tr > < tr > < td > u < /td > < td > < font color=#FFFFFF > ... < /font > < /td > < td > i < /td > < /tr > < tr > < td > y < /td > < td > < font color=#FFFFFF > ... < /font > < /td > < td > a < /td > < /tr > < tr > < td > < /td > < td > < font color=#FFFFFF > ... < /font > < /td > < td > g < /td > < /tr > < tr > < td > < /td > < td > < font color=#FFFFFF > ... < /font > < /td > < td > r < /td > < /tr > < tr > < td > < /td > < td > < font color=#FFFFFF > ... < /font > < /td > < td > a < /td > < /tr > < /table >	B V u i y a g r a

Figure 3.1: HTML code compared to when it is rendered

[2]Please refer to appendix section A.2 for an example of a short spam message

Nonetheless, content analysing filters like Bayesian based filters[3], and SpamGuru[4] attempt to find a difference between spam and user e-mail.

3.1.1.3 Analysing body content

As most classifiers make a distinction between the header and body components of an e-mail message, and as there exist differences in their general purpose, choosing a section can help determine what tests can be performed. Header analysis has been shown to be primarily syntactic, or protocol based, and is dependent on the underlying architecture; whereas body content is much more dynamic and directly related to individuals and their communication style. The selection of body content for analysis allows for a much wider variety of tests to be performed, which could result in better classification.

The analysis of syntax and protocol compliance in the current architecture has not yielded success in message classification, and so forms the basis for architectural modification proposals (see section 2.1). By analysing body content, modifications to infrastructure do not need to be deployed. Instead, textual analysis methods require investigation.

As spammers are constantly changing the way information is presented, the field of textual analysis in e-mails is proving to have an effect on their success in message delivery. For this reason, body content is selected as the component for analysis in this study, which necessarily means that the problem is handled at the recipient's end, addressing 'the social cost' (figure 2.1).

3.1.2 Understanding body content and user profiles

In addressing the recipient user's side of the problem, using body content, an understanding of the user's e-mail is imperative. As body content is created by users, a single user's inbox is likely to contain messages that vary in authorship style, due to messages from multiple senders, however the nature of the messages are likely to be classifiable into only a few subject classes. The combination of subject classes are likely to be fairly unique to each individual, and the content of message bodies will reflect this. These subject classes are likely to be represented in the messages a user sends as well, since a user will typically only write about material they are interested in. As a result, the messages a user receives, and does not discard, along with the messages a user sends, will allow for the generation of a *profile* for the user[39].

The discarded messages on the other hand, if not explicitly tagged as spam by the user, run the risk of containing messages the user considers aged and irrelevant. This type of discarded message is not suitable for inclusion as spam as it does not fit the definition[5]. For this reason, it is important to recognise that while a profile of what an individual considers spam can be generated, the method of collection of profiling data must ensure that the data does not contain old messages which were previously considered valid.

Therefore the two profiles created, based on the textual properties of the body content, must define the applicable feature differences. The success of the naive Bayesian classifier[31, 30][6] in spam categorisation indicates that there is a difference between spam messages and user e-mail which can be detected through a text categorisation technique. This implies that spam has detectable features

[3] See section 2.2.5
[4] See section 2.3
[5] See section 1.3
[6] See section 2.2.5

within the text, which are different to that of typical user e-mail. The use of text categorisation techniques other than Bayesian have also shown to yield promising results[40, 33], which further suggests that classification of e-mail may be similar to that of document categorisation and authorship attribution.

3.1.3 The benefits of multiple filters

SpamAssassin[7][32] and SpamGuru[8][34] both work on the premise of combining the results of multiple tests to yield one outcome. While SpamAssassin compares a message against an extremely large rule set, the SpamGuru package uses several different techniques, ranging from rule based, to content analysis, to server feedback. The outcome of using multiple tests against a message has consistently proven to yield much higher accuracy in spam classification and yield lower false positives[34].

Additionally, by understanding the dynamics of body content, it is possible to see why multiple filters are likely to work better than solitary filters. While one filter can identify certain aspects of a message by generating a particular profile, multiple different filters will be able to detect many different features through the various profiles they have generated. The concept of multiple feature detection through multiple filters is further explored in section 3.2.1.

3.2 Analytical methods

This section begins by exploring textual analysis methods that are conjectured to yield significant information about messages that can be used to differentiate between spam and ham. Following this, a discussion of some of the possible methods that can be used to produce statistical results highlighting these differences is presented.

3.2.1 Textual analysis

Text analysis has proven to yield results of high accuracy in document classification[41, 42, 43, 44] by subject, and in some cases results as high as 99.8%[43] through use of N-gram techniques. This, along with theorised features of spam messages, motivates the methods discussed here. It is conjectured that the following methods, once aggregated appropriately, will yield an e-mail classifier of high accuracy.

3.2.1.1 Static dictionary, complete lookup spell checking

Static dictionary, complete lookup spell checking[45] supports multiple methods of querying a list of words to check validity, and the method employed depends on the application of the spell checker. In the context of text feature detection this means deciding on the type of information that is desired to be gathered from the sample.

It is possible to create character trees, as shown in figure 3.2, from a static dictionary. In the event of a spell check failing, a probability can be assigned as to the most likely desired word out of the dictionary. A contrived example could be that a user desired to spell the word "CHE", but instead

[7] Section 2.2.6
[8] Section 2.3

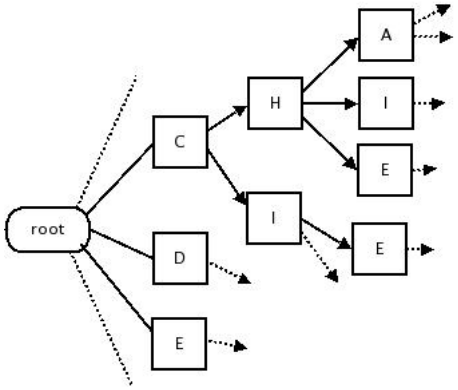

Figure 3.2: An example of a potential character tree used in a dictionary

typed "CJE"; if figure 3.2 represented the available possibilities, then from "CJE" it can be determined that there exist two possibilities for the user's intentions. This allows for a multitude of methods to assign a probability as to how "incorrect" the spelling is, which in turn results in the ability to assign probabilities as to the occurrence of any word, and thus the entire document.

It is also possible to perform total hashing of the words within the static dictionary and attempt to hash the words provided for verification for complete matching. The difference in the hashing method is that words which are mildly incorrect will be flagged as wrong, and it is not possible to use the hash table to estimate the likelihood of the desired dictionary word. It instead provides the total number of incorrectly spelled words and the ability to further analyse the incorrect ones in a different module if desired.

Total hashing is faster than tree based lookup methods since hashing applies a function to the word in question and a mapping is checked for existence, whereas tree based methods require trees to be constructed as well as traversed. For the purposes of e-mail content analysis, what may be interesting is to initially determine the percentage of words in the message that do not exist in the basic dictionary and later examine how many of those words could possibly be obfuscated words.

`V1agr@` *instead of* `Viagra`

Figure 3.3: An example of a commonly obfuscated word

Obfuscated words, as shown in figure 3.3, are more likely to occur in spam than in ham as spammers will want to hide well known spam words, such as "Viagra" and "Cialis"[46]. This in turn implies that spam messages are more likely to have a higher occurrence of unrecognised words than real user correspondence. For this reason, a static dictionary, complete hashing, complete lookup spell checker is used to determine the percentage of unrecognised words in messages; it is conjectured therefore, that user e-mail will have more words in common with the dictionary.

3.2.1.2 Static dictionary, augmented word checking

The static dictionary, complete lookup spell checker[9] provides the possibility to further examine words that were not recognised. This presents an opportunity to determine if there exist possible obfuscated words in the message. Spammers are known to want to hide words within e-mails from

[9] See section 3.2.1.1

filters, and so checking for obfuscated words helps determine if there are portions of the message which have been varied with this intent.

Checking for obfuscated words would require use of character trees (figure 3.2) for dictionary representation, as traversal through each character would be necessary to establish where the obfuscation takes place, if at all. As the goal is to determine if an actual word was hidden, rather than a genuine misspelling, a dictionary defining viable character substitutions[10] would provide for more accurate feedback on obfuscation, as opposed to a sender's competence with the English language.

Therefore, the implementation of an obfuscation detection module would require to use the same dictionary as the spell checking module, as well as an additional dictionary of viable character substitutions. The interesting results from such a module would be the number of previously unrecognised words that after analysis, match a word in the same dictionary.

3.2.1.3 Character N-Gram analysis

Character n-grams count the occurrence of a particular length of character string within a message. N-gram frequency analysis is a well known text feature extraction method[44, 47, 41, 43, 42] used in areas such as word prediction, document classification, and authorship attribution. Figure 3.4(a) demonstrates bi-gram analysis of the word `London`, with the corresponding frequency table in figure 3.4(b).

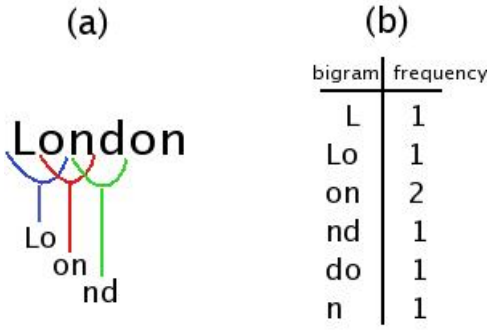

Figure 3.4: An example of bi-gram feature extraction

The concept behind n-gram analysis is that feature vectors, containing the n-grams and frequencies, of classes of texts is created from samples from each category; this creates category profiles. Unseen documents to be categorised are profiled through the same n-gram feature extraction technique used when creating the category profiles, and the "distance" from the existing categories is computed. The category which is determined to be the shortest distance away from the document is considered to be the class the document belongs to. This is summarised in figure 3.5.

The motivation behind using n-grams is that they have already been shown to work with reasonable accuracy in document categorisation[41, 42, 43, 44], as well as some studies on spam analysis[40]. The interesting aspect to observe is if the profiles of spam and user e-mail have a sufficiently large distance between them to enable accurate e-mail profiling.

[10] A viable character substitution would be using '`1`' instead of '`i`' or '`l`'

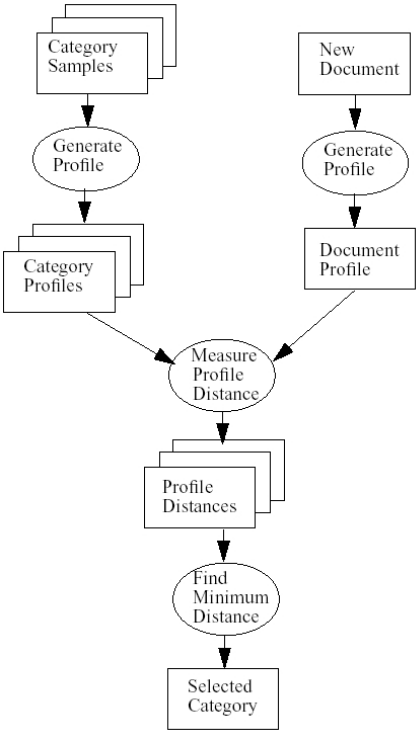

Figure 3.5: Data flow for N-Gram based Text Categorisation[43]

3.2.1.4 Character ratio calculator

The concept of character n-gram analysis is based on the fact that different types of texts have different frequency distributions of character strings. In addition to analysing a message's distance from the frequency profile of a category, the occurrence of particular characters against each other may prove interesting. For example, while the two most common characters in the English language, generally, are "e" and "t", their ratio of occurrence to each other may be different in spam. Additionally, the percentage of certain characters in capital letters may prove to be a discriminator, as spam is suspected of making use of capital letters for emphasis[32]. Therefore analysing the ratio between all characters may yield interesting results.

3.2.1.5 Part Of Speech (POS) Tag frequency analysis

POS tagging allows for the generalisation of sentences from containing individual sequences of words to the general tree structure of sentences, as shown in figure 3.6. A common part of computational linguistics is to perform POS tagging of documents in order to prepare the document for computational analysis. Studies have shown that POS tags can be useful in document classification[40, 48] as they can provide contextual information about the document. A well known and widely used POS tagset is the Penn Treebank Tagset[49], and many POS taggers based on this tagset are widely available.

POS tags may prove useful in distinguishing against spam and ham if the general structure of the messages is different. It may be reasonable to assume that the messages do differ simply because of the excessive use of HTML and word obfuscation that is present in most spam messages. For example, while the occurrence of "`buy V!<!>ag<!>r@`" may be very rare across all the spam messages that try to sell Viagra, the structure of sentence, "`VB NN . SYM . SYM NN SYM . SYM NN IN`", is more

```
Tiger Woods finished the big tournament at par
```

becomes

```
(NX Tiger/NNP Woods/NNP NX) (VX finished/VBD VX) (NX the/DT big/JJ
tournament/NN NX) at/IN (NX par/NN NX)
```

Figure 3.6: An example of POS tagging

likely to occur due to the number of unrecognised symbols and words that are used together.

