2011-07-13

Naive Bayesian Spam Filters for Log File Analysis

Russel William Havens
Brigham Young University - Provo

Naïve Bayesian Spam Filters

for Log File Analysis

Russel W. Havens

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Barry Lunt, Chair
Joseph J. Ekstrom
Chia-Chi Teng

School of Technology

Brigham Young University

August 2011

ABSTRACT

Naïve Bayesian Spam Filters
for Log File Analysis

Russel W. Havens
School of Technology, BYU
Master of Science

As computer system usage grows in our world, system administrators need better visibility into the workings of computer systems, especially when those systems have problems or go down. Most system components, from hardware, through OS, to application server and application, write log files of some sort, be it system-standardized logs such syslog or application specific logs. These logs very often contain valuable clues to the nature of system problems and outages, but their verbosity can make them difficult to utilize. Statistical data mining methods could help in filtering and classifying log entries, but these tools are often out of the reach of administrators.

This research tests the effectiveness of three off-the-shelf Bayesian spam email filters (SpamAssassin, SpamBayes and Bogofilter) for effectiveness as log entry classifiers. A simple scoring system, the Filter Effectiveness Scale (FES), is proposed and used to compare these filters. These filters are tested in three stages: 1) the filters were tested with the SpamAssassin corpus, with various manipulations made to the messages, 2) the filters were tested for their ability to differentiate two types of log entries taken from actual production systems, and 3) the filters were trained on log entries from actual system outages and then tested on effectiveness for finding similar outages via the log files.

For stage 1, messages were tested with normalized bodies, normalized headers and with each sentence from each message body as a separate message with a standardized message. The impact of each manipulation is presented. For stages 2 and 3, log entries were tested with digits normalized to zeros, with words chained together to various lengths and one or all levels of word chains used together. The impacts of these manipulations are presented.

In each of these stages, it was found that these widely available Bayesian content filters were effective in differentiating log entries. Tables of correct match percentages or score graphs, according to the nature of tests and numbers of entries are presented, are presented, and FES scores are assigned to the filters according to the attributes impacting their effectiveness.

This research leads to the suggestion that simple, off-the-shelf Bayesian content filters can be used to assist system administrators and log mining systems in sifting log entries to find entries related to known conditions (for which there are example log entries), and to exclude outages which are not related to specific known entry sets.

Keywords: Russel Havens, log file analysis, Bayesian content filter, spam filter, SpamAssassin, SpamBayes, Bogofilter, filter effectiveness scale, fes

ACKNOWLEDGMENTS

I wish to express my appreciation to my committee chair, Dr. Barry Lunt, who guided me into the program and through the final painful process of writing and editing. I am also thankful for Dr. Mike Miles, because of whom I was able to start the daunting writing task, and for Dr. Dennis Eggett, who helped me overcome my fear of statistics and think analytically. I also wish to thank my other committee members, Dr. Chia-Chi Teng, for helping me see what publication is like and Dr. J Ekstrom for his zeal for learning and inspirationally creative "wouldn't that be an interesting study" line of thinking. I am also grateful for Ruth Ann Lowe's above-and-beyond support through the bureaucratic minutiae of academic effort at a large university.

More personally, my good wife, Lisa, and my three children, Rachel, Catherine and Daniel, have paid the highest price for this academic endeavor, and I am most grateful to them for their support and sacrifice. I am thankful for parents and grandparents who love learning and passed that love to me. Finally, I am grateful to Him from Whom all blessings come, my Father in Heaven, and to Him through Whom all blessings become most meaningful, Jesus Christ.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

With the proliferation of computer technologies in the workplace, many organizations have become more and more dependent on computers. Knowledge workers, accountants, management, sales and marketing people, even line workers use computers every day to do their jobs. Computers have become indispensable for many of these workers, and thus for their organizations. When critical computer systems crash or have end-user impacting issues, these employees often cannot do their jobs bringing in revenue and providing services. For many organizations, this can add up to thousands to millions of dollars in revenue per hour depending on the nature of the outage. To add insult to injury, the company still pays these workers to sit around and wait for their tools to come back online. Downtime is expensive, so organizations work hard to minimize it by preventing outages in the first place, or quickly mitigating the ones that do happen. Even a few minutes less downtime per incident can add up to huge savings for some organizations.

When computer administrators are working to troubleshoot an issue, some of their most valuable assets are log files written by the hardware, operating systems and applications that comprise the system. These log files often contain clues pointing to the nature of the problem and give administrators insights into how the problem can be quickly resolved. Unfortunately, even in a mid-sized computer data center, servers produce too many lines of logs for administrators to read them all. Though these log entries often contain valuable troubleshooting

information, the volume of entries means that they are generally only used for immediate reactive troubleshooting and root cause analysis. If there were some way to quickly and easily separate out log entries that warn of system problems, many problems could be more quickly resolved and some may even be prevented.

1.1 **Log Analysis – Research Question**

Log analysis has been an active area of study for some time, with a number of approaches being attempted with varying levels of success. Some of those approaches, filtering and data clustering for example, have been commercialized while others have remained academic research projects. One of the challenges many analytical approaches have is that while they can be quite effective, they are complicated and not immediately usable outside the realm of the statistically enlightened. These sorts of tools will eventually find their way into commercial or open source products, but in the mean time, problems are occurring and valuable clues to those problems are being ignored because of the vast number of log entries that obscure them.

This leads to the following research question: is it possible to find a widely available, advanced filtering and data clustering technology that is useful for log analysis?

Because of the growth in unwanted e-mails, commonly called "spam," many spam filters, such as Spam Assassin, SpamBayes and the like utilize naïve Bayesian content filters to categorize e-mails as spam or non-spam. It is hypothesized that such filters can be trained to differentiate system log entries, such as those produced through syslog or other systems, and that the filtered log entries can be used to predict Linux and application problems.

For the purposes of this research, a log file is a system message file generated by a server's firmware, operating system or application software. Log entries in these files often have

minimal structure, which makes automated analysis challenging. One of the most common logging systems is syslog, which was once a loosely defined de facto standard, but was formalized considerably in IETF RFC 5424, as of March 2009 (Gerhards n.d.). Syslog entries have traditionally comprised a date, severity (ranging from debug to critical), facility (indicating the type of service, such as kernel, mail, clock, etc.) and usually a source system hostname and program, though these were often omitted. RFC 5424 includes more useful information such as hostname, app_id, msgid (indicating type of message), and a number of other useful values. As this more formal standard is adopted by the various communities which use syslog, these records will become easier to analyze and more useful for troubleshooting and problem detection.

Syslog is used by many Unix-like operating systems, including Linux, which is an open source operating system, the kernel of which was written by Linus Torvalds in the early 1990's (Torvalds n.d.). Linux makes use of many GNU tools, making it very Unix-like itself. It is quite popular due to its flexibility, design simplicity, security and robustness.

Many applications also write log files of their own. Most often, these follow a structure which is quite similar to that of Syslog messages, with a time-date stamp, severity, source program name and message body.

1.2 **Bayesian Classifiers**

According to I. Rish, "Bayesian classifiers assign the most likely class to a given example described by its feature vector" (Rish 2001). Making strong assumptions about the independence of the classes simplifies the technique, making the classifier "naïve." Naïve Bayesian spam filters use Bayesian classifiers to categorize e-mails as spam or non-spam. In essence, one trains the filter by giving it spam and non-spam messages; the filter takes the

likelihoods of the various words to appear in the two message types. Then, as other e-mails are analyzed with the trained filter, the likelihoods of all the various words to be in the spam or non-spam sets are combined using Bayesian methods, effectively combining the various probabilities. This gives a richer classification than if the appearance of a single word threw the message into the spam or non-spam category. Generally, the larger the training set and the greater the coverage of terms from the messages to be analyzed, the more effective the filter will be.

SpamAssassin (Apache Foundation n.d.) is an open source e-mail spam analysis program which includes Bayesian filtering and other analysis tools. It originated from work done by Justin Mason and, earlier, Mark Jeftovic. It uses several techniques for detecting spam, one if which is a Bayesian filter which can be trained by system users. SpamAssassin has an active development community and a rich API which allows its functionality to be used in novel ways.

SpamBayes (SpamBayes n.d.) is an open source Bayesian spam filter which came out of Paul Graham's "A Plan for Spam" (Graham 2004) and Gary Robinson's subsequent suggestions to improve Graham's original approach. It was introduced in the 2004 Conference on Email and Spam (Meyer 2004). Its source code is available on SourceForge.

Bogofilter (Raymond n.d.) is another open source Bayesian spam filter, based, in part, on the same research that spawned SpamBayes. Started by Eric Raymond in 2002, this project adds further statistical tools to the Bayesian classifier, attempting to make the filter more effective for spam filtering.

1.3 **Research Focus and Limitations**

This research will be limited to the use of these three filtering tools and Linux syslog entries. These syslog entries will be collected from the computer systems of a large university

and a mid-sized non-profit entity, so no inferences can be drawn beyond the original systems, though the results of the analysis are nonetheless useful. Further, the outage data from the non-profit's monitoring systems will be used for comparison. Since monitoring can be rather uneven, with some systems heavily monitored and others only lightly monitored, these data will be simplified to a problem/no problem form and a time frame.

Again, though the statistical inferences are limited, the analysis is still useful. It brings together a need common to most organizations' IT departments with a technology which is well-known in the industry. The integration work between the need and the solution is fairly simple, requiring only a small amount of glue coding to be done. The post-run analysis code framework is likely to be several times the length of the actual glue code.

2 LITERATURE REVIEW

2.1 Logging

2.1.1 Log Files

The computer is opaque to its users, including the very users who give the computer its marching orders. After all, the inner workings of the computer are encoded as electrical pulses in a myriad of circuits. Because humans cannot perceive the inner workings of computers, even though those workings are the creation of humans, programmers have worked hard to make those inner workings available to system users. The end-user of a computer system will see the results of the inner workings of the system through the computer's user interface, most commonly in the form of a graphical user interface (GUI), but also commonly manifested as a shell prompt or even as lights or ink on some sort of output device.

Often, however, the programmer or system administrator will want to understand what is going on deeper inside the system, to understand the state of a program or set of programs which are running in some way outside of the normal user interface to the system. One common way of giving insight into the internals of a system is to print messages to an output device, commonly the screen or to a file on a file system. Because displays are already in use with output for an end-user, the file is usually the window of choice when a programmer wants to see into a program. Writing a log of events to a file can provide extremely valuable insights into

system operation, not only because of the insights into the variables and actions involved, but also because of the timing of those variables and actions in relation one to another. Programmers can use these log entries to gain insight into potential problems with the code; system administrators can use the log entries to guide efforts of system management and troubleshooting.

From the early days of computing, each programmer would write log entries out in his or her own ways, and there was no real standard for log entries beyond writing them to a file. After all, a word processor, a network driver, an operating system and a web server will have very different logging needs. Generally, each programming project would standardize on a logging framework and a set of conventions for what and how to send to log files. This standardization makes log parsing much easier for a given system, and once programmers or administrators of a given system become familiar with that system's logging method and style, they can skim through logs and find issues or areas of concern fairly quickly and easily.

2.1.2 Syslog

In the 1980's, Eric Allman, creator of Sendmail, developed a logging standard called syslog, also known as BSD Syslog because of its original ties to the BSD Unix distribution (Lonvick, RFC 3164 n.d.). This standard includes not only a basic layout for a log entry, but a remote protocol allowing syslog entries to be transmitted to logging servers and collected for many devices. Syslog became the de facto standard for Unix and Linux systems and was codified in the IETF's RFC 3164 in 2001. In 2009, the IETF expanded and more clearly defined the syslog protocol, moving from a protocol description document, RFC 3164, to a protocol

definition document, RFC 5424 (Gerhards n.d.). Additional transport and security RFC documents have also been produced since RFC 3164 (Lonvick, RFC 3164 n.d.).

The newest standard has yet to be widely adopted by the Internet community as of late 2010. The de facto standard, as documented in RFC 3614, provides a simple standard message format, consisting of the following (encoded as 7-bit ASCII in an 8-bit encoding unless otherwise specified).