Additionally, rather than simply counting the number of times certain tags occur, it is more interesting to determine how often the tags occur together. For this reason, by adopting the principle behind character n-grams, and by creating "POS tag n-grams", it may be possible to create more distinct profiles for both classes of e-mails.

3.2.1.6 Word frequency analysis

When considering the material that a person is interested in, and the language used to discuss the topics relating to those subjects, it is likely that a particular set of words will occur more than others. It is also likely that spam will not contain such a high count of those words, and rather contain more words involved relating to immediate transactions. The Bayesian filter's[11] success[31, 30] is based on this principle.

While the Bayesian filter uses probability of word occurrence, and location of occurrence, to ascertain how likely the message is to be spam[30], how often a particular word is used in relation to others may also prove interesting. This concept is based on the character n-gram technique, as profiles of the word occurrences in spam and legitimate e-mail need to be constructed for comparison, and a message is then ranked against profiles for distance measurement.

While single word occurrences will yield much about the subject orientation of the message, how words are grouped will also be an important reflection of the message orientation. This is likely to be the case as groups of words will yield more contextual information, and similar groups of words are likely to keep re-occurring in legitimate user e-mail. However, in spam messages, due to the level of word obfuscation that takes place, the re-occurrence of word combinations is expected to be low. If however, this is not the case, the objective in profiling spam would still have been achieved by finding the most common spam combinations.

3.2.1.7 Sentence Boundary Detection

Typical grammatically correct English uses punctuation to terminate sentences, however punctuation can also be present at different parts within the sentence where termination is not its purpose. Consider the sentence in figure 3.7, if the *full stop* were considered to be a sentence terminator, the boundaries detected would be incorrect. It is, however, possible to overcome the problems with heavily punctuated sentences with methods that can be trained on pre tagged corpora, or even pre-trained methods[50].

There are multiple benefits behind knowing the sentence boundaries within messages. These include being able to determine if there exists a difference in general sentence length between spam and ham, as well as the ability to extract the sentence for further parsing.

[11] See section 2.2.5

"The invoice from Dr. Richards came to $2034.51, which excludes V.A.T. at 17.8%."

Figure 3.7: An example of a heavily punctuated sentence

Accurately determining sentence boundaries allows for the full sentence tree to be ascertained[12], enabling for greater precision in word tagging[51]. Furthermore, word level n-grams can make use of the tags and restrict the n-grams to within sentences, rather then spanning over them.

Unfortunately, the implementation of an accurate sentence boundary detector can not be implemented when considering e-mail messages. This is mainly due to the fact that people do not necessarily use punctuation well in their messages, if at all. If there is no consistency across solicited human communication then it is conjectured to be unlikely to occur in unsolicited messages.

Therefore while sentence boundary detection appears interesting, within e-mail it appears to be a problem in its own right. It is more likely that other methods would yield more valuable information than sentence length. For these reasons, this component is highlighted, but not proposed for implementation.

3.2.2 Similarity measurement techniques

Each module implemented will require to return a statistical result to indicate the outcome of the processing. This result should be indicative of the e-mail's similarity, or disparity, to either category, and is an important aspect of module implementation. There are several methods available for similarity measurement, and the method used depends on the data made available by the module.

3.2.2.1 Euclidean Distance

Euclidean distance measurements plot results in a n-dimensional feature space, where n is the number of features being compared. Each result is treated as a point in the feature space and the distance, computed through Euclidean geometry, between a point and a target location is used as the fitness measurement value to what the target point represents.

As the feature space gets larger, computation of distance becomes much slower resulting in the requirement of feature space reduction. Principal components analysis is known to be useful in reducing points represented by large dimensions to fewer relevant dimensions.

Principal Components Analysis (PCA)

PCA performs extraction of the most significant features of a data set with high-dimensionality, and allows for the number of dimensions to be reduced while still maintaining a high degree of accuracy. It has been commonly used in applications such as facial recognition[52] where the number of dimensions is typically huge, and has yielded good results.

PCA allows for dimensionality reduction through significant feature selection. PCA will sort the dimensions of the data set into the most significant, and therefore highlight the least significant features

[12] See section 3.2.1.5

which would have minimal impact on the results if removed. It is important to note however that removal of any feature vectors will result in reduction in accuracy, as shown in figure 3.8[13].

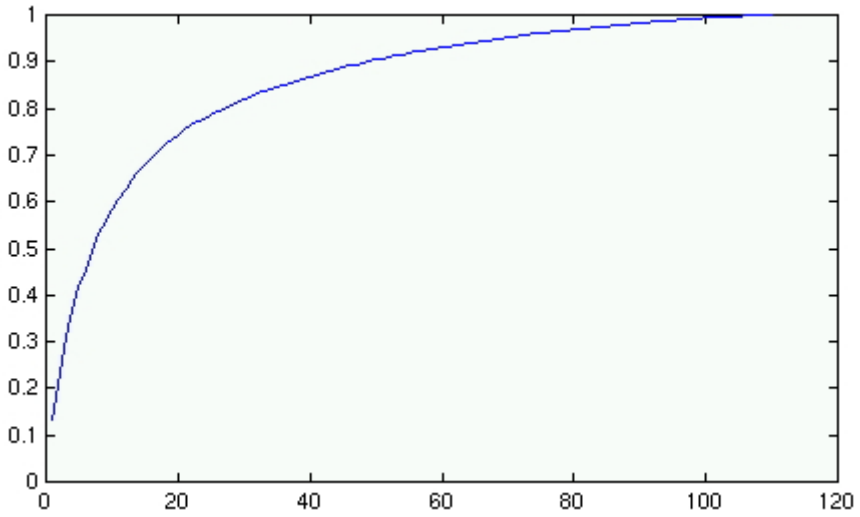

Figure 3.8: PCA analysis for facial recognition, *accuracy against no. of feature vectors*

The dimension size of the vectors being compared need to be the same size, consequently it is applicable to tests which do not introduce unforeseen components. Unforeseen components can require new dimensions to be created, and therefore result in the dimensionality being unbounded. An example of a space which would not lend itself easily to PCA is *word frequency analyses*. Word frequency analysis[14] has a very high likelihood of discovering words in new messages that are not present in the training set. This is a feature not only of the English language, but societal evolution, and thus beyond the control of the system. This implies that the words can not be used as dimensions, unless a static dictionary approach is used with one extra dimension for the number of occupancies of `unseen` words. The benefits of this however are restricted to the scope of the dictionary, therefore restricting the potential scope of the results.

A good example of a candidate space, other than facial recognition, would be any bounded space such as a character n-gram space. The dimensionality of character n-grams are restricted to $(\texttt{sizeOfCharacterSpace})^n$, and thus desirable to reduce if Euclidean distance measures are being used, and as n becomes large.

3.2.2.2 Spearman Rank Order Correlation Coefficient (SROCC)

The SROCC is a rank statistic that measures the "strength of the associations between two variables"[53]. It can be used to give a measure of "fitness" with another distribution of data, and as a consequence, can be measured against several distributions to find the best match.

The SROCC is an approximation to the exact correlation coefficient computed from original data, but as "it uses ranks, the Spearman rank correlation is much easier to compute." It is computed using the equation shown in figure 3.9.

Where T_i is the rank of a variable in the target data set, X_i is the rank of the corresponding variable in the sample being measured, and N is the total number of variables.

[13] Taken from http://joplin.ucsd.edu/Tutorial/matlab.html (last verified 20 Feb 2005)

[14] See section 3.2.1.6

$$\boxed{r_s = 1 - \frac{6(\sum_i^N (T_i - X_i)^2)}{N(N^2-1)}}$$

Figure 3.9: Spearman Rank Order Correlation Coefficient

This coefficient is useful when comparing data from n-grams for example. Primarily due to the fact that the item count of n-grams within the category database will be much larger than that found within one document. Thus, it is not appropriate to compare the two counts directly, but the comparison of rankings would yield a much better interpretation of similarity.

3.3 Aggregating Results

There are multiple methods for aggregating the data from results, and each method has it's advantages[54]. As there are many methods, and much theory behind them, this section only briefly introduces two well known methods, starting with the "Artificial Neural Network", followed by "Support Vector Machines."

3.3.1 The Artificial Neural Network

The concept behind the artificial neural network is inspired by the design of the brain, and represents a highly simplified model of it. Since it's inception there have been many sophisticated modifications to the model which have led to closer modelling of the biological design. A brief overview of the units within a neural network is provided, followed by an outline of the possible structures using these units[1, 2], with a brief summary of training neural nets.

Neural networks are composed of nodes, and every node is connected by a link. Each node is known as a neuron, and the links are sometimes referred to as synapses. Each neuron can have several inputs, which would be links carrying data into the neuron, and several outputs. Moreover, each link from neuron j to neuron i is used to propagate data; and every link has associated with it a weight, $W_{j,i}$, which is meant to be representative of the strength of association of the link. Additionally, every neuron has associated with it, it's own threshold, which is used in determining how it should respond to its inputs. Following this design, for the neuron network to function, every neuron computes a function over the sum of its weighted inputs, to produce some output that is sent down it's output links, i.e.:

$$output_i = f(\sum_{j=0}^{n}((W_{j,i} * output_j) - threshold_i))$$

The function computed is referred to as the *activation function*, and allows for the neural network to function as a whole. The activation function of the network requires to be non-linear, otherwise the "neural network collapses into a simple linear function."[2] The sigmoid function is typically used as the activation function and is represented by the equation $\frac{1}{1+e^{-x}}$.

"There are two main categories of neural network structures: acyclic, or feed-forward networks and cyclic or recurrent networks."[2] Feed-forward networks have no internal and state and represent a function of their immediate input; whereas recurrent networks use the output they generate to feed back into the network's inputs which means their behaviour constantly changes. Feed-forward networks are explored here as their models are simple, yet powerful.

A feed-forward network can be single-layered, or multi-layered. In short, the difference between the two is that single-layer neural networks are used to find linear separability in problems, whereas multi-layer neural networks are used to try and solve problems that may not be linearly separable. A single layer neural network, consisting of only one neuron, with all the inputs leading to it, as depicted in figure 3.10(b), is also referred to as a perceptron.

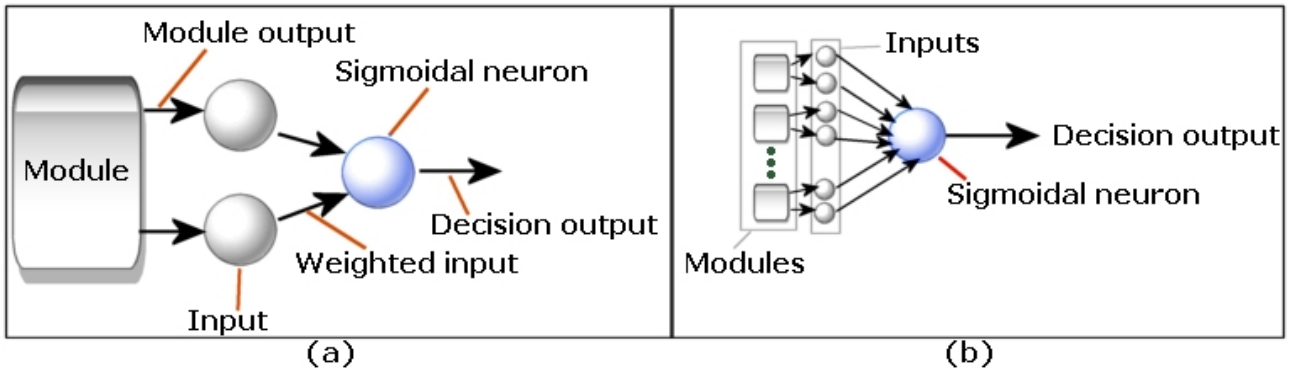

Figure 3.10: A simple illustration of how the neuron is used

A common use of the perceptron is depicted in figure 3.10(a), where the outputs of some module need to be used to make some binary classification. In this instance, if the separation between the outputs is linear then the perceptron will provide good results. It is important to note however, that a perceptron must be trained, otherwise the neuron has no ability of knowing how to categorise the inputs. Training augments the weights on the inputs and the threshold of the neuron depending on what the desired output of the neural net is. In order to carry out training on a neural net, a number of examples must be provided and a training algorithm executed over the net.

3.3.1.1 Training neural networks

A well known method for training single-layer neural nets is through least-squares techniques[1, 2] based on the sum-of-squares error function. The sum of squares error function essentially computes the difference between the outputs of the neural net and the target outputs, and it is used by attempting to minimise the difference. Multiple methods to enhance the rate at which this can be done have been devised ranging from gradient descent techniques, to iteratively reweighted least squares. The details of these methods are covered extensively in [1].

To train a neural net, a *training set* is required, and it should have sample inputs with ideal outputs. It is generally accepted that the larger the sample size, the better the neural net will be in the general case.

3.3.2 Support Vector Machines

Support Vector Machines (SVMs) are a different type of learning method to neural networks, and are part of a more general criteria known as *kernel machines*[2]. A brief overview of the concept behind SVMs is provided in this section.

SVMs are capable of representing complex, non-linear functions and have a very efficient training algorithm. The basis behind Support Vector Machines is that any problem is linearly separable if

represented in a high enough dimensioned space. This is illustrated in appendix figure B.1 and is the motivation for the learning method.

Unfortunately, solving a problem is not as simple as projecting it into a space represented by an exceedingly large number of dimensions as there is the risk of over-fitting the solution to the training set. To solve this problem, SVMs are designed, through quadratic programming, to find the optimal number of dimensions, so the different datasets have the largest margin between them. This allows for the linear separator to be robust enough to generalise. More information on support vector machines is available in [2, 55]

4. Implementation

In section 3.2 several ways to analyse text for distinguishing features were proposed. Section 3.3 presented a variety of techniques in which to extract meaning from the combined results of those methods. This chapter presents the technical details on how the methods detailed in the aforementioned sections were implemented. The implementation enables the evaluation of the methods, which is discussed in the following chapter.

Section 4.1 opens with the outline of the mechanism used for storage of data. Leading into section 4.2 which discusses how profiles are created, and statistics generated; along with the inner workings and relationships of the filters, as modules. Followed by section 4.3 which describes how the statistics are preprocessed; before they are trained and tested on the neural network described in section 4.4. The derived work-flow for this procedure is presented in figure 4.1.

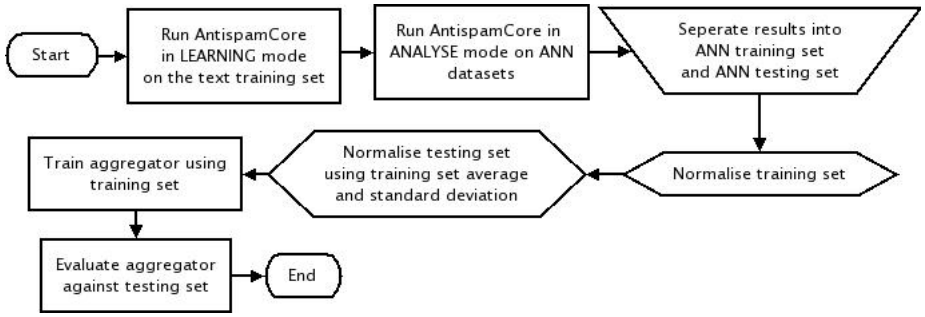

Figure 4.1: The overall work-flow

4.1 Data storage

One of the basic requirements of the system is the storage of messages, and results from processing pertaining to those messages. Rather than spend time developing efficient file sorting and accessing modules, it was decided to use efficient methods of storage already available. For this reason, existing database storage techniques were investigated.

Storage of the messages and results in a database meant that all the data would be available in a standard format, and databases offer the flexibility of integrating with any programming language. They also allow for the data structures of the entire data set to be changed when required without any programming code needing modification. Additionally, the databases can be configured to automatically reject duplicate messages, which would maintain the integrity of the sample data. Finally, by using a database to store the data to be processed, all the data supplied would be reformatted, and made available in a common format. Thereby overcoming any bias that may exist from data formatted differently.

The investigation of databases found that MySQL[1] offered the fastest performance out of many other open-source options[56], and was therefore selected as the data storage mechanism.

[1]MySQL is an open-source, highly optimised, database engine.Available at http://www.mysql.org (last verified 21 Feb 2005)

4.2 Profile and statistical result generation

The purpose of generating profiles of spam messages and ham messages is to enable comparison of messages to the profiles. Comparison of messages against the profiles produces statistics which aid in determining which category the messages are closer to, as discussed in section 3.2.2.

Consequently, the methods used to generate the profiles are quite similar to those which evaluate against the profiles. Therefore the modules were designed to be used in either circumstance. Creating profiles was defined as `LEARNING mode` whilst comparing against profiles was termed as `ANALYSE mode`. While the profiler and analyser were merged into one unit, they were kept separate from the aggregator as its function is fundamentally different.

The methods discussed in section 3.2.1 were implemented in several different modules, and the "`AntispamCore`" mentioned within figure 4.1 was developed as the controller of all these modules. It is the module which manages the profile generation and analysis result storage. The AntispamCore's general architecture is discussed here, followed by an overview of its internal processing sequence.

4.2.1 The AntispamCore architecture

The AntispamCore was developed in JAVA with each module represented as a different class. The relationship between classes is detailed in figure 4.2. This section provides an insight into the general architecture of the profiler/statistical generator by providing a description of each module implemented.

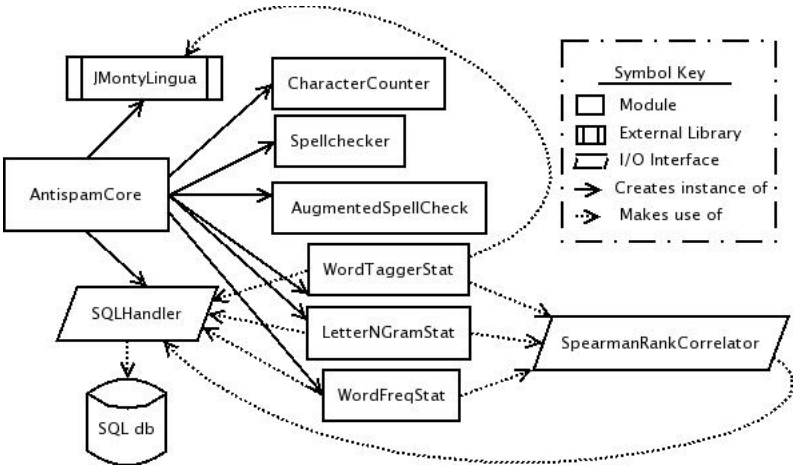

Figure 4.2: The relationships between modules

It is important to note that the AntispamCore creates instances of most of the methods, and passes the objects to the classes which require access to the resources they provide. This is done so that the same library does not need to be reloaded with each method, which is important for the external library `JMontyLingua`[57] and the SQL interface, `SQLHandler`, which both have considerable loading overhead.

JMontyLingua JMontyLingua[57] is a library provided to enable tagging of sentences using the Penn Treebank tagset[49]. It allows for large amounts of text to be supplied, and returns that text with the library's best effort at tagging.

JMontyLingua is loaded at startup by the AntispamCore, and passed to `WordTaggerStat.java` when it is invoked to perform a function on data.

SQLHandler The SQLHandler acts as the conduit between the application and the SQL engine managing the SQL database, as discussed in section 4.1. Each module which requires to store or access data, requires access to the SQLHandler.

SpearmanRankCorrelator This class is used only in `ANALYSE mode`, and is invoked only by methods which use the Spearman Rank Order Correlation Coefficient (see section 3.2.2.2) as the similarity measurement method. The class provides the option to either correlate the ranks of all attributes, or a specified number of attributes. The option is retrieved from the AntispamCore as a static variable.

CharacterCounter Character percentage calculation, as discussed in section 3.2.1.4, is implemented by this class. The total number of each possible character is calculated and divided by the total number of characters in the message, thus yielding an indication of the significance each character holds within the message.

The AntispamCore launches an instance of this class and invokes it by sending an e-mail message to it. Once calculations are complete the class returns an array containing the percentages. This module only runs in `ANALYSE mode`.

Spellchecker As described in section 3.2.1.1, a static dictionary spell checker is implemented. The dictionary contains 38,472 words and the word comparisons are case insensitive. The dictionary is loaded in a hash table for *complete lookup*, and all non-matching words are stored in a dynamic array so they can be later retrieved by another method.

The AntispamCore launches an instance of this class and invokes it by sending an e-mail message to it. Once calculations are complete the class returns the number of unknown words along with the total number of words in the message. This module only runs in `ANALYSE mode`.

AugmentedSpellCheck The unknown words from the spell checking class are examined by this class. Along with the original static dictionary, an additional dictionary with alternative symbols that a person could use to achieve the same visual effect of certain characters, is also loaded. The concepts of this method are outlined in section 3.2.1.2.

The AntispamCore launches an instance of this class and invokes it by sending a list of unrecognised words to it. Once analysis is complete, the class returns the number of words that with substitution of characters, matches an item in the dictionary. This module only runs in `ANALYSE mode`.

WordTaggerStat Using the Penn Treebank tagset[49], and JMontyLingua[57], this module can run in two modes, `LEARNING mode` and `ANALYSE mode`. The module is invoked to record the number of occurrences of two tags together, and three tags together in a given message.

In `LEARNING mode` this number is added to the profile of the category the message is meant to be in. For example, if the message is marked as spam, then the occurrences would be added to the spam profile POS tag database.

If the module is invoked in `ANALYSE mode` then the classification of the message is not available to the module, and the tags are ranked according to their counts. The Spearman Rank Correlator is then invoked with the POS tag profiles, and the ranked counts for the message. The SROCC module performs the necessary calculations and returns the correlation coefficient to the spam and ham profiles.

The AntispamCore launches an instance of this class and invokes it by sending an e-mail to it. Once analysis is complete, if the class is in `ANALYSE mode`, it returns the coefficients as given by the SROCC module, otherwise it returns a status code indicating learning took place successfully. The concept behind POS tag frequency measurement is detailed in section 3.2.1.5.

LetterNGramStat This class functions in both `LEARNING mode` and `ANALYSE mode`, and performs similarly to the WordTaggerStat. The exceptions are that the class does not make use of JMontyLingua or its Penn Treebank tagset, and counts groups of characters as opposed to tags. The idea behind character n-grams is covered in section 3.2.1.3.

LetterNGramStat runs tests over monograms, bigrams, trigrams and quadgrams. The AntispamCore launches an instance of this class and invokes it by sending an e-mail to it. Once analysis is complete, if in `ANALYSE mode`, the class returns the coefficients as given by the SROCC module.

The work-flow within the AntispamCore is detailed in figure 4.3, demonstrating how the flow is different depending on the mode the AntispamCore is invoked in.

WordFreqStat This class functions in both `LEARNING mode` and `ANALYSE mode`, and performs similarly to the LetterNGramStat module. The difference is that instead of characters, this module works with words. Multiple separators between words were defined, including a range of punctuation marks, spaces, and certain brackets.

WordFreqStat runs tests on word monograms and bigrams, as discussed in section 3.2.1.6. The AntispamCore launches an instance of this class and invokes it by sending an e-mail to it. Once analysis is complete, if in `ANALYSE mode`, the class returns the coefficients as given by the SROCC module.

4.2.2 The AntispamCore processing sequence

As the AntispamCore is responsible for generating profiles as well as statistical results, there needs to be a clearly defined difference in the mode of operation. As described in earlier sections, these two different modes have been referred to as `LEARNING mode` for generation of profiles, and `ANALYSE mode` for statistical results. An overview of the difference in processing sequences is discussed here, and is presented in figure 4.3.

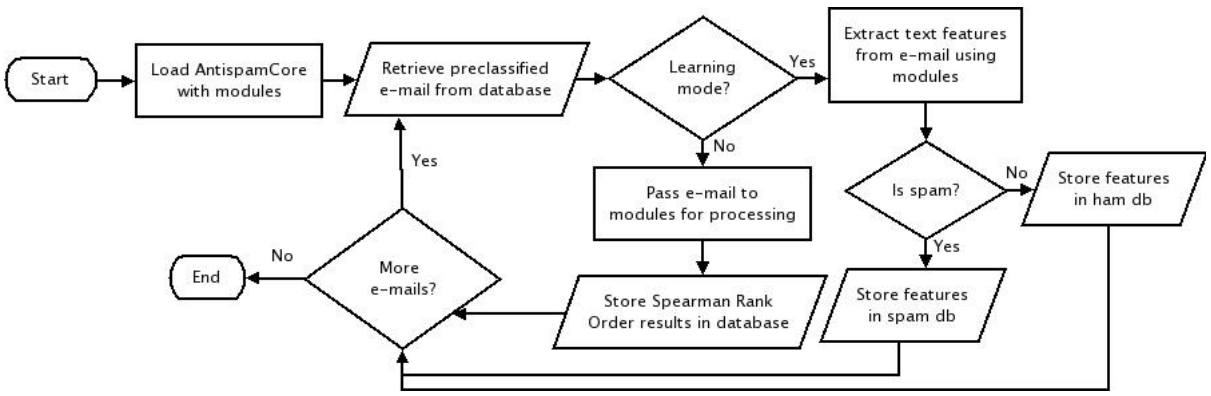

Figure 4.3: Work-flow of the AntispamCore

Figure 4.3 shows how the processing of the AntispamCore varies depending on the mode it is invoked in, specifically how it handles the data extracted from e-mails. In `LEARNING mode` the reclassification

tag, of ham or spam, is used to prescribe which database the data from the learning modules is to be stored in. This is the key feature which allows for profiles to be created. If the messages were not tagged, it would not be possible to create profiles with any meaning. `LEARNING mode`, necessarily, is configured to run and build profiles before the core can operate in `ANALYSE mode`.