2.1.2.1 **PRI**

The first section of the syslog payload is called the PRI. The PRI has 3 to 5 characters which indicate the priority and facility of the log message. The first character is the left angle bracket or less-than character. The next one to three decimal digits, collectively called Priority value, comprise two fields, called Facility and Severity. The final character is the right angle bracket or greater-than character.

The Facility field is a numerical code intended to represent the source of the message, as shown in Table I.

Table I - Facility Field Numeric Codes (RFC 3164)

Numerical Code	Facility
0	user-level messages
1	kernel messages
2	mail system
3	system daemons
4	security/authorization messages
5	internal syslogd messages
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon
10	Security/authorization message
11	FTP daemon
12	NTP subsystem
13	log audit
14	log alert
15	clock daemon
16	local use 0 (local0)
17	local use 1 (local1)
18	local use 2 (local2)
19	local use 3 (local3)
20	local use 4 (local4)
21	local use 5 (local5)
22	local use 6 (local6)
23	local use 7 (local7)

The next digit gives the severity code as shown in Table II.

Table II - Severity Field Numeric Codes (RFC 3164)

Numerical Code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

The facilities codes are starting to show their age, as some of the facilities are no longer widely used, and there is a distinct lack of flexibility in the priority-specifying system (in fact, RFC3164 states that certain facilities are not ever used in practice). Still, the system is sufficiently flexible and expressive to have been adopted by the vast majority of Linux and Unix distributions.

2.1.2.2 **Header**

After the PRI is the HEADER. The HEADER contains a timestamp, in the "Mmm dd hh:mm:ss" format, where "Mmm" is a 3-character month name abbreviation, based on US English month names. After a single space separator, the header then contains the host name of the origination system. The host name field may contain the IP address of the host if the host has no host name.

2.1.2.3 **MSG**

The remainder of the syslog packet contains the TAG and CONTENT fields. The TAG field specifies the source program and is delimited, most commonly, by a colon (":"), left square bracket ("[") or space (" ") character. The CONTENT field is the log message, made up of ASCII characters. Most early implementations allowed up to 1,024 characters in the CONTENT field. The MSG, and particularly the CONTENT portion, is notoriously free-formed. It is not uncommon for applications to omit the TAG. The CONTENT field is not structured beyond the use of ASCII and the common length limitation, though individual projects and programs tend to use common conventions within that specific project.

2.1.2.4 **Network Transport**

For network transport, syslog messages are sent over UDP/IP, using UDP port 514. Since these log entries are system messages, speed and simplicity was considered more important than the delivery guarantee that TCP/IP promises. Some loss was considered acceptable as trade-off for simplicity and speed. Using UDP also allows packets to be sent to target hosts regardless of whether that host is up or down – separating the status of the sender from the status of the receiver (otherwise, the sender would receive TCP timeout errors when the target is down and would have to implement more complicated message handling). With the much greater speed and reliability of modern networks, many administrators prefer to transport these packets over TCP/IP; IETF RFC 3195 addresses this transport usage (IETF n.d.).

The syslog protocol and its protocol handlers also include the ability to relay messages. This allows arbitrarily complicated hierarchies of loggers to be assembled, allowing even very large, very diverse and very distributed computer systems to utilize the protocol for logging.

2.2 **Syslog Analysis**

Because of its flexibility, many applications, operating systems and devices use the syslog protocol. These log entries can then be easily consolidated to various log servers in an organization. This popularity and ease of aggregation has given rise to syslog as a de facto standard for enterprise logging, in spite of the notoriously loose structure of the CONTENT portion of syslog messages. This has also given rise to many frustrations for system administrators who would like to get more out of their syslog systems but are hindered by the sheer volume of loosely structured records.

Each field in the syslog packet provides useful information to a log analyst. This research will focus largely on the application-specific CONTENT field, using the other syslog fields in various support roles. The CONTENT field contains specific status information concerning a given application or service on a host and, because of its infamously free-form nature, requires more effort to analyze than other fields.

Log files can be a tremendous resource for programmers, system administrators and system analysts. Because of the ease of integration and great power to aggregate logs, syslog is particularly useful for systems integrators and administrators. Because so many applications, operating systems and devices utilize syslog, a great deal of data can be easily collected for later use. This is a very good thing for systems analysts, in the sense that system data can be collected in a single place. However, the down side of this is that even small to moderate data centers can generate millions of log entries per day. The sheer amount of data can overwhelm even the best-intentioned administrators.

Because of the quantity of log data available, and the value of the information contained in those logs, log analysis has been an area of ongoing study. Many researchers have applied various statistical and visual data mining techniques to logs of various types with varying levels of success.

2.2.1 Background Research

In 1996, Doug Hughes (Hughs 1996), then of Auburn University, described how they were using various visualization tools for managing their systems and networks. One of the tools discussed, tklogger, was used to decrease the noise in log files by visually dividing out the high and low priority messages. Its grouping/clustering mechanism is not discussed. However, even

this simple tool had proven itself useful to administrators, helping them to filter out less important records.

In 2008, Wei Xu, et al, (Xu 2008) proposed analyzing source code for all possible log output lines, then simplifying down that data set using Principle Components Analysis (PCA), and finally mining console logs for these errors. Their sample system, a Hadoop file system, lent itself to this sort of analysis, because it is open source and produces millions of lines of logs. This is clearly a novel technique to determining which log entries are important and which are not. Unfortunately, it is probably too limiting (due to required source code access) and too knowledge-heavy (due to the need to extract every meaningful log statement from the source and then apply a PCA transform to the data set) to be widely used without a large investment. Companies selling large commercial log analysis tools (which have the most resources for such projects) are unlikely to invest the required effort for a tool that cannot even be used on their own proprietary code without revealing its internals, or which is tied to specific versions of software.

As the value of logs has become more visible, commercial and free systems have been developed to generate notifications or to take automatic action based on log entries. There are many tools with these sorts of capabilities, including Swatch (Swatch n.d.), Splunk (Splunk Inc. n.d.), Quest Big Brother (Quest Software, LLC n.d.), Zenoss (Zenoss Inc. n.d.), Tivoli TEC (IBM n.d.), Nagios (Nagios Enterprises, LLC n.d.), LogSurfer (Thompson n.d.) and the like. These tools can be quite useful, but their automated analyses are generally simplistic at best, usually limited to pattern matching. In order for a rule to be created, a domain expert must find the pattern in the logs and create a matching rule. This makes initial setup costly and time-consuming. However, once set up, these rule sets can be augmented so that their value increases over time. Joseph Hellerstein, et al, (Hellerstein 2002) proposed using data mining techniques to

find event bursts, periodic patterns and mutually dependent patterns in historical logs for the purpose of generating correlation rules for automated real-time systems management and intruder detection tools. Their techniques analyze historical data to automatically select patterns, assisting an analyst in the creation of appropriate rules.

Risto Vaarandi (Vaarandi, Sec - A Lightweight Event Correlation Tool 2002) proposed a lightweight event correlation engine, called SEC, which uses pattern matching rules. Vaarandi's 2003 paper, "A Clustering Algorithm for Mining Patterns from Event Logs" (Vaarandi, A Data Clustering Algorithm for Mining Patterns from Event Logs 2003) introduced an enhanced log correlation engine, called SLCT.

In 2004, John Stearly (Stearly 2004), of Sandia National Labs, described a system to analyze syslogs using a bioinformatics-inspired algorithm for detecting anomalies in large volumes of log data. The Sisyphus toolkit which he developed utilized an IBM-developed pattern discovery algorithm, called Teiresias (Rigoutsos 1998), which was originally used for analyzing biological sequences. The technique compares well with previous techniques, particularly the a priori-based techniques in SLCT and Loghound, though it is still only an academic research project.

Another method of clustering event logs was used by Makanju, et al, (Makanju 2009) and presented in 2009. Their technique, called IPLoM, or Iterative Partitioning Log Mining, divides the set of log entries through a two or three stage iterative partitioning technique. This technique slightly outperforms the SLCT, Loghound and Teiresius algorithms to which it was compared, but, again, it is as yet an academic exercise and not widely available.

Gunter, et al, (Gunter 2007) analyzed syslog entries to manage complex middleware running on a large Grid computing system. Their research concluded that no single type of

statistical analysis they tried was optimal for all situations and that multiple analysis techniques should be used.

In the introduction to his 2002 paper, Vaarandi points out that most tools for this sort of correlation are large, complex, platform-dependent and expensive, and that these were motivation for him to develop SEC. While there are more cross-platform, open-source and free alternatives today, the playing field is quite full of complex, expensive commercial tools (some of which were previously mentioned). This brings up an important point: if an analytical tool is too expensive or complex, its value will be severely limited in the world of real use. Administrators already have much to do and many budgetary constraints. They need financially and intellectually approachable options.

Numerous log visualization techniques have been used as well (Aharon 2009) (Takada 2002) (Hochheiser 2001). This research will focus on simple, commonly available filtering tools, for which filter scores and simple score vs. time scatterplots are quite effective visualizations, and leave these more advanced visualizations for later research.

2.2.2 Syslog Analysis Tools and Products

As mentioned earlier, because syslogs are potentially valuable resources for many aspects of system management, many tools have sprung up over the years for analyzing them. Many tools exist for viewing, searching, filtering, parsing or managing logs. Some, like Microsoft's Log Parser (Microsoft Corp. n.d.) are simple log parsers and formatters. Others, like Apache ChainSaw (Apache Foundation n.d.) and Octopussy (Thebert n.d.), provide viewers reporting and alerting as well. Still others, Splunk (Splunk Inc. n.d.), XpoLog (XpoLog Ltd. n.d.), Novell

Sentinel Log Manager (Novell, Inc. n.d.) and LiquidLabs LogScape (Liquidlabs n.d.), provide log management, forwarding, viewing, searching, alerting and many other features.

In all cases, these tools provide, at most, simple visualization and query tools for filtering through logs. Of the most popular tools on the market, only LogScape and Splunk provide significant statistics for log entries, and those are limited to frequencies of matched entries and other simple metrics. Statistical filtering of log entries is still a nascent area of concern in the log management, and worthy of further investigation.

2.3 **Spam Control**

E-mail spam, or unsolicited, unwanted e-mail messages, is often considered the great plague of the today's information society. The International Telecommunications Union's 2005 legal analysis of spam law put spam as 88% of e-mail that traverses the Internet (Bambauer 2005). Symantec, a major provider of anti-spam software, in May of 2009, put the figure at over 90% of all e-mails being spam (Ragan n.d.).

Because of the very high levels of activity in the spam world, a great deal of effort has gone into battling the spam problem. Because spammers are actively trying to get through spam filters, this has created a sort of arms race, with the see-saw tipping towards the spammers, then the blockers, then the spammers, back and forth. One of the tools proven to be most useful for the anti-spam forces has been the Bayesian spam filter, which is of particular importance to this research. These filters are trained, and therefore can be quite resilient to variations in text as presented by spammers. This resiliency should also make spam filters useful for filtering log entries, which are generally similar, but rarely exactly the same across machines and over time and software upgrades.

2.3.1 Bayesian Spam Filtering

In 1998, Mehran Sahami, et al, (Sahami 1998) wrote a seminal article describing how spam could be filtered using a Bayesian probabilistic classifier. This article proposed the use of Bayesian probabilistic machine learning techniques on the then-new spam classification problem. The article specifically suggested the use of the naive Bayesian classifier.

In 2002, Paul Graham's (Graham 2004) similarly influential article entitled "A Plan for Spam" argued that it was possible to stop spam precisely because spam must convey a message, and that a naïve Bayesian classifier, used in other areas of the field of text classification, could be used to analyze that message. He posited that an effective filter could be created with a simple algorithm just taking the probabilities of certain words and combining them with a simple Bayesian calculation.

Since the actual text of a spam message must be of a certain type in order to convey its message to the reader, text classification tools can be brought to bear on the message itself, which must be there and must be plain enough to communicate its message to the recipient. Text classification has many aspects, but the particular aspect on which this paper focuses is the aspect of Bayesian classification, from the world of Bayesian statistics.