`ANALYSE mode` is dependent on the profiles created in `LEARNING mode` and will fail to operate unless profiles exist. Once running in `ANALYSE mode`, the AntispamCore has each module generate statistics based on the features they are meant to detect, and return them to the core. The individual modules are not informed of the preclassification tag associated with the email. When all the modules have finished processing the message, the core then associates the results with the e-mail and stores them in a generic results database along with the preclassification tag. The tag, along with the statistics, is needed for later use with the results aggregator.

Overall, the AntispamCore is a critical component of the implementation as it handles two of the early phases in the overall work-flow (figure 4.1), learning the difference between the categories, and preparing the statistics for the neural network. `LEARNING mode` must be run before `ANALYSE mode` so that there exist profiles to compare unseen messages to. It is important to recognise that the reliability of the statistics produced in `ANALYSE mode` can be directly attributed to the *size* of the profiles, where *size* is the number of e-mails processed in `LEARNING mode`. If too few e-mails have been used in generating the profiles, then the data in unseen messages could easily be calculated to be different by the same amount to both profiles.

4.3 Statistics preprocessing

The neural net runs as a separate piece of software, consequently it does not have access to the SQL database through the SQLHandler. So, after the AntispamCore generates statistics for the training and testing datasets, they need to be extracted from the database and reformatted to suit the neural net. This constitutes preprocessing of the data.

As the data in the database is not explicitly flagged as training or testing, the data must undergo manual separation once it has been extracted from the database. MySQL supports the data being exported in Comma Separated Variable[2] (CSV) format, which is readable by the neural net framework's software, Matlab[3].

Once the data is imported into Matlab, separation of the data into two different datasets is carried out. After re-defining the datasets the results in the training set are normalised to have mean zero and a standard deviation of one. This is carried out by applying the functions in figure 4.4 to the training set.

$$\mu_{training} = \frac{\sum_{i=1}^{N} X_i}{N}$$
$$\sigma_{training} = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \mu_{training})^2}{N-1}}$$
$$S_x = \frac{R_x - \mu_{training}}{\sigma_{training}}$$

Figure 4.4: Equations for normalisation

[2]CSV is a well known standard format which represents the data in ASCII with each field separated by a comma, and each row separated by a carriage return

[3]Matlab is a mathematical software toolkit, released by MathWorks, available at http://www.matlab.com (last verified 24 Feb 2005)

After applying the function, the parameters $\mu_{training}$ and $\sigma_{training}$ are yielded, along with a normalised training set. The normalisation function is then applied to the testing set results, but rather than generating a $\mu_{testing}$ or $\sigma_{testing}$, the parameters from the training set are used. This ensures that the testing data is aligned the same way the training set is, enabling the neural network to evaluate each result fairly.

In summary, the preprocessing of the data after generation, and prior to working with the neural net, is a simple reformatting method. Ultimately no actual modifications are made to the results in a way that could be seen as altering the integrity of the data.

4.4 Neural net framework

Single layer neural nets are a well known adaptive classification method, and are recommended as the starting point for exploring unknown search spaces[58] before more complex architectures are explored. As a result, a neural net was used as the aggregator for the results, and the framework used was called Netlab[59], based in Matlab. The framework was used to construct a single layer, logistic neural net with the numbers inputs matching the number of outputs from all modules, and one output for evaluating all the filters together. Individual filters were evaluated using a smaller single layer neural net with a single output. As discussed in section 3.3.1, there are two phases to a neural net, the training phase and the testing phase.

In the training phase, the statistical results from the *training set*, along with the classification tags of the corresponding messages, were used with the iterative re-weighted least squares[4] (IRLS) algorithm to set the weights of the neural net.

Once the training phase had reduced the error as far as possible on the training set, the testing set was used against the neural net. The output from the neural net was stored in an array and, for varying thresholds of binary classification, compared to original classification tags of the messages. The threshold for classification was varied as the neural net outputs a number between zero and one indicating the likelihood of the message belonging to either category. As there are only two categories, the threshold is considered to mark the boundary of binary classification. Therefore, it is altered to find the threshold which yields minimum error. This is done on the testing set as a user would typically be able to vary the threshold of the spam filter, and this option reflects such a case.

[4] See section 3.3.1.1

5. Experimental method and results

The theorised outcome of the implemented filters was discussed in chapter 3, and the implementation of those filters, along with the data handling methods, was outlined in chapter 4. This chapter provides further details the experimental setup used with the implementation, and presents the different cases analysed. The conclusion to the chapter follows, with an analysis of the results, and a comparison to some of the methods detailed in section 2.2.

5.1 Experimental setup

The multi-staged procedure that has been derived is executed on the filters through two experiments. The first experiment uses all the text features within an e-mail in evaluation, whereas the second only uses the top 15 features. Both experiments share the initial profile generation stage.

This section begins by introducing the corpora used in the experiment, followed by how the corpora was prepared for analysis. The profile generation phase, is then outlined, and the two different test cases are presented and described.

5.1.1 Introducing the experimental corpora

5.1.1.1 SpamArchive.Org for the spam corpus

A large number of message samples are required for reliable experimentation, and large repositories with dated collections are preferable. There exist multiple spam repositories, and spam is considered generic enough by definition[1] for it to be collected from any, or multiple sources. As such the selection between available repositories is a matter of personal preference; for the purposes of this study spam was harvested from SpamArchive.Org[60] over the same time duration within the ham corpora, ten months: from January 2004, until October 2004.

5.1.1.2 Enron email for the ham corpus

The e-mail of personal users, to create the ham corpus, is much more difficult to obtain than spam. This difficulty is mainly due to privacy reasons, and due to the problem of being able to acquire large numbers of messages within a particular time frame. While it is possible to use a single volunteers archive, it is unlikely that they will have more than a few thousand messages within a year. Using multiple volunteers from dissimilar backgrounds, unless very carefully selected, may result in a very broad range of interests being reflected in the profile[2]. Samples which are too small, or which contain messages from multiple unrelated sources can lead to unreliable results. Messages from newsgroups, while abundant, may not be entirely representative of a personal user's e-mail as the messages are typically not directed towards one individual.

[1] See section 1.3

[2] See section 3.1.2

In light of this, the e-mail of multiple users within a corporation is considered the best option. The classifier is then explored for its ability to accurately recognise the messages for an entire corporation as opposed to an individual, much like SpamAssassin or SpamGuru (see section 2.2.6 and 2.3). The Enron corpus[61] contains a large amount of messages that were used in day to day communications within Enron, and thus contain large amounts of e-mail targeted to individuals all with common interests. Multiple people from within the available data set were selected at random to generate the ham corpus. Messages were taken primarily from user inboxes, but portions of outboxes were also used. The date range spanned by the ham corpus is a ten month period from May 2001 until February 2002.

5.1.1.3 Limitations of the available corpora

The Enron corpus[61] that is available has undergone preprocessing by Dr. Melinda Gersavio[3] of the AI team in SRI to correct data errors that were present. As a result of this processing, header information within the e-mails has been lost, file attachments have been stripped, and although not documented, it appears as though any HTML segments that may have existed, have been removed. While the loss of file attachments may not be significant, the lack of headers means that no header analysis is at all possible. This means that body content is the only aspect that can be examined with this corpus.

Analysis of the body content is further restricted by the stripping of HTML, however most MUAs in use by clients produce both HTML and text portions to e-mail messages upon sending, and the portion rendered is an option for the recipient. Additionally, popular spam filters like SpamAssassin[4] actively look for messages which contain HTML and/or differences between HTML and text portions, and messages with more HTML are ranked more likely to be spam[32]. It is therefore not unreasonable to expect a message to contain a plaintext portion, and preference that part for analysis. Thus, analysis will favour the plaintext parts of all messages, including spam, and in the event of an absence of plaintext, HTML will be accepted; any message that does not contain a plaintext portion or HTML portion will not be examined as it will not be viewable by a user of a typical MUA.

5.1.2 Preparing experimental data

The corpora of messages that were used to represent and test spam and normal e-mail, needed to be made available to be tested on. The data that was made available for testing, as it was to be stored in a common message database, would be automatically checked for duplicates in the message ID string as well as body content. This section briefly reviews the corpora chosen, and details how messages from both corpora were made available for analysis.

The Enron corpus[61] was chosen to be representative of ham messages, while SpamArchive.Org[60] was chosen as the source for spam, as discussed in section 5.1.1. Both corpora were stored differently and required to be reformatted for use, consequently a small JAVA application was made to read the `unique ID`, `subject`, and `body content`[5] from the different corpora, and store them in the database. As the categories of the messages were known, it was possible to add the preclassification tag for every e-mail, to mark it as either ham or spam. It is important to note that due to the limitations of the

[3] http://www.ai.sri.com/~gervasio (last verified 18th Feb 2005)

[4] See section 2.2.6

[5] See sections 3.1.1.3 and 3.1.2

corpora as detailed in 5.1.1.3, the JAVA message harvester was configured to be biased towards the text component of messages as required.

A total of 8,083 messages were harvested from the corpora and inserted into a database. The total size of the messages stored was 16MB. 4,366 (54.0%) of the messages were spam, and 3,717 (46.0%) were messages from randomly selected users within the Enron corpus. The three different data sets were prepared into sizes as shown in figure 5.1.

data set	**Spam**	**Ham**
Profiling	1009	1000
Training	1500	1275
Testing	1857	1442
Total	*4366*	*3717*

Figure 5.1: The data set breakdown

The *profiling* dataset, in figure 5.1, was used to generate the profiles for each category, while the *training* and *testing* datasets were allocated for analysis, and use with the neural net.

5.1.3 Generating profiles

In order to generate the profiles for the spam and ham messages, the AntispamCore was set to `LEARNING mode` and invoked on the *profiling* data set (figure 5.1). The core generated profiles for the two different sets with the breakdown of unique entries as shown in figure 5.2.

profile component	**Spam record count**	**Ham record count**	**Total possible size**
Character n-grams	350,934	151,599	270,549,120
POS tag n-grams	13,538	15,234	47,952
Word frequencies	192,496	123,299	∞
Total	*556,968*	*290,132*	

Figure 5.2: The data set breakdown

5.1.4 Analysing the corpora

Once the profiles were generated, it was possible to begin generating statistics for the messages in the *training* and *testing* data sets. Two variations of the same type of analysis were carried out and they are both detailed here.

The tests involve varying the number of records that the Spearman Rank Order Correlation Coefficient (SROCC) is computed for. In one of the tests, every record has a SROCC generated, and in the other only the top 15 ranking items in the message are compared. In ranking and comparing every item, it is assumed that the coefficient will be of high precision, and quite accurate in depicting the orientation of the message. The ranking of the top 15 occurrences in the message however, is inspired by Graham's[30] method used with naive Bayes. It is to see if an email can be characterised by its top 15 most occurring attributes.

The "top 15 test" runs the risk of having the coefficients extremely bias away from the categories. This is because only the top 15 ranks within the email are searched for against the profile, if a corresponding

entry in the profile is not found, *maximum distance* is assumed[6]. Therefore, if a message's top 15 attributes across all the tests do not occur in either corpus, the orientation of the message will be unknown. This is however thought to be of an extremely low probability, especially since normal user correspondence should contain common attributes.

To create the statistical results, the AntispamCore was put in `ANALYSE mode` and the *testing* and *training* data sets were processed by the implemented modules, and the results were stored associated with the original message[7]. Once all the messages in the data sets had completed processing, the results had to be manually separated into two data sets again, and preprocessed as described in section 4.3. The "top 15 test" was run after the test which compared all attributes, both tests used the same messages in the *training* and *testing* data sets to ensure fairness.

5.1.5 Aggregating results

Once the results from the individual filters had been generated by the AntispamCore, separated into the different tests, further separated into the *training* and *testing* sets, and then preprocessed by Matlab as described in section 5.1.4, they were eligible to be used with the neural net[8].

Results produced from individual modules were initially evaluated independently using a single layer neural net for each module. The preliminary examination of the results yielded information about the quality of the results from tests, and although this is further discussed in section 5.2, the initial investigation of statistics led to certain results not being used. The `Character Ratio Calculator`, `Static Dictionary Spell Checker`, and `Static Dictionary Augmented Word Spellchecker`, were not used for reasons explored later. This left 16 statistics from 8 modules, for each test, for analysis with a larger single layer neural net. Although the structure of the neural networks are the same for both tests, two neural nets were generated so that each test could be evaluated independently.

The larger neural net was used to aggregate the results from the remaining modules, so it was trained and then tested, as described in section 4.4. The testing phase on the neural net yielded the net's perception on the most likely classification of the messages. These results from the neural net are the defining results of the overall procedure and are analysed in section 5.2.

The generation of results from the neural net for the entire testing set, and both tests, concludes the experimental procedure.

5.2 Analysis of results

After experimentation was conducted on the implementation of the combined classifier, the results from the testing procedure highlighted some interesting points about the modules and aggregator. Section 5.2.1 discuss possible reasons why several modules did not perform as expected; while section 5.2.2 analyses the modules that did perform well and compares them against each other to find possible reasons why some fared better than others. Section 5.2.3 compares the results of the two tests carried out against each other; analysing the characteristics of the implementation to determine why using all the attributes in a message produces better results than only using the top 15. This is followed by section 5.2.4 which compares the outcome of the implementation to the methods discussed

[6] See section 3.2.2.2

[7] This work-flow has previously been outlined in figure 4.3

[8] See section 3.3.1 and 4.4

in section 2. Finally, section 5.2.5 analyses the procedures and corpora used to justify that the results are free from known bias, but discusses why further testing is required for more confidence.