Although this is somewhat oversimplified, the statistics world is largely divided into two major domains: Frequentist statistics and Bayesian statistics. The statistics most used today is of the frequentist domain, which relies solely on the attributes of the data set to tease out patterns in the data, requiring no previous knowledge of the data. Bayesian statistics, named after the 17th century mathematician and minister Thomas Bayes, on the other hand, takes into account previous experience and combines that knowledge with statistics from the data to make further inferences.

A Bayesian spam filter is a spam-oriented Bayesian content filter. It uses Bayesian statistical theories to combine knowledge of previously categorized messages with analysis of incoming messages, categorizing e-mails as spam or non-spam (Zdziarski 2005). (Non-spam messages are also commoly known as "ham.")

In essence, an administrator trains the filter by giving it a certain number of spam messages, telling the filter that these are spam, and a similar number of ham messages, telling the filter that these are ham. Once the training is completed, when the filter receives an e-mail, it compares the words found in the mail message to the words in its two categories; it then combines the probabilities of the new messages words with the probabilities of similar words in the two categories and determines how likely it is that this message belongs in one category or the other.

There are a number of email filtering products which implement various Bayesian algorithms; some of these algorithms are not, technically, using Bayes' theory, but all of them are lumped together as Bayesian because of their similar properties. The best known of the open source products is SpamAssassin (Apache Foundation n.d.). Two other well-known filters are SpamBayes (SpamBayes n.d.) and Bogofilter (Raymond n.d.).

2.3.2 SpamAssassin

SpamAssassin is actually a very rich spam filtering tool which uses multiple techniques to recognize and filter spam from a mail stream. One of its filters is the Bayesian filter.

SpamAssassin is very widely integrated into mail systems because of its rich API set and open licensing. As part of the Apache umbrella, it is released under the very liberal Apache 2.0 license, which allows derivative works to be made and sold of it. It also comes with a rich Perl

Mail:SpamAssassin:Conf API library, and thorough documentation on the web and in numerous books.

2.3.3 SpamBayes

Tony Meyer and Brendon Whateley (Meyer 2004) took Paul Graham's "A Plan for Spam" ideas and presented SpamBayes, written largely by Python's Tim Peters, at the 2004 Conference on Email and Spam. They took the basic two-bin classification concept and stretched it to a three-bin system, with an "unsure" range. The SpamBayes mail classifier was presented not as a solution unto itself, but as a test harness for ideas to show techniques which might be useful for other engines. The use of Python as an implementation language, with its emphasis on code readability, has allowed this tool to be accessible to other projects.

2.3.4 Bogofilter

The open source filter, Bogofilter, is based, in part, on the same research on which SpamBayes was based. Started by Eric Raymond in 2002, this project also utilizes a geometric mean algorithm with Fisher's method modification from Gary Robinson (Raymond n.d.), attempting to make the filter more effective. It is more similar to SpamAssassin than SpamBayes in how it is trained and run.

2.4 Summary

In summary, attempts to monitor computer system activity for management and troubleshooting have led to various analysis techniques. These analytical procedures require an understanding of many topics, including logging, syslog, text classification and Bayesian content

filtering. This background provides the basis for the methodology used in this current research project, which entails using a Bayesian spam filter as a clustering tool for filtering problem-related log entries from non-problem-related log entries. In particular, SpamAssassin, SpamBayes and Bogofilter will be used, as the first is the most commonly used open source spam filter, while the other two have been thought-leaders in the push to use Bayesian content filters for attacking the spam problem.

3 METHODOLOGY

This research focuses on the novel application of well-known spam filtering tools to the problem of filtering Linux syslog files. Syslog files, and application logs based on syslog concepts, are notoriously loosely formed, which makes them difficult to classify: there is little structure imposed upon them and even the structure which is imposed is loosely interpreted. A message of Error level severity for one application might mean that there is an outage, while a message with the same severity from another application may have no relation to an actual service outage. Syslog is also known for producing copious output, with even mid-sized server farms producing millions of lines of logging per day. This latter attribute may, oddly, actually assist in the analysis of the data, since truly meaningful, outage-related data is sufficiently rare that one needs a great deal of data "ore" in order to "mine" out the information "gold" found in these files.

3.1 Spam and Log Data Sources

For this research, both spam and log data (syslog and application log) will need to be utilized.

SpamAssassin, SpamBayes and Bogofilter all provide Bayesian content filters, which will be used to separate problem-related entries from non-problem-related entries, in effect filtering out some of the noise of uninteresting entries. As part of its offering, SpamAssassin

has a well-tested, well-understood corpus of testing spam messages which can be used for testing spam filters. This corpus will be utilized for the early stages of testing, using the messages, initially, as-is for validating that the tools work as expected, then modifying the spam messages to make them more similar to log entries, which tend to be much shorter.

Once the preliminary spam entry testing is completed, then a set of syslog entries with at least some known failures will be utilized. Some arbitrary records will be used to build a simple test bed of contrived data, allowing for some testing of log entries in a controlled manner. Then, finally, actual entries will be used.

BYU's School of Technology provided 1 month of syslog entries, for use in the contrived log entry differentiation testing. The syslog entries were filtered through the Bayesian content filtering provided by SpamAssassin, SpamBayes and Bogofilter, differentiating the one application's entries from another application's entries.

Actual application log entries from the non-profit FamilySearch.org web site were used for the final log testing. These entries, while not syslog entries, are structured very similarly to syslog. Two outage timeframes were be addressed: four actual outages from the Spring, which have been determined to be similar by the site's administrators, were tested for similarity, randomly selecting one as the training outage. Then, four actual outages from the previous Fall have been found by that site's administrators, and that month's copious log entries were thought to be correlated with those outages. For this second set, the testing trained the filters with log entries from the first outage and attempted to correlate those entries with the subsequent outages.

3.2 Textual Analysis Methods and Clustering of Textual Data

As was discussed in the previous chapter, there are many techniques for analyzing text-based data. The most common of these are the various methods of data clustering, in which some sort of statistical distance metric (such as a matrix of word entries per row) is created for each entry; then this metric is used to cluster or group similar text entries together. Clustering is particularly useful for log analysis, as it mirrors what an administrator would do by grouping related entries together, allowing for deeper analysis on the more interesting or relevant entries, while the uninteresting or unrelated entries can be ignored. Reducing the "log entry noise" is crucial for dealing with the vast amount of data generated by logging systems, and clustering algorithms are ideal for this application.

3.2.1 Bayesian Filtering for Clustering

One particularly useful clustering technique is Bayesian content filtering (often called BCF). It allows for the utilization of foreknown information, about the nature of entries, in the analysis of successive entries. For this to work, the Bayesian filter must be "trained," or given this foreknown information. The filter then uses what it learns from the training data and combines it with data it discovers in subsequent entries, allowing it to differentiate those subsequent entries.

3.2.1.1 Spam Abatement Tools Using Bayesian Filters

In the case of spam filtering, one trains the filter with known spam and known ham (or non-spam messages). The filter calculates the rate of occurrence for various words found in these messages in the two categories. Words like "mortgage" or "Viagra" or phrases like "call

now" are likely to be found more commonly in spam than ham, whereas words and phrases like "Mom" and "going home" are more likely to be found in ham. Words from subsequent messages can then be compared to words in these categories, then their likelihoods of appearing in either category can be combined using Bayesian methods, and the message can finally be categorized as spam or ham. These kinds of filters, when trained properly, can be very effective.

Bayesian filters have their weaknesses, however. Such filters do not take into account word order or other features which may be useful in recognizing patterns useful for categorizing text. Additionally, as with many statistical tests, the more data that can be used for the training, the more effective a filter can be made. This is an ongoing challenge, as interesting log events can be quite rare, especially in small environments.

One statistical technique that has proven more effective has been the use of Hidden Markov Models to capture some of the structure of a given entry. Taking a Spam-related example, we can see that there is a difference between “I refinanced my house today to save some money” and “Refinance today and save big money”. A Bayesian filter would find difficulty in differentiating these short sentences, but a Hidden Markov Model-based filter would utilize word orders to differentiate them -- much as a human reader would.

Hidden Markov Model-based categorizers also have their own weaknesses, the two largest being that they are complex to implement and, more importantly, they are computationally expensive to utilize. One other weakness, and one important to an IT administrator, is that they are not nearly as widely available nor as easily utilized as Bayesian filters.

Log entries were modified and tested to simulate some features of Hidden Markov Models with the Bayesian tools which were used for these tests. It was hoped that these

modifications would make for better filter effectiveness while maintaining the high throughput performance of the Bayesian filtering tools.

3.2.1.1.1 SpamAssassin

SpamAssassin has hundreds of tests for categorizing e-mail as spam or ham. An effective Bayesian filter is included in this set of tests. SpamAssassin can be trained and utilized via simple command-line tools, as long as the messages are provided in an e-mail format such as Unix maibox or mbox. The intention in the log testing portions of this research is to wrap all log entries in a common generic e-mail header and utilize the standard e-mail-based tools for training and analyzing log entries. Because SpamAssassin runs many tests by default, and those tests are unrelated to this research, only the Bayesian filter will be utilized for this set of tests.

3.2.1.1.2 SpamBayes

SpamBayes is a specialized tool providing only an effective Bayesian filter for spam filtering. It is less well supported than SpamAssassin, but its Bayesian filter is better documented and written in the very accessible Python language. Similar to the SpamAssassin tests, SpamBayes' e-mail-based tools will be utilized for training and analyzing log messages.

3.2.1.1.3 Bogofilter

Bogofilter is another commonly used Bayesian Spam filter. It uses similar training and analyzing techniques to SpamAssassin and SpamBayes and will be utilized similarly.

3.3 Bayesian Filter Effectiveness Testing

Each of these three filtering tools will be trained in various ways to understand their effectiveness. Spam messages are considerably longer than log entries, which will likely impact the effectiveness of Bayesian filters. More than this, problem-indicating log entries are quite rare compared to the vast numbers of other log messages. Because of these two factors, the move from spam to log entry tests will be made in steps.

3.3.1 SpamAssassin Corpus Testing

Initially, the SpamAssassin spam corpus will be used to test each tool. The effectiveness of each tool will be noted. This should be straight-forward, as these tools were designed for the purpose of differentiating these sorts of messages, and the messages will be in the correct format for analysis.

3.3.2 Contrived Short Entry Testing

After the direct usage of the spam corpus, the messages of the SpamAssassin corpus will be broken down into shorter entries, by sentence or line. Then the filters will be trained and used again. All the same training set lines will be used, but in shorter form (i.e. there will be many more short messages than before). This will measure the effectiveness of the filters as the length of each message goes down. These data will provide insights into how the shorter lines of log entries might affect the accuracy of the filter.

3.3.3 Controlled Log Entry Testing

A small program will take each entry in a log file and create an mbox mailbox for training the filters. Another small program will be used to then feed one entry at a time into the filter to test whether it matches the problem-related set or the normal set.

For the first run, a known set of log entries will be chosen, such as all the possible entry types for a subset of the data from a given pair of applications (e.g. named or sshd or dhcpd). The rest of the data will then be categorized to determine if the filter can differentiate subsequent entries from these two applications.

Many log entries include IP addresses, MAC addresses and other numerical values which are specific to only a given message rather than to a class of messages. To determine if these values were helpful or hurtful, these values were normalized to zeros for one set of tests, and left as they were for another set of tests.

Because of the brevity of the log entries to be tested, a way to increase the data available from a given message needed to be found. Additionally, it was also desirable to represent the message structure somehow to the filters. In doing so, it was hoped that to the tests would gain some of the benefit of a Hidden Markov Model filter, but with much lower computation cost (Zdziarski 2005). This can be accomplished by "chaining" the words in a message, which is also called word-level n-gram creation in some research (Cavnar 1994). Adjacent words in a given message were initially chained together with underscore characters. These chains were 1 (no chaining), 2, 3, 4, 5, 7 and 9 word chains. For example, a line such as "Now is the time for all" would give these tokens to the filter: T_1:{Now, is, the, time, for} at a chain length of 1; T_2:{Now_is, is_the, the_time, time_for} at a chain length of 2; and T_3:{Now_is_the, is_the_time, the_time_for} at a chain length of 3, where T_C is the set of tokens generated with chain length of

C. For a message with L number of words, the number of tokens (K_c) generated at the level of chain length C is:

$$K_C = L - (C - 1) \quad (1)$$

For a given chain length, two tests were then run. One run was tested with tokens TC only from the highest level chain length C being trained and tested upon. The other test, called "stacked-chains" in this paper, was run with all the tokens $\{T_1, T_2, \ldots T_C\}$ from all chains up to the length C being concatenated together and used for training and testing with the total number of tokens:

$$K = \sum_{i=1}^{C} K_i \quad (2)$$

This process adds three variables: normalize-numbers at two levels (true and false), chain-length at 7 levels (i.e. the numbers of words chained together) and stack-chains at two levels (true with all chained word sets from 1 up to the specified chain length, and false with just the highest order chained word set).