5.2.1 Analysis of poor performing modules

From the modules discussed in section 4.2.1, after using them to produce results, several were deemed inappropriate for use in the larger neural net for several reasons. These reasons include inadequate performance, and inappropriateness for testing purposes. These reasons, and the relevant modules are further explored here on a module by module basis. The section opens by analysing the augmented dictionary spell checker, followed by the static dictionary spell checker, and then with the character ratio calculator, concluding with a summary of the information that may have been lost as a result of poor design in these modules.

5.2.1.1 Static word, augmented dictionary spell checker

The static word, augmented dictionary spell checker, conjectured to be useful by detecting obfuscated words (see section 3.2.1.2), performed the worst out of all the modules implemented. The false positive rate of the module was too high to be considered useful. For this reason, the module was not used as part of the combined filter array. It is probable that the limitations were due to the implementation (section 4.2.1) of the module as opposed to the concept, and this is discussed here.

The detection of obfuscated words is a feature available in several sophisticated spam filters such as SpamGuru and SpamAssassin[9], and both claim to make use of the information yielded by it. While SpamAssassin looks for specific words like “Viagra” and “Cialis”[32], IBM’s SpamGuru simply claims to analyse the message and detect them through some proprietary method[62]. As the feature has been included in such filters, the concept of detecting hidden words appears to be one of interest, and possibly one that can provide further insight into the type of message that’s using them.

However, the poor results of the module may be due to the implementation used in the AntispamCore. The technique utilised may be too primitive for the techniques used by today’s spammers. As the implementation uses a static word dictionary, and a static substitution dictionary, it may be too restrictive and unable to detect newer words.

Additionally, the module relies on the spell checker to correctly separate words. If the word is separated by tokens which the spell checker deems to be word separators, then the augmented word checker will fail. The detection will fail, as the obfuscated word will be processed as multiple words instead of one.

The module also fails to take into account the use of HTML tags within words. Consequently, any messages which contain words obfuscated through use of HTML, will have a higher chance of remaining undetected than words obfuscated using simple ASCII symbols.

In order to improve this module the word dictionary should be dynamic, and based upon the words the mono-word frequency analyser accumulates. Furthermore the module should attempt to intelligently merge words close together in the list of unknown words provided by the spell checker module. The success in intelligently merging tokens can be quantified through use of the dynamic dictionary. By checking if two merged tokens create part of a larger word the exploration method through the dictionary can terminate when merging tokens fails to yield a possible word, or succeed when a word

[9] See section 2.3 and 2.2.6

has been found by altering and merging. An alternative to this dictionary method could be to use word stemming techniques as proposed by Ahmed and Mithun[63], as that may also achieve the desired effect. Additionally, the HTML within messages should either be removed prior to spell checking, or it should be processed to ascertain its role in the display of the message.

Overall, the implementation of the augmented word detector has proven to be too basic for use, and modifications are required before any valid conclusions can be made about word obfuscation in spam messages.

5.2.1.2 Static word, dictionary spell checker

The static word, dictionary spell checker was thought to be useful as spam messages were thought to contain a higher percentage of misspelled words than ham messages (see section 3.2.1.1). While the results show this to be the case, the data is not reliable enough to be useful, as when approximately 95% of spam is detected by this module, ham messages have a 25% chance of being misclassified. The possible reasons for failure of this module are discussed in this section, and potential solutions are also conjectured.

The failure in reliability in this module could possibly be due to the static dictionary being too restrictive. The dictionary used may not be reflective of either category's use of the English language, and thus hold no relevance to any document produced within either category.

In order to overcome this issue, it may be useful to use a dynamic dictionary generated from the corpus representing ham messages, as performed by the mono-word frequency analysis module (section 4.2.1). If this were the basis for the dictionary then it would be more bias towards user solicited messages, and there would be a higher probability of misspelled words occurring in spam messages than in user messages.

The benefits of such modification are that it is still remains dis-similar to the mono-word frequency analysis approach, as well as Bayesian techniques, and simply reports the number of unrecognised words. Furthermore it retains the ability to adapt as the user's language changes.

5.2.1.3 Character ratio calculator

The character ratio calculator was conjectured to expose patterns between the relationships of character usage in spam messages and ham messages, as discussed in 3.2.1.4. The tests on the method revealed that the results were similar to those of the character mono-gram frequency test. Use of both methods was thought to be non-beneficial, especially considering that the character frequency test appeared to slightly under-perform against the character mono-gram. Both modules were compared through the results produced by their individual perceptrons.

5.2.1.4 Summary

From the analyses carried out, it can be ascertained that the loss of the character ratio calculator is insignificant as it underperformed compared to the majority of other tests. However, it is conjectured that potential information has been lost as a result of implementing a static dictionary as opposed to a dynamic dictionary, for the dictionary based modules. Had the modules utilised a dynamic dictionary, that was based on the user's profile of words, they could have potentially provided insight into the

number of times unrecognised tokens appear in spam when compared to the lexicon of users. The number of recognised words in ham, based on a users own dictionary, is expected to be low, but this is not thought to be the case for spam messages.

5.2.2 Comparison of module performance based on test results

As outlined in section 4.4, the results from each module are used to train individual single-layer neural nets for evaluation purposes. The results from each module are then later aggregated using one large single layer neural net. This section presents the graphed performance of each module in both tests, using the top 15 and using all attributes within a message, graphed in figure 5.3 and figure 5.4 respectively. The modules are presented in comparison with the other modules, and a brief analysis of the significance of the results is provided. It should be noted however, that the success of the modules are discussed in the context of the more successful experiment, using all the attributes in a message, as this provides for deeper analysis.

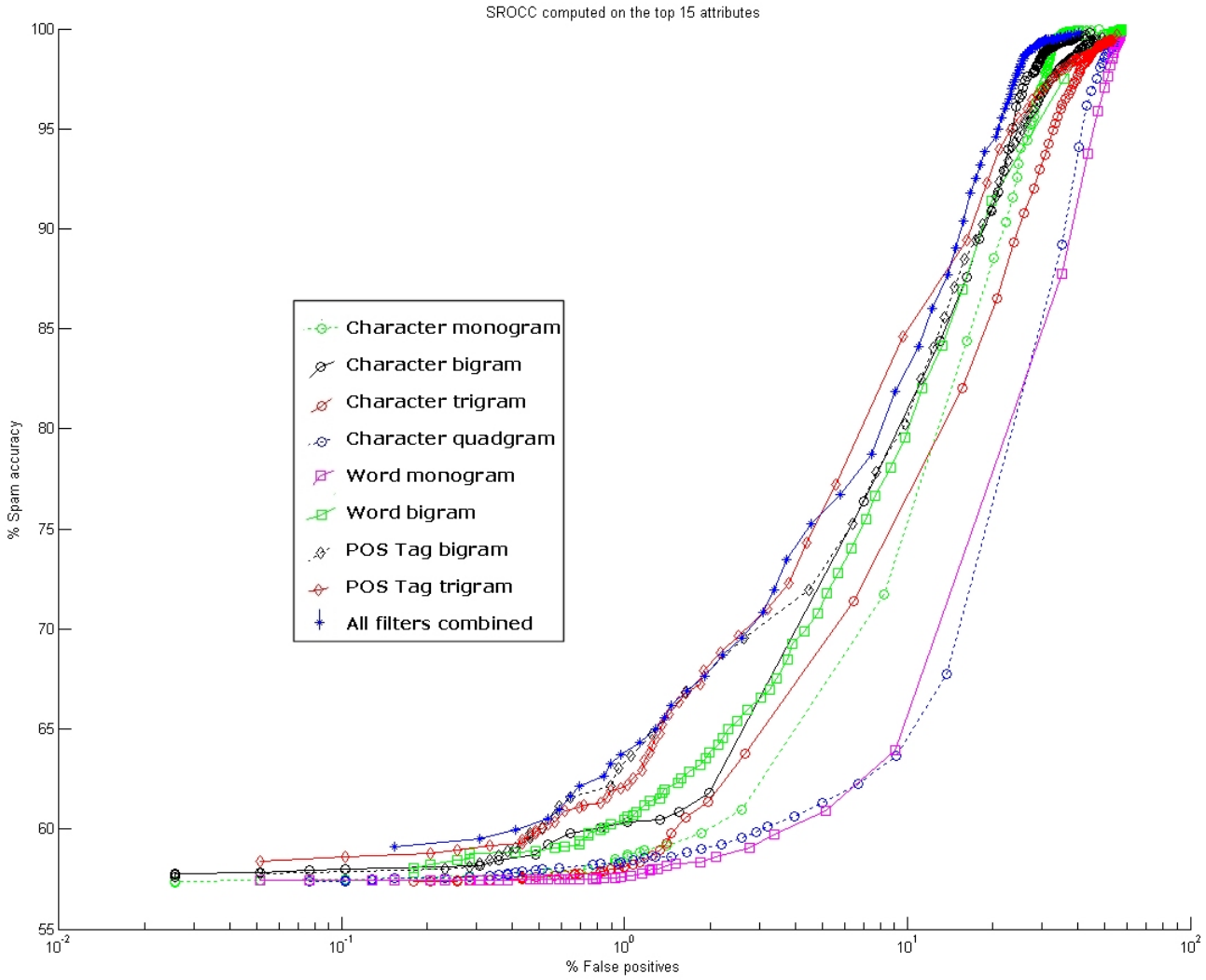

Figure 5.3: The results of individual tests, and all the tests combined, using the top 15 attributes in a message

Character n-grams The character quad-gram frequency analysis proved to be the second best performing module overall, and the best character frequency analysis method. The character fre-

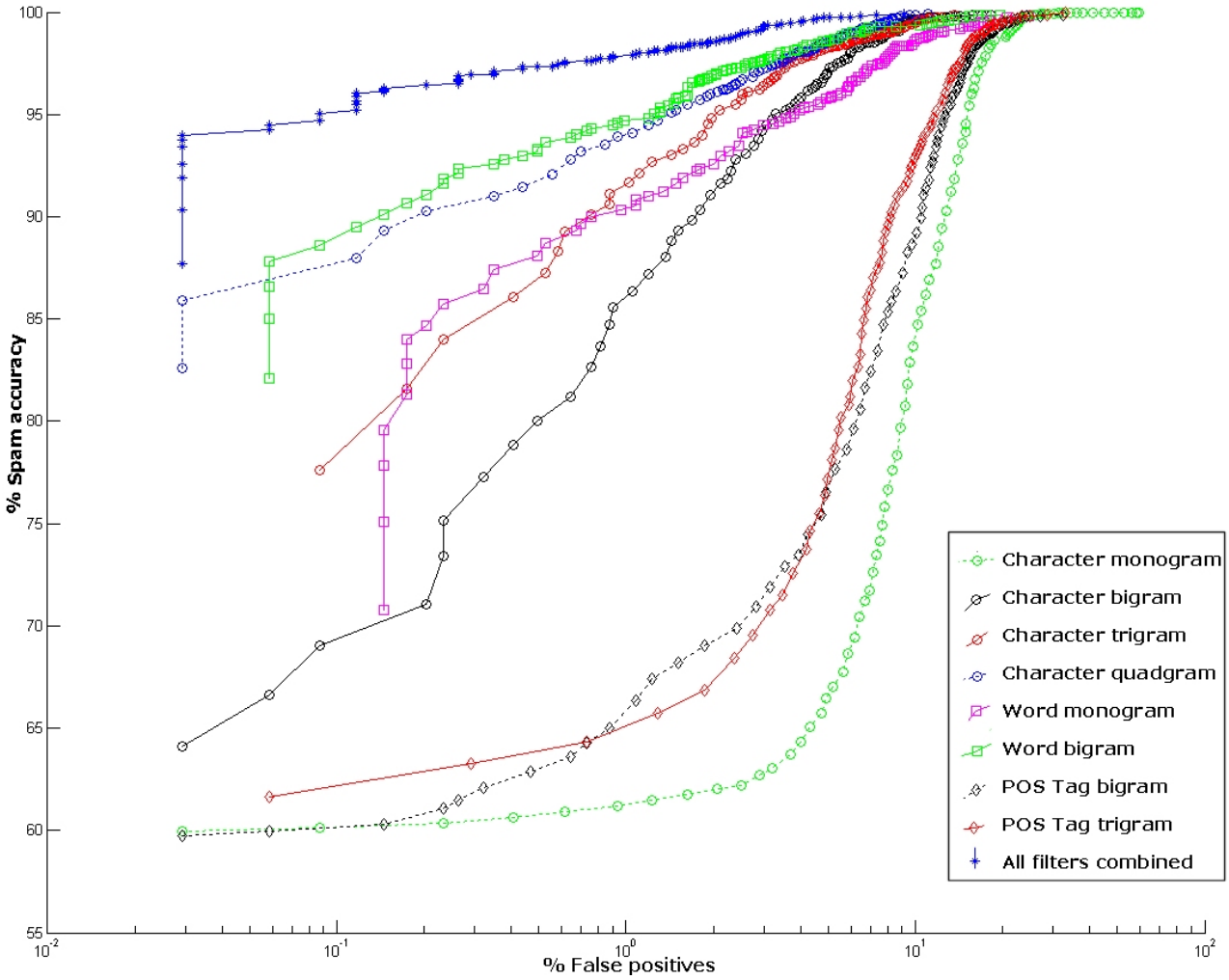

Figure 5.4: The results of individual tests, and all the tests combined, using all the attributes in a message

quency methods tended to perform worse when comparing smaller strings of letter occurrences. This is evident from tri-grams performing better than bi-grams, and mono-grams being the worst module used in the neural net.

It is known in general that character n-grams are very useful in discriminating between different texts[41, 44, 43, 42]. It is also known that larger n-grams generally yield better results, but there does exist a point where larger n-grams begin to degrade in performance. As that point has not been reached in the explored methods, it is something that may be worth investigating further.

Word n-grams Word bi-gram frequency analysis is the most successful module implemented which signifies that the occurrence of word tokens in pairs is a significant feature. The raw results show that the same pairs of words are highly likely to occur in ham e-mail, but the re-occurrence in spam e-mail is only moderate. This can be ascertained from the result tables as while the correlation of ham e-mail is consistently high to the ham profile, this is not the case for the spam messages to the spam profile.

The word mono-gram frequency analysis degrades in performance from the bi-word frequency analysis as much as the character tri-gram does from the character quad-gram. The significance of this is not entirely known, but it is interesting to note.

Part Of Speech (POS) Tag n-grams The POS tag n-grams have not performed as well as conjectured. This implies that spam messages, in comparison with ham messages, do not contain significantly different language constructs despite the numerous symbols within spam messages. The failure of this module is surprising and warrants further investigation, as the points of error could be inherent within the external library or within the processing methods of the AntispamCore. Further conclusions can not be drawn about this module, and the potential it may have had, until the possibility for error is addressed.

5.2.3 Comparison of overall performance between tests

The accuracy of the combined-filter classifier for the test using only the top 15 attributes, and the test which uses all attributes is shown in figure 5.3, and 5.4 respectively. The difference in accuracy between tests is highly pronounced as the graphs demonstrate. This section discusses the reasons for the difference, and the possible limitations behind using a rank order test in comparison to other possible tests.

The test which uses, and ranks, all the features in an e-mail (performance graphed in figure 5.4), notably performs much better than the test which uses the top 15 ranked features in a message (graphed in figure 5.3), primarily due to the fact that all the modules seem to perform better with more data. While this may sound obvious, it is not necessarily always the case, which was the primary motivation for running both tests. Paul Graham's naïve Bayes implementation[30] reportedly works better with fewer attributes to compare. Consequently, Graham's implementation functions using the top 15 significant words in a message, achieving "99.5% accuracy with less than 0.03% false positives." The difference however, is that Graham's filter uses a probabilistic, token based method as a category similarity measurement technique, while the studied implementation uses a rank based method.

A rank based similarity tool, like the Spearman Rank Order (SROCC), may not perform well with smaller portions of the available data with the given methods for multiple reasons. One of the issues may be that the top 15 occurring attributes in most messages, may occur in roughly the same order within the profiles. For instance, the most likely words to occur within any readable English e-mail, would include high occurrences of standard single words like, "`to`", "`of`", and "`a`" , as well as high occurrences of paired words like, "`if you`", "`will be`", and "`on the`"[10]. These examples explain the problems in word frequency analysis, and the character n-gram analysis would suffer the same problems as the most common words, and word combinations, can lead to the most common occurrences of character strings. The results from the "top 15" test appear to reflect this type of problem as for each test the coefficient's to the ham corpus and spam corpus are exceedingly similar.

On the other hand, as the SROCC does prove to be highly accurate with the given methods when all the attributes are used, it can be conjectured that a feature of the given data classification problem is that more data is required by the SROCC to make an accurate classification than is required by probabilistic measurement methods. This implies that rank based techniques are inherently more computationally expensive than probabilistic techniques simply because the number of attributes processed by rank order statistics varies with the size of messages, whereas they don't with probabilistic models.

[10]The examples provided are actual ordered examples from the generated profiles. The profiles are not provided in the appendix due to the size of the profile tables (123,299 for the ham word frequency profile alone)

5.2.4 Comparison of the results by these combined filters with existing methods

The relative success of the combined classifier (CC) will be compared to the methods discussed in chapter 2. The discussion will directly compare the combined classifier against White and black-lists, challenge-response systems, checksum based filtering, naïve Bayes filters, rule-based filters, and SpamGuru. Where results of existing systems are not readily available, the benefits of either system will be compared.

% False positive	% Spam detected	Threshold
0.029129	87.708	0.01
0.029129	90.358	0.02
0.029129	91.902	0.03
0.029129	92.601	0.04
0.029129	93.417	0.05
0.029129	93.766	0.06
0.029129	93.970	0.07
0.058258	94.262	0.08
0.058258	94.495	0.09
0.087387	94.728	0.10
0.087387	95.048	0.11
0.116520	95.223	0.12

Figure 5.5: Sample of the false positive rate in comparison to spam detection rate, from the results generated by the AntispamCore

In comparison with black and white lists, and challenge-response systems (BWCR) the CC would tend to produce less false positives, but has a higher probability of yielding more false negatives. BWCR systems produce results which vary greatly between users. The messages blocked depend entirely on the entries they have in their lists, and a new correspondent may have trouble communicating with them, thus resulting in higher false positives. The spam blocked however, by BWCR systems is potentially 99.99% to the user of the system. The tradeoff with using such a system is that the sender must work in order to communicate with the recipient, but in a way which does not address any of the costs described in figure 2.1 effectively.

Overall, BWCR systems can potentially help one user who only communicates with a few people, but for groups of people that may communicate with new contacts, the combined classifier would be more effective. Additionally, the CC at least addresses 'the social cost' (figure 2.1), whereas BWCR systems exacerbate the overall problem for the rest of the community.

In comparison with checksum based filtering (CBF) the CC seems to be more effective, from the results obtained[24]. However, the results generated by the CBF system is over a much larger sample, and so a direct comparison of results would not be entirely accurate. The implemented CBF system appears to work well in blocking well known spam, but spam with large variances in the text stands a good chance of subverting the algorithm. The difference between that and the CC is that messages with large variances, as the CC is trained, stand a much higher chance of being successfully classified as spam, since the message will most likely bear little resemblance to what the user normally receives.

CBF techniques perform well in potentially addressing 'the inter-ISP cost' and 'the social cost' (figure 2.1) through reduction of known spam between relays, and from displaying in a user's MUA. It may prove beneficial to integrate the CBF algorithm with the CC, as there may be well known spam that is not caught by the CC, and the CC may also help the CBF capture more.

In comparison with naïve Bayesian techniques the CC appears to under perform at the individual user level. The naïve Bayesian classifiers[30, 24] that have been implemented at a personal user level seem to achieve exceedingly high levels of accuracy, however when used for multiple users at a corporate level, the same Bayesian classifiers do not appear to function as well[34, 64]. Users of the the naïve Bayes implementation in large scale classifiers like SpamAssassin have indicated that it "has recently shown to be ineffective in providing accurate results due to spammers adding large amounts of generic text to e-mails"[64]. It is conjectured therefore that for the individual user a naïve Bayes implementation functions well because there are only certain keywords which would identify their message as ham, and very few spam messages are likely to have those words; whereas when multiple users use the same filter, the number, and range, of keywords which identify a message as ham increases, thereby increasing the probability that a generic text will contain some of those words (see figure 2.5).

When the CC is compared to large scale, corporate implementations of naïve Bayes classifiers, it can be seen to perform better[34, 64]. This is likely to be due to the fact that more filters are being used in determining the message's category, and that it is not the most significant keywords which have the most impact on classification in the CC. The CC uses all the tokens in the message, and so the majority of the content must be similar to the user's profile, rather than just generic. Although, it may be possible to reap the benefits of the Bayesian classifier as SpamGuru has, but it may also prove beneficial to maintain independent profiles for each recipient when using Bayes.

Overall, the results from other tests[34, 64] appear to corroborate the claims of [24] about naïve Bayes being ideal for an individual but not for many. The CC however, appears to work well when dealing with multiple different people, and to enhance its effectiveness, it may be interesting to implement a Bayes filter unique to each individual, and combine the results it produces with the other methods.

In comparison with rule based filters like SpamAssassin it is hard to ascertain which one performs better. Tests performed on SpamAssassin have yielded varying results[24, 46], and it is conceivable that it is because of the way the classifier is configured. While some claim to have achieved high accuracy rates[64], it is difficult to gauge as the settings between each configuration are likely to vary and an accurate number of false positives is unlikely to be known[34]. Hence, it may be possible to claim that the combined classifier performs as well as SpamAssassin, as the results depicted in figure 5.5 incite confidence.

The CC has the benefit over SpamAssassin of not needing sophisticated administration, but simple training; SpamAssassin does however, maintain the advantage of being functional out-of-the-box without any lead time. Both classifiers address the same problem, "the social cost" (figure 2.1) at the level of corporate mail server, where a moderately large number of users with a general common interest, are collecting mail from the same point.

In summary, it appears as though the combined classifier may present a viable alternative to the rule based classifier, and thus present administrators the option of using a classifier which can be, and requires to be, manually modified; or with a classifier that is less configurable and adapts by itself. Indeed, it may be possible to combine both, or integrate the techniques used

in the CC into SpamAssassin, as SpamAssassin uses a perceptron, like the CC, to adjust the weightings of tests.

In comparison with SpamGuru the initial results look promising, however the corpora tested on by the CC is significantly smaller in size than that used by SpamGuru, and as such any comparisons based on statistics are likely to be unfair and unreliable. In order to make a value judgement, a corpora of the same size needs to be used to test the CC.

In terms of structure, SpamGuru takes a slightly different approach in how it handles the output from various modules[34], but the concept of combining multiple filters is the same. Both classifiers have shown that a combination of filters ultimately increases the accuracy of classification. IBM's SpamGuru has more sophisticated modules than the CC, as well as overall work-flow, which make it a better option for deployment (once available). The modules, and architecture, allow for SpamGuru to be dynamically modified as well cope with infrastructural changes out-of-the-box.

Therefore, while both systems have reaped benefits from combining multiple filters using a simple perceptron, SpamGuru is better equipped to cope with changes in both spam and e-mail infrastructure modifications.

5.2.5 Analysis of potential result bias

Given the relative success of the filter, it is necessary to investigate if the experimental procedure has somehow biased the results. In the process of investigation it is valuable to analyse the experimental procedure for potential flaws. This analytical process is presented here in order to demonstrate that the results have not been unfairly biased by the experimental procedure, but that there does exist further work to incite more confidence in the result claims. The section begins by analysing potential problems with the experimental corpora, followed by a brief analysis of the implementation. Concluding with the decision that in order to incite more confidence in the results, a larger corpora is necessary, and that tests should be run with several different types of ham corpora from different organisations.

It was known from the outset that the spam corpus was more likely to contain HTML than the ham corpus, which is why the *message harvester* was biased towards text components in e-mail messages (see section 5.1.2). The motivation for this is that if the spam corpus were to contain mostly HTML and the ham corpus has none, or very little, then the combined classifier could be a simple HTML detector. As a result, it was considered important to analyse the datasets, searching for known HTML tags, and figure 5.6 represents the findings. The HTML tags searched for ranged from font definitions (‘<font’), table definitions (‘td’)[11], break-lines (‘
’), HTML declarations (’<html>’), and numerous other standard tags, along with ones that were taken from the spam word database. Figure 5.6 demonstrates that the combined classifier could not possibly be classifying based purely on HTML as over 50% of the spam corpus does not contain recognised HTML tags. This implies that while HTML may have helped certain filters, it could not form the complete basis for classification.

Another aspect worth noting is that while the corpus could not contain any duplicate body text, or unique identifier, as the database was configured to reject identical entries of these fields; there was no similarity check in place, so minor variations to the message would have been accepted. This however is not an integrity issue, as much spam is likely to be varied slightly from other spam, and a feature of a spam detector should be to identify such messages. So, while this could potentially

[11]It was found that the 3% of single tag occurrence in the ham corpus was due entirely to the ‘td’ tag being searched for. The occurrence of the character string ‘td’ are found in multiple word formations used by Enron employees

HTML level	% of spam	% of ham	Description
Heavy	13.8%	0.00%	*Contains at least 3 recognised tags*
Moderate	19.2%	0.00%	*Contains 2 recognised tags*
Light	15.4%	3.07%	*Contains 1 recognised tag*
None	51.6%	96.93%	*Contains no recognised tags*

Figure 5.6: Measurement of the number of HTML tags in the training and testing datasets

be considered a caveat with the corpus, rather than simply reject messages based on some distance measure, a more conscientious approach would be to simply increase the size of the spam corpus. The larger the spam corpus, the less likely several messages are going to be closely representative of each other.