Adding these variables complicates the process, since logs must be pre-processed by some tool, but overall processing time is not greatly impacted by number normalization or word chaining. For 66 training records and 10,000 test records, impact was from 1.1% to 6.6%, the latter being just 30ms for those 10,000 test records.

3.3.4 Full Log Entry Testing

Once the filters could differentiate the controlled log entry set effectively, then the filter was retrained to look for specific log entries relating to system problems, and two sets of live production logs were tested. For each set, one "spam" set of log entries was selected from the few minutes prior to, or immediately following, the initial outage. The matching "ham" set of

log entries were, in one case, randomly selected from the the rest of the lines in the log file, and in the other, entries were randomly selected from the hour's prior entries (keeping the ham/spam entry sets approximately the same size), as the logs had been trimmed to just the bracket due to their large number. The filter performance for each of these sets, and each of these tools, was compared.

3.4 Analysis of Comparisons and Correlation of Full Log Entry Tests with Monitoring Outage Data

The various tools, training techniques and data manipulations were compared by graphing the scores of entries from each time period and comparing those graphs to the actual time periods of each outage. These graphs were generated with R, a widely-used open-source statistical analysis package, which makes such analyses quite straight-forward. The efficacy of the technique was tested using one set of data, while the utility of the technique as a filter was tested with the second set of data.

4 RESEARCH RESULTS

4.1 SpamAssassin Corpus Testing

Installing SpamAssassin, SpamBayes and Bogofilter was fairly straightforward on the OpenSUSE 11.1 testing platform. Bogofilter, in particular, was already installed. SpamAssassin and SpamBayes were also straightforward, the former being installable with the platform's YaST manager and the latter installable as a Python distutils package.

The SpamAssassin public corpus was downloadable from the SpamAssassin corpus repository site at Apache.org (Apache Foundation n.d.).

Like many publicly available packages, this public corpus of emails includes a readme file which describes its package contents like this:

OK, now onto the corpus description. It's split into three parts, as follows:

- *spam: 500 spam messages, all received from non-spam-trap sources.*
- *easy_ham: 2500 non-spam messages. These are typically quite easy to differentiate from spam, since they frequently do not contain any spammish signatures (like HTML etc).*
- *hard_ham: 250 non-spam messages which are closer in many respects to typical spam: use of HTML, unusual HTML markup, coloured text, "spammish-sounding" phrases etc.*
- *easy_ham_2: 1400 non-spam messages. A more recent addition to the set.*
- *spam_2: 1397 spam messages. Again, more recent.*

Total count: 6047 messages, with about a 31% spam ratio.

The corpora are prefixed with the date they were assembled. They are compressed using "bzip2". The messages are named by a message number and their MD5 checksum.

(Apache Foundation n.d.)

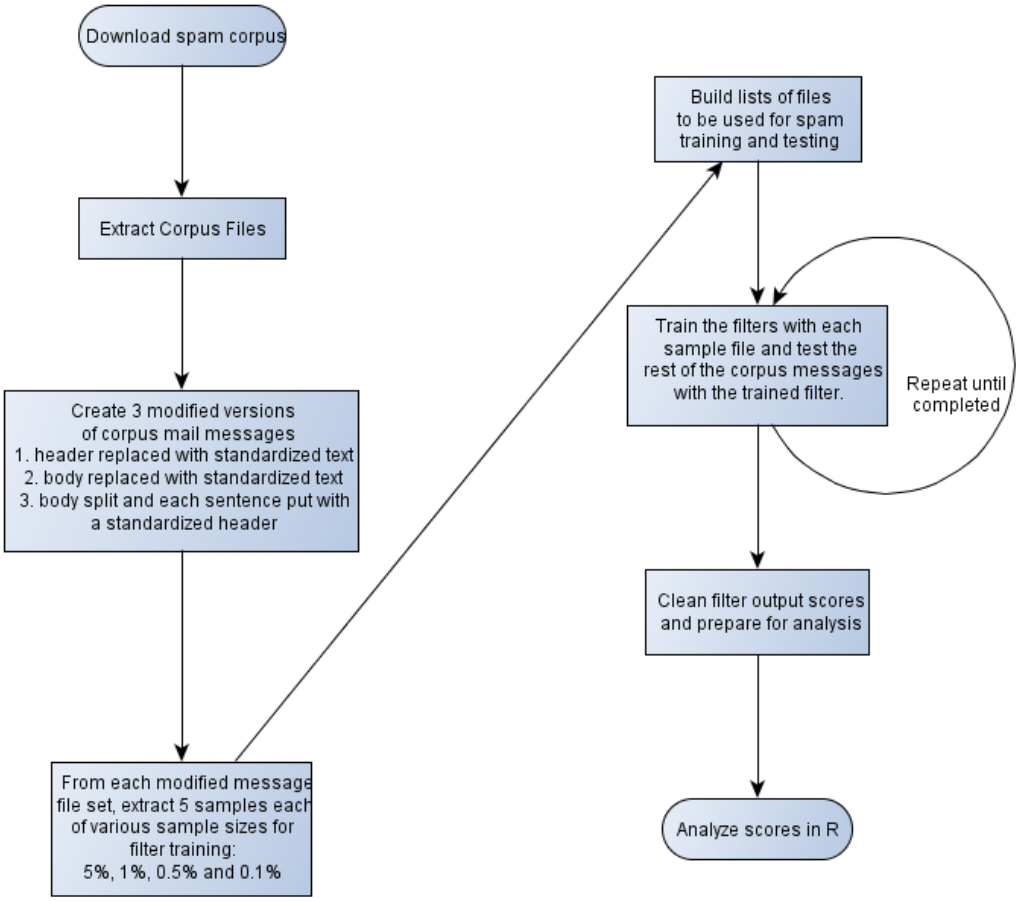

Figure 1 - Spam Testing Flowchart

A simple flowchart of the work done is shown in Figure 1. The actual work went as follows:

1. Extracted SpamAssassin Corpus files: The distribution file set was unpacked. This was run once for the full suite of tests.

2. Three modified versions of each spam e-mail were created: one with the headers removed, another with body removed and a final one with headers removed and the body split into multiple messages by sentence (using a period as a splitting delimiter). These modified files were used for each type of training to determine how much of the spam-vs. ham differentiable text is in the headers vs the body, and if breaking the messages into small (log-like) pieces would make a difference in the training. The unmodified messages were used for the testing of the filter. The tools written for this were shtrim.py, shtrim.body.template.txt, shtrim.header.template.txt, found in Appendix A (as is the case with all scripts and programs mentioned here). This set of programs was run once per set for all tests.
3. Created XX% sample files that list names of message files. This created files with a naming convention of train_[h|sp]am_filelist_(xx%descriptor) and test_[sp|h]am_filelist_(descriptor_100-xx%) for each sample-sized file. There was a concern that the sample sizes might not sufficiently represent the variation in the original data, so the stestgen.py and stestgen.properties files were run 5 times for each sampling size in order to get 5 random samples at each sample size.
4. Built out file lists appropriate for each type of manipulation (e.g. train_ham_b_filelistX or train_spam_bsplit_filelistX). The script gen-altered-list.py was used to build out appropriate files once per sample size set.
5. Those file lists and actual messages were used to train and test with each tool. For this purpose, the scripts train_sabsplit1.sh, train_sah1.sh, train_sab1.sh, train_safull1.sh and test_sa1.sh, with appropriate command-line parameters and

automated with runset.sh, were created and run. This work was done once per sample size set.

6. Those output files were manipulated into tab-separated files suitable for R analysis using data_normalizer.py, then analyzed the data in R using fourthrun-analysis.R.

This work turned out to be far more labor-intensive than originally thought, partially due to the number of variables independently controlled for:

1. Three different spam filtering tools
    a. SpamAssassin
    b. SpamBayes
    c. Bogofilter
2. Three different manipulations of the messages during training
    a. Retain header data only (standardized body)
    b. Retain body data only (standardized header)
    c. Split body data into separate messages (each with standardized headers)
3. Five sample sizes
    a. 10%
    b. 5%
    c. 1%
    d. 0.5%
    e. 0.1%
4. Five separate random samplings and associated runs
5. Spam-only runs vs Ham-only runs (allowing a determination of accuracy)

Summarizing just the accuracy rates of all these runs concisely produced the data in Table III. Since the actual message types are known, accuracy rates could be calculated. The 5 sampled runs for each combination were averaged to reduce and normalize the analyzed data.

Table III - Spam Corpus Testing Results

Tool	Manipulation	Sample_size	Percent_correct
spamassassin	no_manipulation	5.00%	94.86
spamassassin	header_removed	5.00%	94.38
spamassassin	body_split	5.00%	94.38
spamassassin	body_removed	5.00%	94.38
spamassassin	header_removed	1.00%	92.54
spamassassin	no_manipulation	1.00%	92.48
spamassassin	body_split	1.00%	92.48
spamassassin	body_removed	1.00%	92.48
spamassassin	header_removed	0.50%	91.3
spamassassin	body_removed	0.50%	91.28
spamassassin	body_split	0.50%	91.26
spamassassin	no_manipulation	0.50%	91.24
spamassassin	no_manipulation	0.10%	90.82
spamassassin	header_removed	0.10%	90.8
spamassassin	body_removed	0.10%	90.8
Spamassassin	body_split	0.10%	90.78
spambayes	no_manipulation	5.00%	89.74
spambayes	header_removed	5.00%	84.7
spambayes	body_removed	5.00%	84.7
spambayes	no_manipulation	1.00%	73.1
bogofilter	no_manipulation	5.00%	69.6
spambayes	header_removed	1.00%	68.64
spambayes	body_removed	1.00%	68.64
spambayes	no_manipulation	0.50%	63.64
bogofilter	header_removed	5.00%	63.6
bogofilter	body_split	5.00%	63.6
bogofilter	body_removed	5.00%	63.6
spambayes	header_removed	0.50%	58.84
spambayes	body_removed	0.50%	58.84
bogofilter	no_manipulation	1.00%	54.42
bogofilter	header_removed	1.00%	51.22
bogofilter	body_split	1.00%	51.22
bogofilter	body_removed	1.00%	51.22
spambayes	body_split	5.00%	50.0
spambayes	body_split	1.00%	50.0
spambayes	body_split	0.50%	50.0
spambayes	body_split	0.10%	49.96
spambayes	no_manipulation	0.10%	49.46
bogofilter	no_manipulation	0.50%	49.18
spambayes	header_removed	0.10%	49.06
spambayes	body_removed	0.10%	49.06
bogofilter	no_manipulation	0.10%	48.18
bogofilter	header_removed	0.10%	47.0
bogofilter	body_split	0.10%	47.0
bogofilter	body_removed	0.10%	47.0
bogofilter	header_removed	0.50%	45.12
bogofilter	body_split	0.50%	45.12
bogofilter	body_removed	0.50%	45.12

4.1.1 Contrived Log Entry Testing

For a simplified log entry test, a corpus of syslog data received from Brigham Young University School of Technology internal systems was used. This corpus contained 380,397 lines of log entries, most of which were dhcpd entries. Also in this corpus were 2356 lines from dhclient and 1918 lines from the kernel. To train the filter, 25 sample messages were randomly selected from the dhclient and kernel sets (using randlines.py), and these lines were used to train the various filters, setting the kernel messages as spam and the dhclient messages as ham. These lines were parsed so that only the body of each log line was used, discarding the date, server and application name portions of the messages.

```
Aug··9·04:59:06·srv1·ntpd[3501]:·kernel·time·sync·status·change·4001¶
Aug··9·11:49:04·srv5·kernel:·[0.000000]··BIOS-e820:·000000007fef0000·-·000000007feff000·(ACPI·
data)¶
Aug··9·11:49:04·srv5·kernel:·[0.684810]·vgaarb:·loaded¶
```

Figure 2 - Sample kernel/Spam Log Entries

Several sample kernel (spam) and dhclient (ham) lines are shown in Figure 2 and Figure 3, with (addresses and server names modified to protect the innocent).

```
Aug·10·08:23:06·srv9·dhclient:·bound·to·10.38.1.71·--·renewal·in·3009·seconds.¶
Aug·10·11:25:51·srv5·dhclient:·DHCPREQUEST·of·10.38.1.36·on·eth0·to·10.38.1.4·port·67¶
Aug·10·13:04:53·srv12·dhclient:·DHCPACK·from·10.38.1.4¶
```

Figure 3 - Sample dhclient/Ham Log Entries

A combined file with all the dhclient lines and all the kernel lines was then tested, line by line, against the trained filter.