It has been assumed in this study that the style of writing of people, and the style of spam changes with time. For this reason it would be necessary to choose samples that span the same amount of time in both corpora. This was considered in the initial selection phase of the corpora, and so both samples cover an eight month duration in time. This is further detailed in section 5.1.1.

Moreover, analysis of the implementation yields that all data was treated the same. Both corpora were reformatted by the same harvester to be stored in the same table in the database under identical field headings. Also, profile generation was carried out by the same procedures for ham and spam; the only difference was the table the data was stored in as that depended on the preclassification tag (see figure 4.3). Additionally, preprocessing was executed without bias for data orientation, it was only executed differently depending on whether the data belonged to the `training` or `testing` dataset which necessarily contained samples of both ham and spam. Furthermore, the selection for samples for the datasets was completely randomised as no technique was employed for selection other than a target dataset size.

From this analysis, it can be concluded that there is no noticeable bias in the data used for experimentation. However, it would be possible to draw more accurate results if more than one ham corpus was used, as unknown features of a singe ham corpus may influence the results. Unfortunately due to privacy constraints, and time constraints (the analyses take approximately eight hours for every 1000 e-mails analysed) this was not possible.

It should also be noted that the other methods of spam detection which are being compared, should ideally be compared on the same set of test data - so as to avoid a bias in the results due to a feature of the data. Again, due to time constraints this has not been possible. As such, for the purposes of this investigation, as a result of the corpora analysis it is assumed that the test data sets used for any of the measurements (3rd party benchmarks included) contain no anomalies that would bias the results.

6. Conclusions

From the analyses of the results produced by the conducted experiments, it is possible to conclude that the accurate identification of unsolicited electronic mail is possible through multiple text classification techniques. This study provided a background into the existing methods used to address the current spam issue; and revealed how the majority of those methods only use one style of message analysis. Following this, it was shown that combining filters through an appropriate aggregator yields better results than the use of an individual filter.

Although not a new idea, content based filtering, demonstrated by filter analyses and the results of these experiments, is effective. Furthermore, this investigation has proved that the combination of multiple text analysis techniques greatly enhances the accuracy of a spam classifier.

Additionally, it has also been shown that for classification to be robust and accurate that the knowledge-base used in generating results must continually adapt. The filters based on static knowledge-bases that attempted to only use adaptive result evaluation techniques, like the neural net, to identify correlations did not achieve good results. SpamAssassin to some extent corroborates this issue as to maintain it's effectiveness the rule-set used needs to be continually manually updated, even though it uses a perceptron to manage the weightings of each rule.

Within this study it was found that word bi-gram and character quad-gram frequency analysis perform exceedingly well when used with the Spearman Rank Correlation. This potentially indicates that the subject matter, and general authorship style between spam messages and ham messages is fundamentally different, and so is differentiable through basic techniques. It should be noted however, that authorship attribution and subject differentiation through textual analysis techniques, has never achieved 100% accuracy.

Ultimately, this study has revealed that no individual method, even multiple-filter based classifiers, will successfully reduce the costs incurred by spam. Chapter 2 presented the different levels at which spam poses problems and identified the areas each classifier addressed. The only classifier which successfully proposed to address all areas was IBM's unreleased SpamGuru. As previously discussed, SpamGuru's approach is not simply an implementation of multiple filters, but rather an extensible adaptation to existing infrastructure that supports phased deployment.

Unfortunately, further investigation into SpamGuru is not accessible as it is private research held by IBM. This makes a comparative analysis to it very difficult to reliably quantify. The comparisons against the other methods is also difficult to confidently confirm, as the corpora used by those studies varies greatly between them. Additionally, the other methods could not be executed over the corpora used for this study due to time constraints and unforeseen problems with file I/O efficiency.

The implementation of the AntispamCore did not function as efficiently as was projected. Due to inefficient data handling, the core took approximately eight hours to process 1000 messages. This encumbered the testing process of the core, and caused unavoidable delays throughout development. These time delays would have been experienced by any other method that was plugged into the core as the delays were experienced at the level of message retrieval, rather than analysis. MySQL did not function as efficiently as anticipated.

However, despite the limitations of my research, the results very strongly indicate that a combination of filter methods is a tenable solution to spam. Given the increasing problem that spam poses to businesses and home users, and the hitherto unseen ubiquity of the internet, the proposed concept of

classifier aggregation certainly contributes to the study of spam categorisation techniques, and merits further investigation.

7. Future work

In order to address the problems in analysis, future work calls for the testing of existing methods on the corpora used by this study. Additional work would see the implementational issues addressed so that storage and processing times are dramatically reduced to make the system feasible for testing with much larger corpora.

Additionally, the static knowledgebase methods require to be adapted to support dynamic dictionaries so that further information concerning the difference in word use in spam and ham be ascertained.

Furthermore, it will be interesting to observe the difference in classification accuracy when support vector machines are used, as opposed to neural networks.

In the development of this system it was realised that spam may not necessarily be a binary classifiable problem. This is may be due to the concept that people may see some messages as more important than others, and certain messages are undesirable in their inbox. However, this does not necessarily mean they do not want all of those messages, rather some may be of a much lower priority. This extends the problem of spam to a more general text categorisation problem, rather than what it is viewed as today. Rather than a technical problem, this calls for a social study on the general user's approach to handling email. Such a study may give further insight into how the classification of messages would take place more accurately.

Appendix A. Spam examples

A.1 Email headers

```
From - Wed Feb 23 16:45:20 2005
X-Account-Key: account2
X-UIDL: UID9417-1070005951
X-Mozilla-Status: 0011
X-Mozilla-Status2: 10000000
Return-Path: <esha_munot@yahoo.com>
Delivered-To: ka_force-kameel-kaiesh@kaiesh.com
Received: (qmail 71018 invoked from network);
        23 Feb 2005 16:47:35 -0000
Received: from unknown (HELO ptb-mxcore01.plus.net) (212.159.14.215)
  by ptb-mailstore02.plus.net with SMTP; 23 Feb 2005 16:47:35 -0000
Received: from pih-mxlast02.plus.net ([212.159.6.18])
         by ptb-mxcore01.plus.net with esmtp (Exim)
                id 1D3zhY-0003He-MU
        for kaiesh@kaiesh.com; Wed, 23 Feb 2005 16:49:28 +0000
Received: from [202.134.186.254] (helo=kaiesh.com)
        by pih-mxlast02.plus.net with esmtp (Exim 4.30)
        id 1D3zfT-0003lc-B8
        for kaiesh@kaiesh.com; Wed, 23 Feb 2005 16:47:20 +0000
From: esha_munot@yahoo.com
To: kaiesh@kaiesh.com
Subject: Re: patched
Date: Wed, 23 Feb 2005 22:17:46 +0530
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="----=_NextPart_000_0016----=_NextPart_000_0016"
```

A.2 A short message

```
This is a multi-part message in MIME format.

------=_NextPart_000_0016----=_NextPart_000_0016
Content-Type: text/plain;
        charset="Windows-1252"
Content-Transfer-Encoding: 7bit

I have attached your document.


------=_NextPart_000_0016----=_NextPart_000_0016
Content-Type: application/octet-stream;
        name="bill.zm9"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="bill.zm9"

```

```
UEsFBgAAAAAAAAAAAAAAAAAAAAAAAA==
------=_NextPart_000_0016----=_NextPart_000_0016--
```

Appendix B. Learning methods

B.1 Support Vector Machines

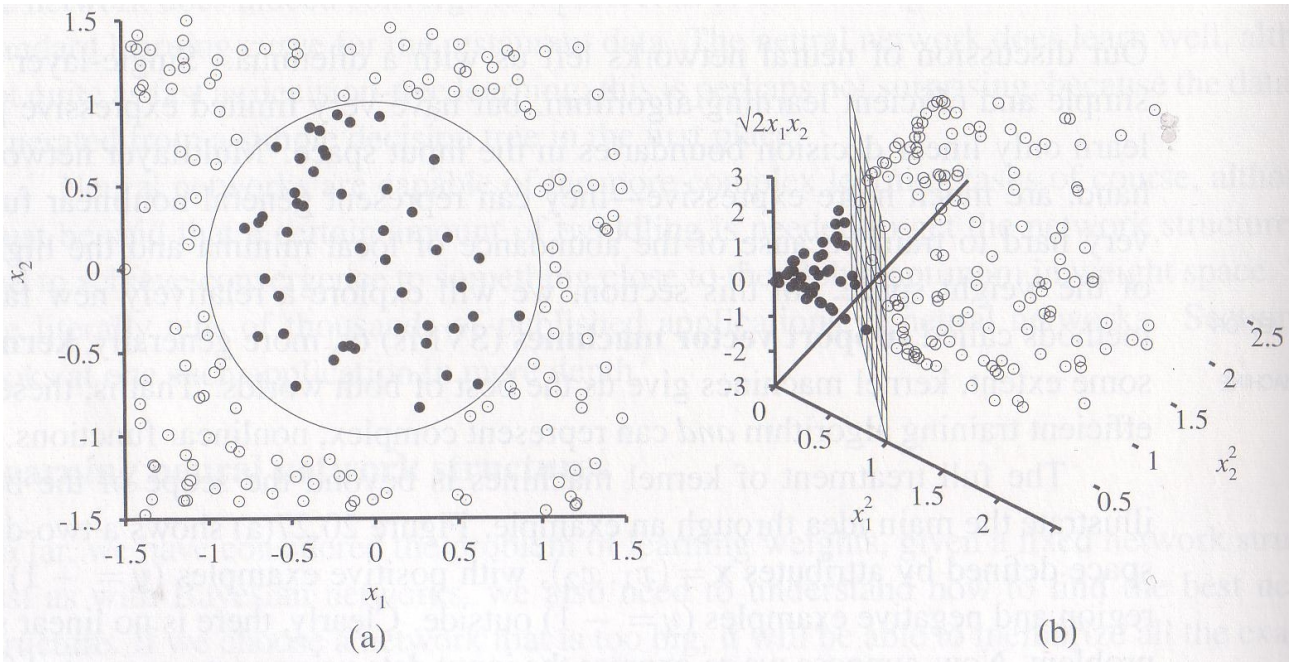

Figure B.1: (a) A two-dimensional training with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 <= 1$, is also shown. (b) The same data after mapping into a three dimensional input space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in (a) becomes a linear decision boundary in three dimensions.

Appendix C. Character n-gram frequency analyser

The results produced by the character n-gram frequency analyser

Character mono-grams

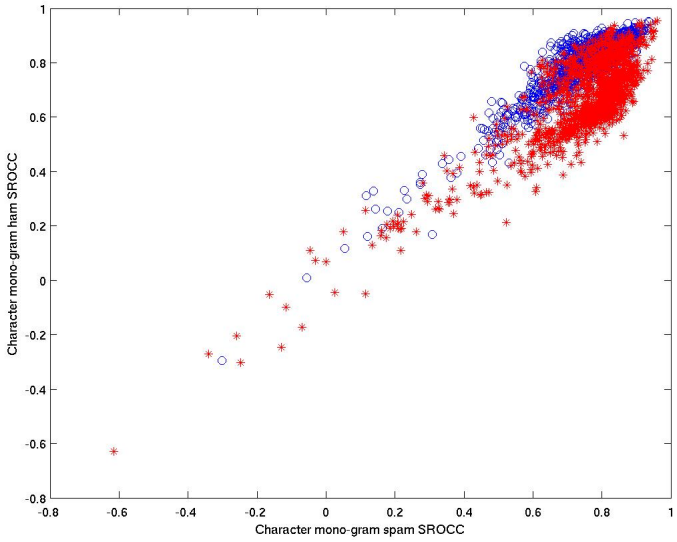

Figure C.1: The SROCC results for Char mono-grams, *red represents spam, and blue represents ham*

Character bi-grams

Character tri-grams

Character quad-grams

C.0.0.1 Character counter

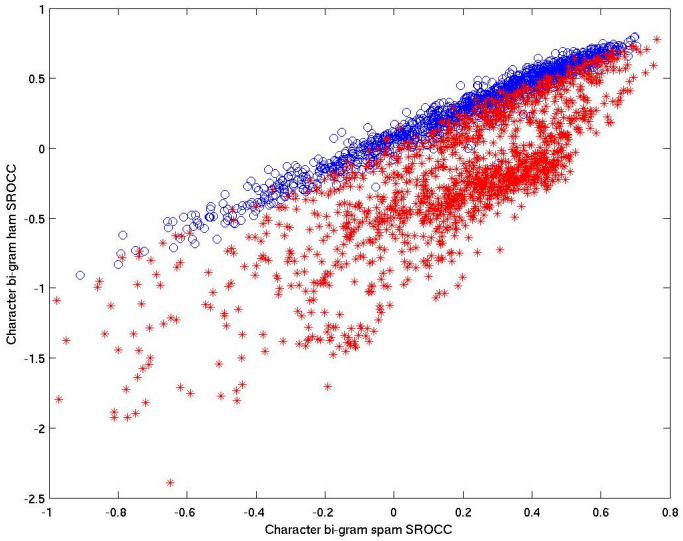

Figure C.2: The SROCC results for Char bi-grams, *red represents spam, and blue represents ham*

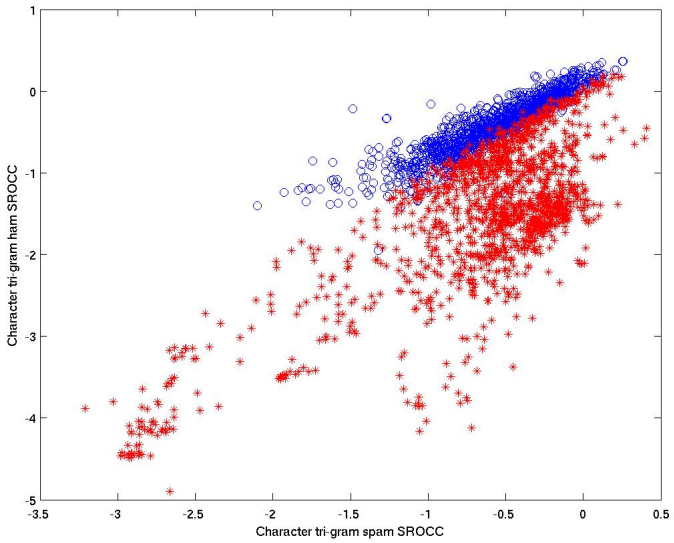

Figure C.3: The SROCC results for Char tri-grams, *red represents spam, and blue represents ham*

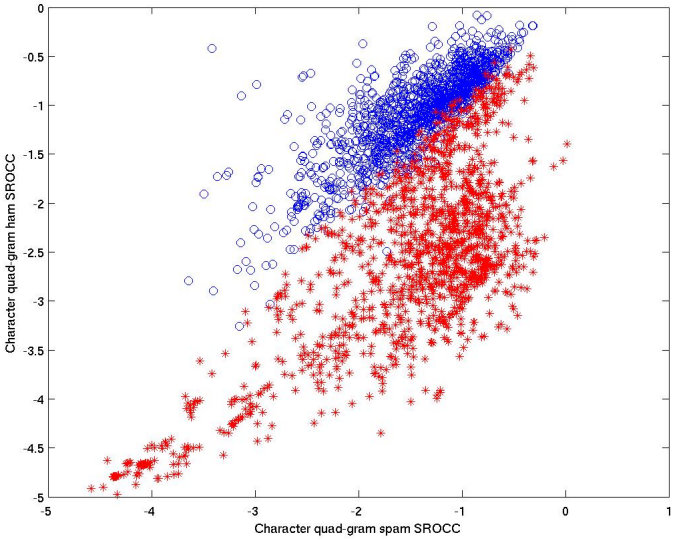

Figure C.4: The SROCC results for Char quad-grams, *red represents spam, and blue represents ham*

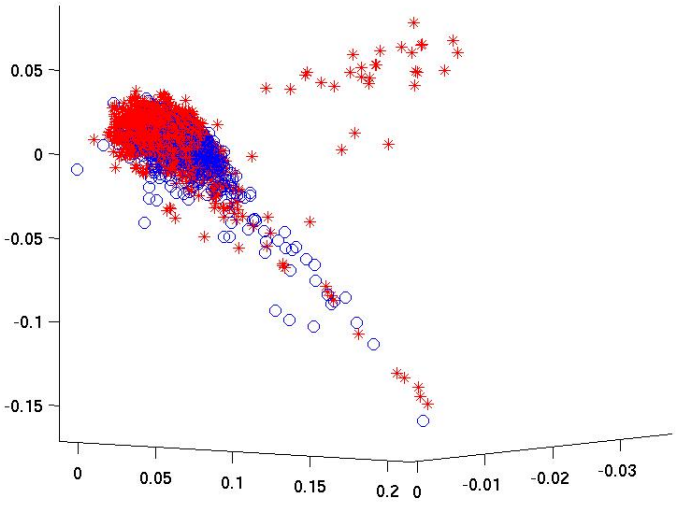

Figure C.5: The distribution of single character ratios within messages, *red represents spam, and blue represents ham, the axes are the 3 most significant vectors found through PCA*

Bibliography

[1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 2003.

[2] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Pearson Education Inc., Upper Saddle River, New Jersey 07458, second edition, 2003.

[3] J. Goodman and R. Rounthwaite. Stopping outgoing spam. *Proceedings of the 5th ACM conference on Electronic commerce*, Session 2:30 – 39, 2004.

[4] D. Sorkin. Spam laws. http://www.spamlaws.com/, December 2004.

[5] C. Wong. A study of mass-mailing worms. In *Proceedings of the 2004 ACM workshop on Rapid malcode*, volume Session 1, pages 1 – 10. ACM, 2004.

[6] Sophos. Bobax worm turns computers into spam zombies. Technical report, Sophos, May 2004. http://www.sophos.com/virusinfo/articles/bobax.html.

[7] LURHQ Threat Intelligence Group. Sobig.a and the spam you received today. Technical report, LURHQ, 2003.

[8] J. B. Postel. Simple mail transfer protocol (rfc788). Technical report, University of Southern California, USC, 4676 Admiralty Way, Marina del Rey, California 90291, USA, November 1981.

[9] Sendmail. Sender authentication: The key to fighting email fraud. Technical report, Sendmail, 2004.

[10] M. Wong. Sender authentication, what to do. *Messaging Anti-Abuse Working Group*, December 2004. http://spf.pobox.com/whitepaper.pdf.

[11] P. G. Capek, B. Leiba, M. N. Wegman, and S. E. Fahlman. Charity begins at... your mail program. Technical report, IBM Thomas J. Watson Research Center, Hawthorne, NY 10532, USA, 2003.

[12] R. Rivest and A. Shamir. Payword and micromint: Two simple micropayment schemes. In *Proceedings of the Fourth Cambridge Security Protocols Workshop*, 2001.

[13] M. Abadi, M. Burrows, M. Manasse, F. Dabek, and T. Wobber. Bankable postage for network services. In *Proceedings of the 8th Asian Computing Science Conference*, December 2003.

[14] D. A. Turner and D. M. Havey. Controlling spam through lightweight currency. In *Proceedings of the Hawaii International Conference on Computer Sciences*, January 2004.

[15] A. Back. Hashcash - a denial of service counter-measure. http://citeseer.ist.psu.edu/back02hashcash.html, August 2002.

[16] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO 2003)*, 2003.

[17] B. Schneier and N Ferguson. *Practical Cryptography*. Wiley Publishing Inc, first edition, 2003.

[18] W. Diffie and M. E. Hellman. New directions in cryptography. Technical Report 6, IEEE Transactions on Information Theory, 1976.

[19] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *Proceedings of the 10th Annual Network and distributed System Security Symposium*, pages 25 – 39, February 2003.

[20] M. Delany. Domain-based email authentication using public-keys advertised in the dns (domainkeys) [ietf internet draft]. http://www.ietf.org/internet-drafts/draft-delany-domainkeys-base-01.txt, August 2004.

[21] Spamhaus. Increasing spam threat from proxy hijackers. http://www.spamhaus.org/news.lasso?article=156, February 2005.

[22] E. Z. Ye. Technical report tr2002-417. Technical report, Department of Computer Science, Dartmouth College, 2002.

[23] B. McWilliams. Spf not poisonous to phish. Technical report, O'Reilly Network, September 2004. http://www.oreillynet.com/pub/a/network/2004/09/28/spf.html.

[24] F. Garcia, J. Hoepman, and J. van Nieuwenhuizen. Spam filter analysis. Technical report, University of Nijmegen, the Netherlands, 2004. University of Nijmegen, the Netherlands.

[25] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *Proceedings of Eurocrypt*, 2003.

[26] G. Mori and J. Malik. Recognizing objects in adversarial clutter – breaking a visual captcha. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, June 2003.

[27] A. Kolcz, A. Chowdhury, and J. Alspector. The impact of feature selection on signature-driven spam detection. In *Proceedings of the First Conference on Email and Anti-Spam*, Mountain View, CA, USA, July 2004.

[28] Niels Provos. A virtual honeypot framework. Technical report, Center for Information Technology Integration, University of Michigan and Google Inc., 2003.

[29] N. Krawetz. Anti-honeypot technology. *Security and Privacy Magazine, IEEE*, 2(1):76 – 79, Jan - Feb 2004.

[30] P. Graham. Better bayesian filtering. In *Proceedings of the First Annual Spam Conference*, MIT, Cambridge, Mass., USA, January 2003.

[31] I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. Spyropoulos. An evaluation of naive bayesian anti-spam filtering, 2000.

[32] D. Kohn. Spamassassin: Howscoresareassgined. http://wiki.apache.org/spamassassin/HowScoresAreAssigned, July 2004. (last verified on 09-Feb-2005).

[33] Cormac O'Brien and Carl Vogel. Spam filters: bayes vs. chi-squared; letters vs. words. In *ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies*, pages 291–296. Trinity College Dublin, 2003.

[34] R. Segal, J. Crawford, J. Kephart, and B. Leiba. Spamguru: An enterprise anti-spam filtering system. In *Proceedings of the First Conference on Email and Anti-Spam*, 2004.

[35] Tong Zhang and Frank J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4(1):5–31, 2001.

[36] Mahran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Proceedings of AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[37] Isidore Rigoustsos and Tien Huynh. Chung-kwei: a pattern-discovery-based system for the automatic identification of unsolicited e-mail messages (spam). In *Proceeds of the First Conference on E-mail and Anti-Spam*, 2004.

[38] P. Resnick. Internet message format (rfc 2822). Rfc, Network Working Group, April 2001. http://rfc.sunsite.dk/rfc/rfc2822.html.

[39] B. D. Sheth. A learning approach to personalized information filtering. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1994.

[40] C. Orasan and R. Krishnamurthy. A corpus-based investigation of junk emails. In *Proceedings of the LREC*, Las Palmas, Spain, 2002.

[41] R. Clement and D. Sharp. Ngram and bayesian classification of documents for topic and authorship. *Literary and Linguistic Computing*, 18(4), 2003.

[42] F. Peng, D. Schuurmans, V. Keselj, and S. Wang. Language independent authorship attribution using character level language models. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 267 – 274, Budapest, 2003.

[43] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.

[44] V. Keselj, F. Peng, N. Cercone, and C. Thomas. N-gram based author profiles for authorship attribution. Pacific Association for Computational Linguistics, 2003.

[45] Thomas N. Turba. Checking for spelling and typographical errors in computer-based text. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, pages 51–60, 1981.

[46] C. O'Brien and C. Vogel. Comparing spamassassin with cbdf email filtering. Computational Linguistics Group & Centre for Computing and Language Studies, Trinity College, University of Dublin, 2003.

[47] J. Furnkranz. A sudy using n-gram features for text categorization. Technical Report OEFAI-TR-98-30, Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010 Wien, Austria, 1998.

[48] R. E. Madsen, J. Larsen, and L. K. Hansen. Part-of-speech enhanced context recognition. Technical report, Department of Mathematical Modelling, Technical University of Denmark, 2004.

[49] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330, June 1993.

[50] J. C. Reynar and A. Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, pages 16–19. Morgan Kaufmann Publishers Inc., 1997.

[51] Mark Stevenson and Robert Gaizauskas. Experiments on sentence boundary detection. In *Proceedings of the sixth conference on Applied natural language processing*, pages 84–89. Morgan Kaufmann Publishers Inc., 2000.

[52] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Proceedings of Computer Vision and Pattern Recognition, IEEE Computer Society Conference*, Media Lab., MIT, Cambridge, MA, 1991.

[53] E. W. Weisstein. Spearman rank correlation coefficient. From MathWorld, 1904. A Wolfram Web Resource. http://mathworld.wolfram.com/SpearmanRankCorrelationCoefficient.html.

[54] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155. ACM Press, 1998.

[55] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[56] I Gebert. Open-source database management systems in small and medium-sized companies. Technical report, University of Rostock, 2003.

[57] H. Liu. Montylingua: An end-to-end natural language processor with common sense. http://web.media.mit.edu/ hugo/montylingua, 2004.

[58] Steve Renals. Lectures on com336: Single-layer networks. Lectured in The University of Sheffield, Department of Computer Science, 2004.

[59] Ian Nabney and Christopher Bishop. Netlab: A neural net toolbox for matlab. http://www.ncrg.aston.ac.uk/netlab/.

[60] Spamarchive.org. http://www.spamarchive.org.

[61] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning*, volume 3201 / 2004, Pisa, Italy, September 2004.

[62] N. Borenstein. Ibm anti-spam research: Anti-spam filtering. http://www.research.ibm.com/spam/filtering.html, February 2005. (date last verified).

[63] S. Ahmed and F. Mithun. Word stemming to enhance spam filtering. In *Proceedings of the First Conference on Email and Anti-Spam*, Mountain View, CA, USA, July 2004.

[64] DNS Ltd. In conversations with a representative, February 2005. dns ltd, 83 Princes St., Edinburgh, EH2 2ER, United Kingdom.