The following variables were controlled for independently:

1. Three different spam filtering tools
    a. SpamAssassin
    b. SpamBayes
    c. Bogofilter
2. Numbers in log entries
    a. Left as-is
    b. Normalized to zeros
3. Words were chained together with underscores in order to retain some of the structure of each line. Chains are formed by putting together adjacent words so they form n-grams in the form of "superwords": e.g. creating three word chains from the phrase "Now is the time for all good men to" would give "Now_is_the is_the_time the_time_for time_for_all for_all_good all_good_men good_men_to". Initially, only odd numbers of words were used to reduce the number of test runs (1, 3, 5, 7, 9), but 2 and 4 word chains were also run as there appeared to be an inflection point in accuracy at the lower chain lengths:
    a. 1 – no words were chained; the unmodified line was passed into the filter
    b. 2 – two words chained (e.g. "Now_is")
    c. 3 – three word chains (e.g. "Now_is_the")
    d. 4 – four word chains (e.g. "Now_is_the_time")
    e. 5 – five word chains (e.g. "Now_is_the_time_for")
    f. 7 – seven word chains (e.g. "Now_is_the_time_for_all_good")
    g. 9 – nine word chains (e.g. "Now_is_the_time_for_all_good_men_to")

4. Chain stacking

    a. Chains were stacked, meaning that lower order chains were retained in the document used for training and testing. The same levels were used. (e.g. for "Now is the time for" at a 5 word chain, the output would include 4, 3, 2 and 1 word chains and be tested with the text "Now_is_the_time_for Now_is_the_time is_the_time_for Now_is_the is_the_time the_time_for Now_is is_the the_time time_for Now is the time for")

    b. Chains were not stacked, meaning that only the highest-order chain was used for training and testing.

The output from the filters was parsed to give the score (in the case of SpamAssassin) or the detected message type name and score (in the case of SpamBayes and Bogofilter).

Then the actual message types (kernel=spam, dhclient=ham) were prepended to each line.

SpamAssassin gives matches a numeric score. Because of the previous spam corpus experience, a score of 3.0 and above was scored as "spam" and below 3.0 as "ham." SpamBayes reports "Ham" and "Spam" in addition to a 0 to 1-scale score. Bogofilter reports "Ham," "Spam" and "Unknown" in addition to a 0 to 1-scale score. matchrate.py looks for these scores and names and determines if the given name or score matched the actual message type, giving counts for each file.

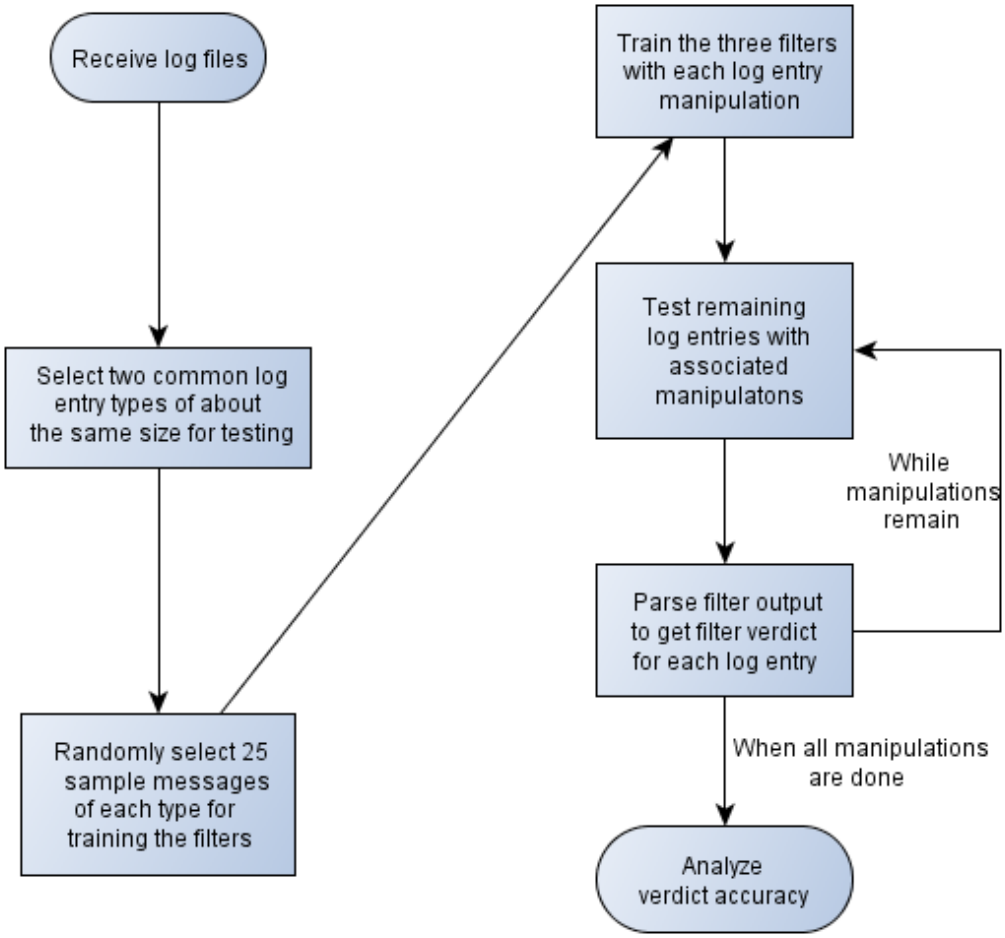

Figure 4 - Trivial Log File Testing Flow

The straight-forward general flow for this trivial log file testing and is shown in Figure 4.

The programs used for this work, randlines.py, l_train.py, l_test.py, l_salib.py, l_check_common.py, l_runtests.sh and matchrate.py are included in the appendix.

A histogram showing the SpamAssassin 3-chain score histogram is shown in Figure 5. It shows the distinct bimodal distribution of scores expected from the set of both spam- and ham-trained messages. The message recognition accuracy rates of the filters are also given below.

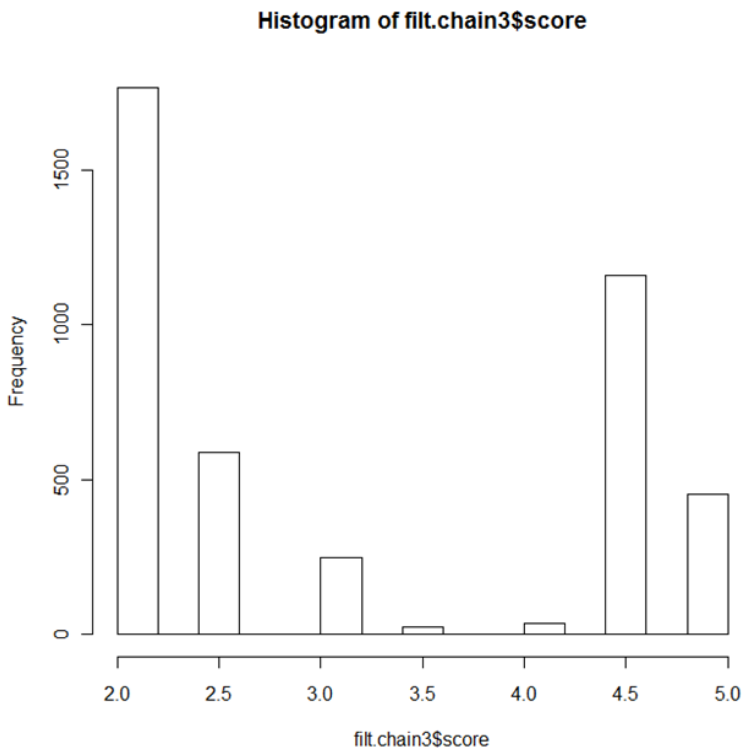

Figure 5 - SpamAssassin 3-Chain Score Histogram

Analysis and manipulation of data was accomplished with custom Python scripts. Statistical analysis was accomplished with R version 2.12.0 (2010-10-15) (R Development Core Team n.d.) running under 32-bit Windows 7.

The output of these runs produced 358680 rows of data: one output row representing one dhclient or kernel log entry with a unique set of each of the 4 variables.

The following tables will give an overview of the data. Table IV shows the actual ham/spam split (i.e. the split of dhclient and kernel messages, per the syslog application field) and Table V shows the detected ham/spam split (i.e. the split of dhclient and kernel messages, per the filters' detection). For reference, the basic statistics for the output scores from each filter are shown in Table VI; as previously mentioned, SpamAssassin uses a floating scale, where it was determined that a 3.0 or higher indicated spam, while SpamBayes and Bogofilter use scales

from 0 to 1, where SpamBayes scores records with scores above 0.5 as likely to be spam and Bogofilter records records with scores above 0.95 as likely to be spam. Note that although the range of the SpamAssassin scores is nearly 5 times that of SpamBayes and Bogofilter, its standard deviation is not quite 3 times as great; this could indicate that SpamAssassin tends to score slightly strongly towards the ham or spam ends of its scale. Also note that means for both SpamBayes and Bogofilter are slightly towards the ham end of the scores, while SpamAssassin is slightly towards the spam end of the scores. The differences are fairly subtle, so a bit more analysis would be required to definitively explain why this is the case.

Table IV - Actual Ham/Spam Split

Type	Record Count
Ham	197791
Spam	160889

Table V - Detected Ham/Spam Split

Detected Type	Record Count
Ham	139512
Spam	142796
Unsure	76399

Table VI - Filter Score Statistics by Filter

Statistic	*SpamAssassin*	*SpamBayes*	*Bogofilter*
Min	0.80	0.00	0.00
Mean	3.09	0.47	0.48
Max	5.500	1.00	1.00
Standard Deviation	1.18	0.42	0.40

In addition to the Table VI filter statistics, note the following basic facts to get some idea of the comparative sizes of data sets in the analysis.

119560 records from each of the three filter tools

179340 records each of numbers normalized (true vs. false)

51240 records of each chain length (lengths: 1, 2, 3, 4, 5, 7, 9)

179340 records each of stacked chains (all chains up to the tested chain length used for analysis vs. just the tested chain length)

A logical column was added to indicate whether the type and discovered type (the type as detected by the filter) were the same for a given record. Two analyses were done with those data: a simple table of successful match rates at given variable levels, and a logistic regression.

Table VII shows the table analysis. For this, the percentage of correctly identified records for each unique unique filter, normalization, chain length and stacked value was calculated. As can be seen in the table, SpamAssassin was the most accurate filter, especially when using chains of 2 or 3 words—99.906% vs. 91.639%. SpamBayes, which was the next most accurate filter, actually did slightly worse with chained words, 96.815% vs. 95.035%. The same was true for the less-accurate Bogofilter, 92.435% vs. 90.984%. Stacked vs non-stacked chains seemed to make almost no difference across the board. Normalized numbers did not affect SpamAssassin or Bogofilter, but seems to have helped SpamBayes slightly.

A logistic regression was performed for a more rigorous statistical analysis by fitting a model with the matched column (set to TRUE when type and discovered-type were the same) as the dependent variable and the 4 other variables as independent. The model tested was: *matched ~ filter_tool + normalized + chain_length + stacked_chains*.

Table VII - Test Results Sorted by Correctness

Filter	Normalized	Chain length	% correct (stacked)	% correct (non-stacked)
spamassassin	false	3	99.906	99.906
spamassassin	true	2	99.906	99.906
spamassassin	true	3	99.836	99.836
spamassassin	false	2	99.696	99.696
spambayes	true	1	96.815	96.815
spambayes	true	2	95.035	95.035
spambayes	true	2	94.801	94.801
bogofilter	false	1	92.436	92.436
bogofilter	false	1	92.436	92.436
spamassassin	true	1	91.639	91.616
bogofilter	true	3	90.984	90.984
bogofilter	false	3	90.984	90.984
bogofilter	false	2	90.703	90.703
bogofilter	true	2	90.703	90.703
spambayes	false	1	90.703	90.703
spamassassin	false	1	86.628	86.628
spamassassin	true	5	86.136	86.089
spamassassin	false	5	86.112	86.112
spamassassin	true	4	86.112	86.112
spamassassin	false	4	86.112	86.112
spambayes	true	5	83.63	83.583
spambayes	true	4	83.232	83.232
spambayes	false	5	83.021	83.021
spambayes	false	4	82.282	82.482
spamassassin	true	7	81.546	81.522
spamassassin	false	7	81.546	81.546
spambayes	false	3	81.405	81.405
spambayes	true	3	78.618	78.618
bogofilter	false	4	77.049	77.049
bogofilter	true	4	77.049	77.049
bogofilter	true	5	72.248	72.248
bogofilter	false	5	67.728	67.728
spambayes	true	7	54.965	54.965
spambayes	false	7	54.075	54.075
spamassassin	true	9	44.965	44.988
spamassassin	false	9	44.824	44.824
bogofilter	true	7	20.937	20.937
bogofilter	false	7	20.141	20.141
spambayes	true	9	14.309	14.239
spambayes	false	9	14.192	14.192
bogofilter	true	9	5.691	5.691
bogofilter	false	9	5.691	5.691

Table VIII - Logistic Regression Results

Type	Coefficient Estimate	Standard Error	P-value
(Intercept)	3.2777149	0.0141475	< 2e-16
filter_tool_spamassassin	1.5810554	0.0123121	< 2e-16
filter_tool_spambayes	0.5601675	0.0109058	< 2e-16
normalized_true	0.0609702	0.0093141	5.91e-11
chain_length	-0.5735802	0.0020611	< 2e-16
stacked_chainsTrue	0.0003469	0.0093129	0.97

As can be seen in this logistic regression, in Table VIII, stacked chains are not statistically significant, while filter tool, chain length and normalized numbers are very significant predictors of the correctness of the filter. This supports the thinking that the chosen filter tool is important, particularly in showing that SpamAssassin is the most effective of the filters. Chain length is significant, but its effect is negative, suggesting that longer chain lengths are to be avoided. Normalized numbers are also significant.

The summary table, Table VII, shows, in particular, that filter type and chain length are the most effective combinations of variables, as seen for SpamAssassin with word chains with a length of 2 or 3 words. In the right combination, the filter scores at 99.906% effective in correctly matching the test records to the trained record types. A chain length of 1 and no number normalization, which is the "untreated" data, only gets 86.628% correct. This makes for a 13.278% improvement in differentiation for this log set.

SpamBayes does best with data that have only had numbers normalized, scoring 96.815% correct; it fared only a bit worse with completely "untreated" data, at 90.703%, the manipulation making 4.112% improvement.

Finally, it is interesting to note that Bogofilter does the same with completely "untreated" data and data that have had numbers normalized, both scoring at 92.436%. All word chaining was detrimental to this filter, so its "untreated" data score was its best.

Bogofilter's "untreated" score was actually 5.808% better than the "untreated" score from SpamAssassin, and 1.733% better than the "untreated" message score from SpamBayes. At its best, SpamAssassin was 3.091% better than SpamBayes' best score and 7.47% better than Bogofilter's best accuracy score. These sorts of differences can mean thousands or tens of thousands of lines in the case of even relatively large log sets.

All of this suggests that there is not really a single way to treat data that is optimal for all three filters, aside from the use of very short chains (of length 1 for both SpamBayes and Bogofilter). These findings were useful for the next stage, where actual, non-trivial, log filtering would be tested, with the full range of actual logs from a production application.

I suspect that the lack of differentiation from chain stacking is caused by the filters only selecting the most "interesting" words in the token set. This is a common defense against so called "word salad" spam where random words are placed in the message body.

4.1.2 Actual Log Entry Testing and Outage Record Comparison

Finding a corpus of log files for actual log entry testing turned out to be quite difficult. Most organizations do not keep log entries for very long, often 7 days or less. Since outages for any given server or service are a relatively rare occurrence, with any given server or service failing perhaps once every month or two, finding problems within a given system's log retention window proved to be difficult. Finding outage records that repeat similar outages was even more difficult.

However, I had the great fortune of obtaining two sets of logs from the Family History Department (FHD) of The Church of Jesus Christ of Latter-Day Saints (LDS Church) which runs the FamilySearch.org web site. This department runs applications which reside on hundreds of servers, each of which logs heavily.

For set one, four outages with similar symptoms occurred in late March, 2011. Symptoms included a spike in network traffic and network errors, and eventually the shared SAN-mounted filesystems gave errors and dismounted. Related entries were pushed to local syslogs. The short-term solution was to restart networking or restart the affected server; the root cause is as yet unknown as of this writing. The logs looked similar for the time frames, so these logs were used to ask, "Can the filters detect log entry similarities which are seen by an administrator?"

For the second set, in the November of 2010, FHD experienced four outages with similar symptoms: thread starvation in the search application was associated with a cascading failure in a search cluster. This cluster logged heavily and was set to retain logs for over a month. The lead system manger from IT operations believed that these outages were all related, so this set of outages became the other research target, with the prime question being, "Are these outages in fact related?"

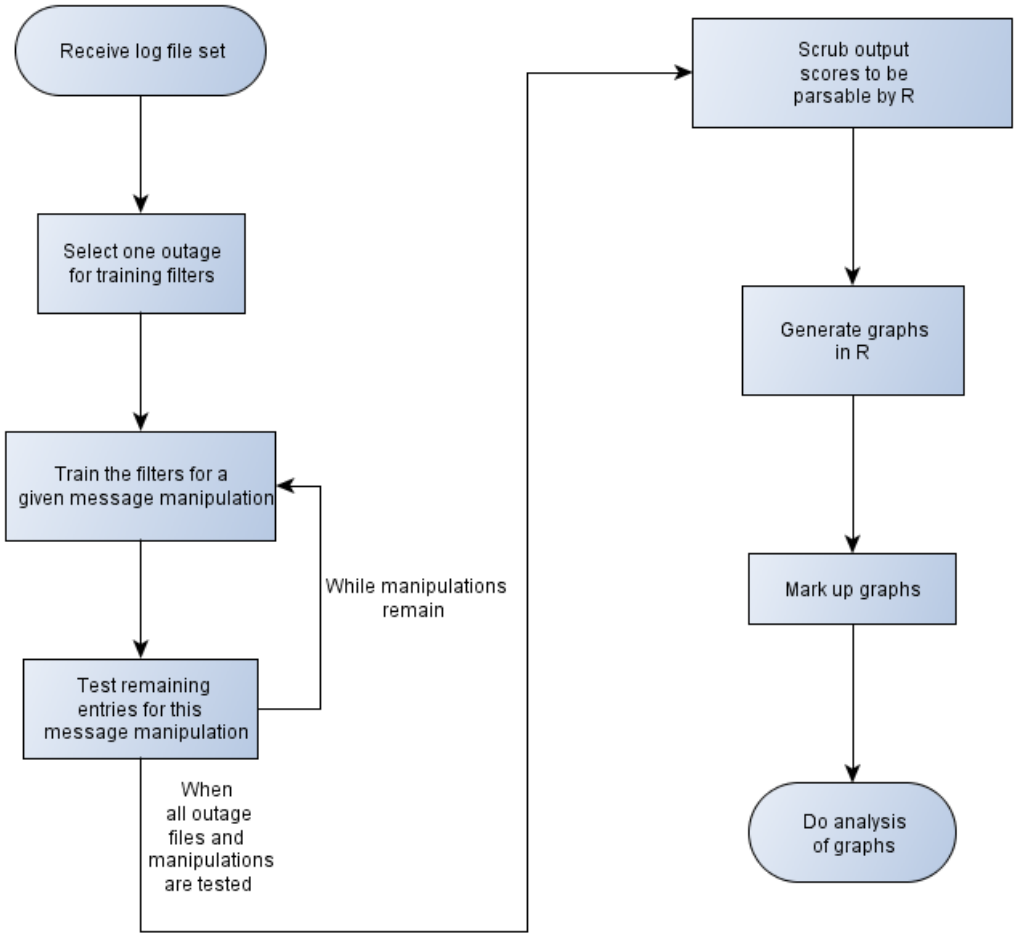

Figure 6 - Full Log Testing Flow

The general flow chart for these analyses, including the looping iterations, can be seen in Figure 6.

4.1.2.1 Spring Outage Syslog Analysis

To answer the question of whether the filters could be trained on log entries from one outage and then detect other related outages from their log entries, the Spring 2011 outage syslogs were used, as the relatedness of the outages was already determined by looking at the logs.

These outages occurred on two systems, three on app2 and one on app1, as the application was moved from app2 to app1 and then back again in an attempt to rule out hardware as the cause of the problem. App1's syslog file contained 8,070 lines of syslog entries and app2's syslog file contained 30,555 lines. Time-wise, the app2 logs ran from Feb 27 04:02 to Mar 31 22:26, and the app1 logs ran from Mar 13, 04:02 to Mar 31 22:29.

The documented outages occurred on March 18th, 23rd, 25th and 30th. The outage of the 25th was randomly selected from the set and used to train the filters. The original log files were fed separately into the filters with all combinations of normalized numbers (true or false), chained word lengths (1-5 words) and stacked chains (whether only the output of the specified chain length was used, or all the chain lengths up to the specified length were included, or not).

The detected outage for the 25th started at 17:03 UTC and was not resolved until 18:11 UTC. (Syslog entries from these servers use UTC, the Coordinated Universal Time standard from which world time is calculated, based on time at 0° longtitude.) All 30 lines of log entries from that time frame were specified as spam for the filters; 31 lines taken randomly from the remaining 30,525 lines of that log file were specified as ham for the filters, using randlines.py, printed in the appendix. Bayesian filters are thrown off when the samples are of widely different sizes; the one extra ham entry was a mistake when the sample was taken, and not discovered util after the work had been done, but not considered a problem for the balance of the filter.

The logs were run through l_train.py and l_test.py (available in Appendix A – Program code and Templates) using a small shell script to specify the variations used for testing (number normalization, chain length and chain stacking), and to train and test them with all three filters (SpamAssassin, SpamBayes and Bogofilter).

Scatterplots were used in the analysis because they can compactly represent the tens of thousands of numbers in the score output, and do so in a way which is intuitive for the pattern-recognition skills of the human mind. The graphs are not perfect, as tens of thousands of values must be squeezed into just 1200x1000 pixels, meaning that adjacent scores can be lost, visually. To assist the graph viewer, the regression line and lowest smooth and smoothed conditional spread were also fitted by the graph function chosen (scatterplot from the R "car" package (Fox 2011)); these allow some of the larger patterns to be seen more easily on the various graphs. Additionally, box-and-whisker plots were produced in the plot margin; these plots can give additional insight into the data, as they show the distribution of the scores in each axis.

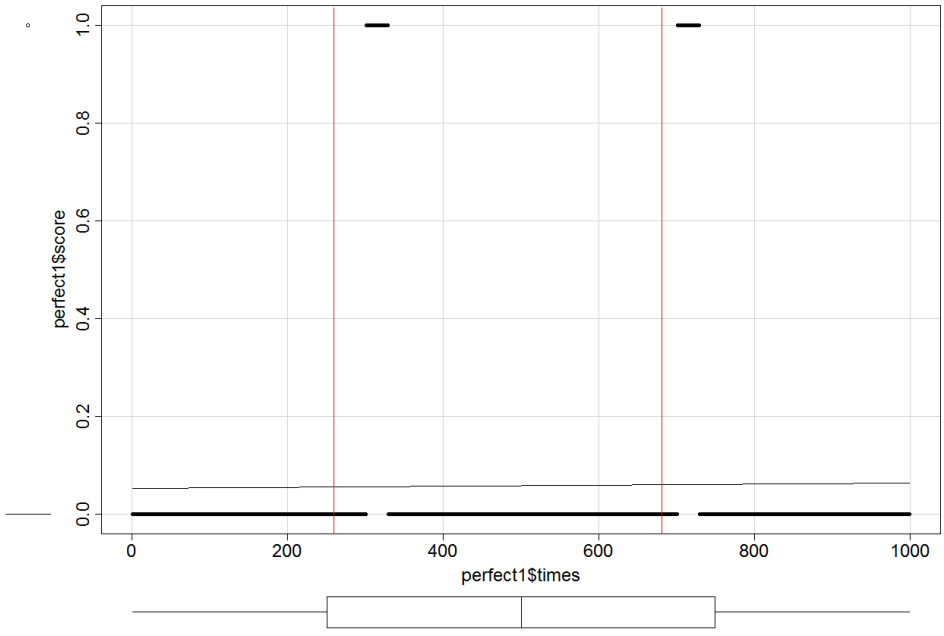

Figure 7 - Simulated Graph of Scores from a Perfectly Effective Filter

If the three filters were perfectly effective at recognizing outage-related records, then records directly related to outages would have scores at the "spam" level (e.g. 1 in SpamBayes or

Bogofilter), and scores at the "ham" level (e.g. 0 in SpamBayes or Bogofilter) for all other records. In a hypothetical scenario of 1000 records, with 30 records each at 300 and 700, the generated graph would look like the simulated values in Figure 7.

Note that vertical lines, used as a general visual guide to where the outages occurred, were created with the R abline(v=300,col="red") and abline(v=700,col="red") functions, perfectly matching the simulated data (i.e. lines 1-300,331-700,731-1000 had scores of 0 and lines 301-330,701-730 had scores of 1). These function calls set vertical lines 300 and 700 pixels from the left edge of the image rather than from the zero point of the graph; additionally, the horizontal scale of the graph is not one value per pixel. Therefore, there is a horizontal offset of the vertical lines from their associated "outage" score sets, which becomes closer to correct alignment as we move to the right of the graph. In spite of this visual offset, the vertical line is helpful in guiding the eye to the "outages" (both in this graph and in the actual graphs later on), so the lines will be included. The reader is assumed to be able to see the pattern and use the vertical lines as a general guide for the eyes.

For contrast, a perfectly ineffective filter would not be able to differentiate any lines and would return either all the same score (e.g. a horizontal line of some value from zero to one), or a random set of scores as in the simulated values from Figure 8.

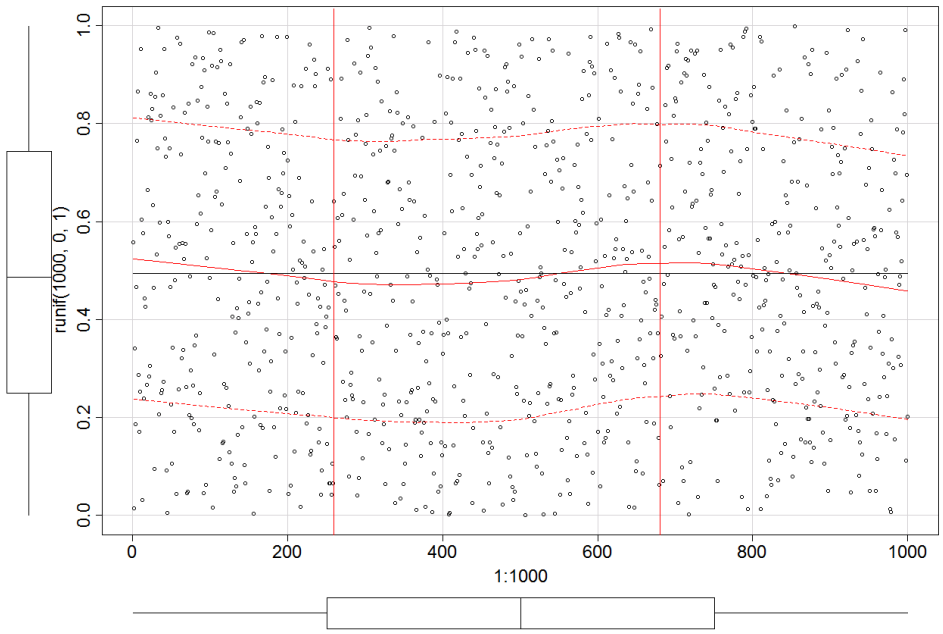

Figure 8 - Simulated Graph of Scores from a Perfectly Ineffective Filter

If one were to create a measure of effectiveness for these filters, three factors would weigh in more than any others: effectiveness at recognizing "spam" log entry types from "ham" log entry types; the "noise" level of scores for records which do not match the "spam" set; and the ease of implementing such a filter. The first of these factors is obvious and clearly most important—a filter which cannot differentiate is of no use. The second factor becomes obvious as one looks at the major difference between Figure 7 and Figure 8; if scores are all over the gamut, "spam" score patterns may be difficult to recognize even if their record scores are numerically differentiable from "ham" scores. The third factor becomes obvious when one

thinks operationally: nobody will use the technique at all if it is too difficult to implement or too difficult to understand. These factors would not reasonably weigh equally. On a scale of 0 to 10, spam/ham recognition might reasonably represent 5 of the 10 points (i.e. a scale of 0-5 representing differentiating ability). The "noise level" of output might represent 3 points (0=scores all over the score gamut, 1 and 2=scores scattered through less of the score gamut, 3=low levels of score spread in the score gamut). And the implementation difficulty of such a filter might be represented by 2 points (0=very difficult to implement and interpret, 1=less difficult to implement and interpret, 2=easy to implement and interpret). This scale will be referred to as the Filter Effectiveness Scale (or FES) going forward.

On such a scale, assuming implementation difficulty in line with the other two differentiating factors which are obvious in the graphs, the ideal filter that would generate Figure 7, and the completely ineffective filter that would generate Figure 8, would receive FES scores of 10 and 0, respectively.

4.1.2.2 **Spring outage actual scored log entries**

The actual scored log entries, as output by the three spam filters, used textual time/date stamps, so these values were converted to Unix epoch time values for graphing. Unix epoch time is the number of seconds since midnight (00:00) January 1, 1970. This long integer, being a numeric, is easly used in scatterplots and the like, and is less likely to cause parsing problems for R. Actual times and dates of the x-axis markers have been added to help the reader understand the timelines of the log entries. Patterns in the graphs have been circled to make them more obvious to the reader.

The output scores of these programs was graphed using the so-graphs.R script, found in the program's appendix. The times of known outages were marked as vertical red lines on the

graphs for reference; blue lines were also added to represent the end-times of outages. These start and end times were taken from the trouble-tickets written by operations personnel in the Family History department, and are approximate, being tied to the time when the server engineer was notified of the problem and when the problem was perceived to be resolved. Remember that these are very approximate, due to the offset and horizontal scaling issues discussed before, so they are used only to guide the eyes to the graph area where issues occurred.

Figure 9 through Figure 11 show the three filters' scores of the app2 log, using no word chaining (chain-length=1), no number normalization, and of course no chain stacking. Essentially the only manipulation of logs was to put the log message body into an email template for the filters (the same template was used for all training and testing). Figure 12 through Figure 14 show the same graphs for the app1 log files; these logs cover the last half of the month.

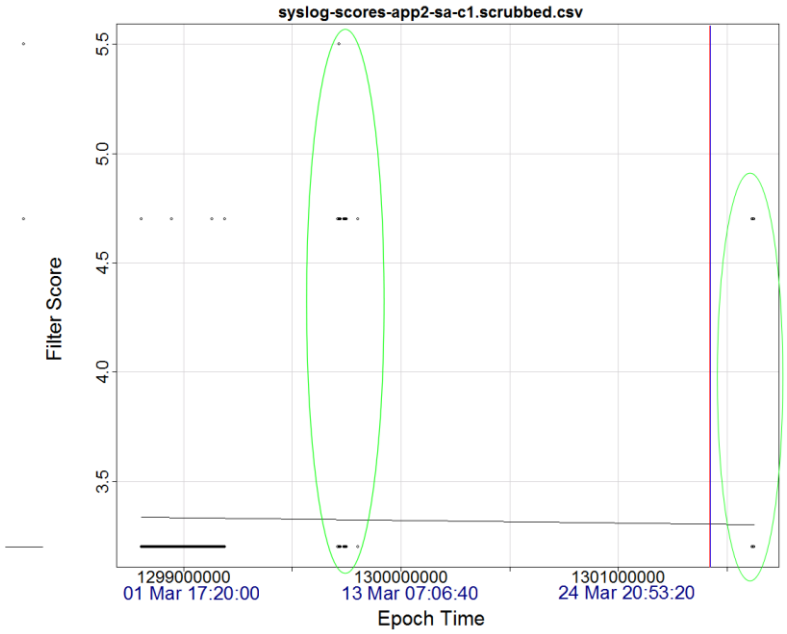

Figure 9 - SpamAssassin – App2, Chain-length 1, No Normalization, Marked .

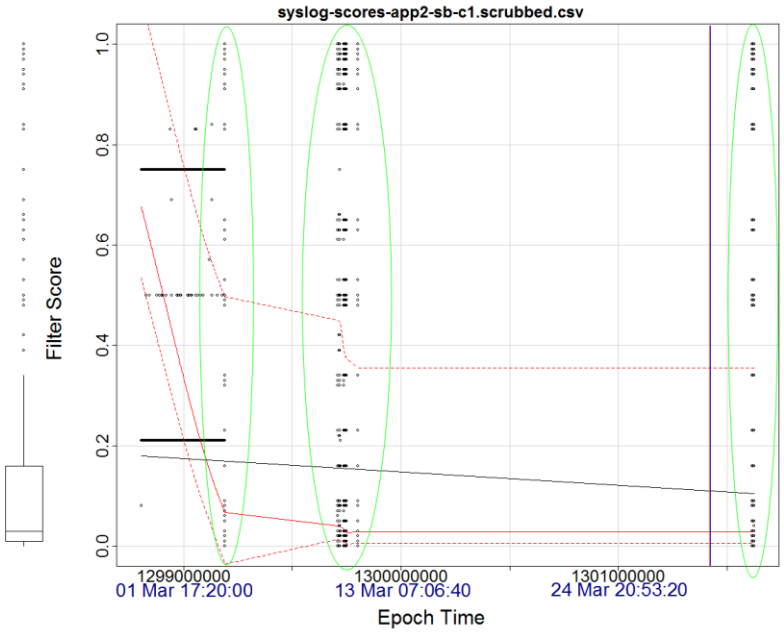

Figure 10 - SpamBayes – App2, Chain-length 1, No Normalization, Marked

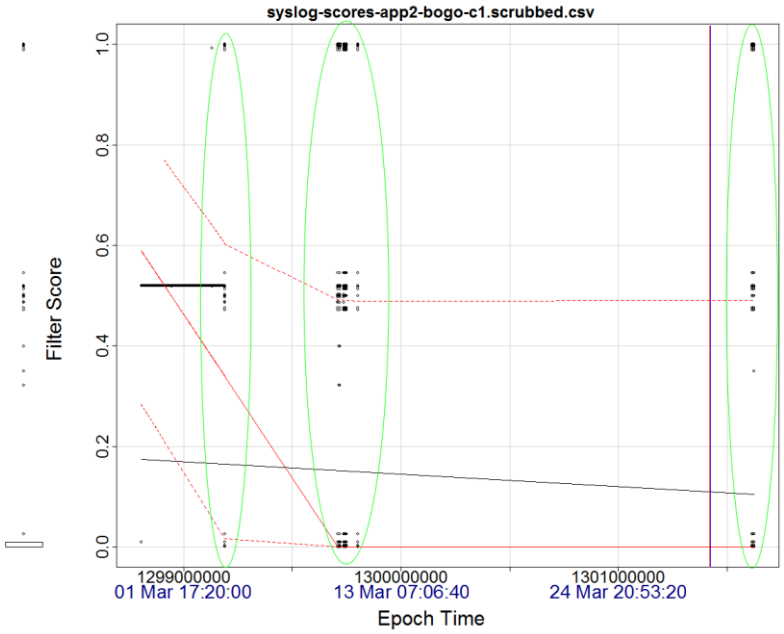

Figure 11 - Bogofilter – App2, Chain-length 1, No Normalization, Marked

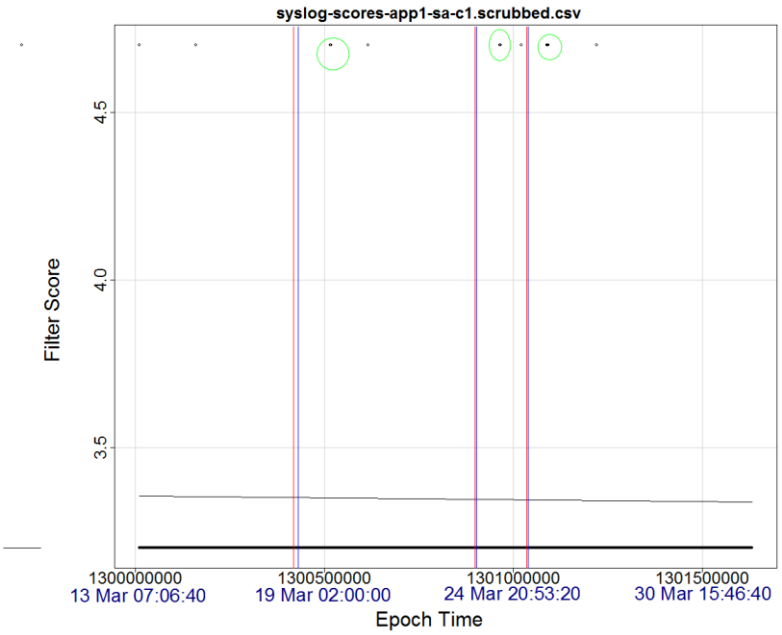

Figure 12 - SpamAssassin - App 1, Chain-length 1, No Normalization, Marked

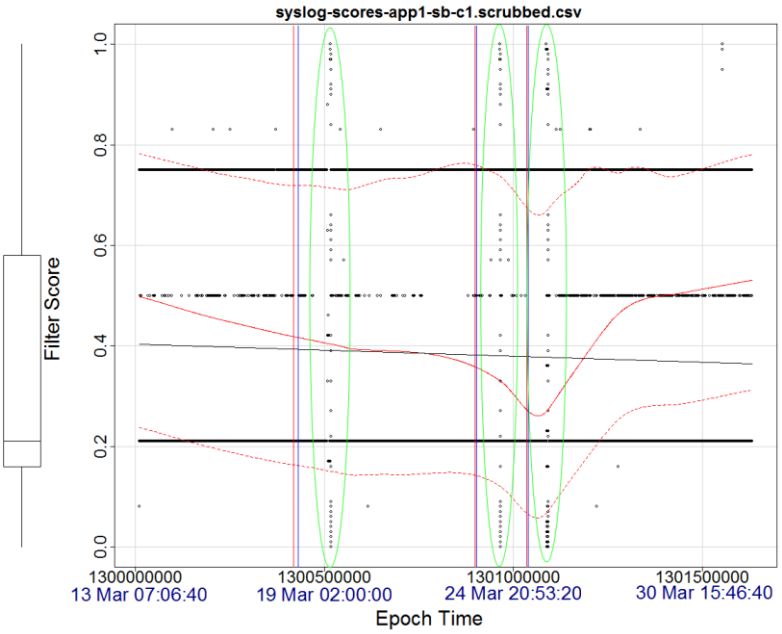

Figure 13 - SpamBayes - App1, Chain-length=1, No Normalization, Marked

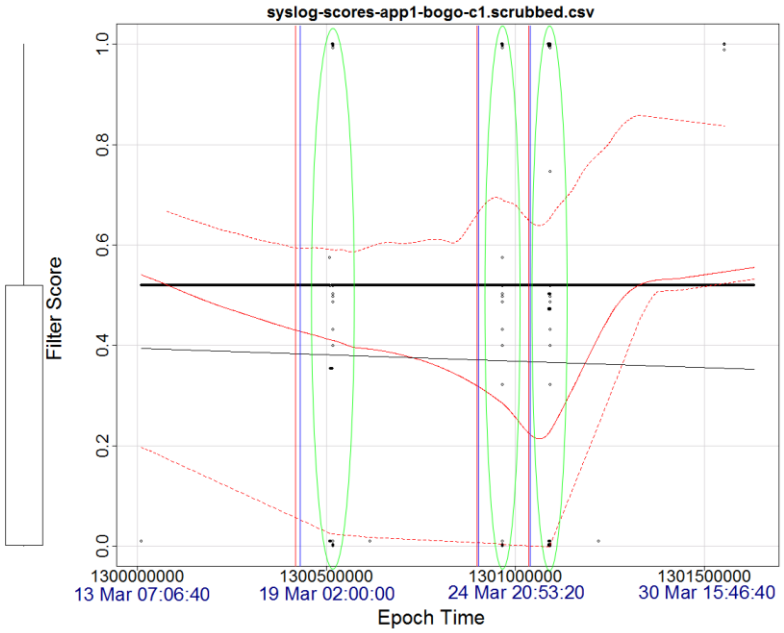

Figure 14 - Bogofilter - App1, Chain-length=1, No Normalization, Marked

Between the two logs, three of the four outages have associated vertically-spread datapoints which are very plain in the SpamBayes and Bogofilter graphs (the SpamAssassin graph will be addressed shortly). In fact, there are similar patterns on the app 2 graph suggesting that the problem occurred at those times as well. Since these servers are in transition, and management of the servers is shared between IT operations (which documented the 4 outages used here), and development (which has not documented any outages, but says there have been several), this is not only possible, but quite likely. A quick scan of the logs in those time frames suggests that the servers indeed did have problems then and were restarted to resolve them.

Recall that the scores returned by the three filters are driven by whether the words in the log messages are more similar to those in the spam record set or the ham record set. The dozens of log entries for each outage time frame include more words from the spam set than is normal,

which skews the score higher, depending on how many spam words are included in the message. The pattern seen in the SpamBayes and Bogofilter graphs for those time frames, is a more or less vertical line of dots, indicating a number of records close in time which have a higer proportion of spam-like words (i.e. words from problem records which were used to train the filter as "spam"). These patterns are quite plain to see, but are marked on the graphs to increase their visibility. They are also fairly plainly correlated with the four outages noted by the vertical red lines, and seem to show several other, earlier, undocumented outages on app2 during the time frame of the graphs.

Following the scoring criteria listed above, both SpamBayes and Bogofilter would score 5 for ability to differentiate "spam" records from "ham" records. However, SpamAssassin would have to score a 0: while there are points at the expected places on the graph, they are not differentiable from other points which appear to be unrelated to the outages. As far as score noise goes, SpamAssassin has very little, while SpamBayes and Bogofilter have a bit more, giving them scores of 3 , 2 and 2. All three filters score a 2 for ease of implementation: a look at the code in Appendix A will show that they are all about the same difficulty with integration, and all three filters are widely available free tools, with straight-forward command-line interfaces. Thus these scores might receive the following FES scores: SpamAssassin 5, SpamBayes and Bogofilter 9 and 9.

SpamAssassin, which in earlier tests was so effective at differentiating logs, seems to not show the outages very clearly – just a few darker areas of data points with high scores. However, when the points are jittered, the number of points in these areas becomes more clear, showing that this filter does indeed detect the outage-related records, as seen in Figure 15 and Figure 16. SpamAssassin is assigning the same high (4.7) and low (3.1) values to all of these log

entries, making its score graph actually closer to the ideal graph (Figure 7) than SpamBayes and Bogofilter.

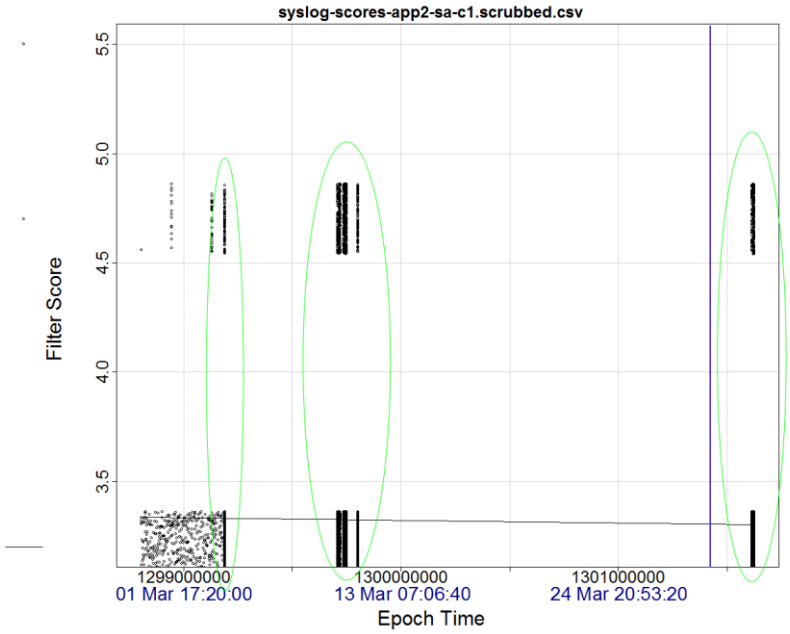

Figure 15 - Jittered SpamAssassin, App2, Chain-length=1, No Normalization, Marked

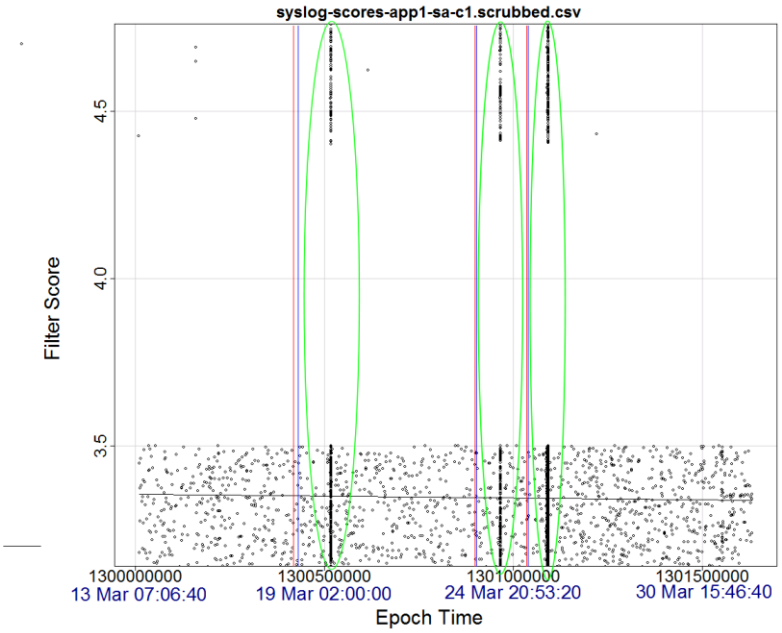

Figure 16 - Jittered SpamAssassin, App1, Chain-length=1, No Normalization, Marked

Thus the SpamAssassin scores might be reasonably adjusted to 5 for effectiveness and 2 for noise, for a FES score of 9, the same as the other two filters, at least with this data set. This also brings out the fact that the score patterns for SpamBayes and Bogofilter are actually easier to see on a graph than the score pattern for SpamAssassin, even though the SpamAssassin pattern more closely matches the "perfect filter" graph.

Differences in the filter effectiveness and internal scoring mechanisms of the three filters explain the differences in graphs, including SpamAssassin's score specificity. Overall, the vertical graph patterns which represent higher filter scores (whether seen with or without jittering) are clear in each graph.

Considerable time was taken in earlier parts of this thesis to show how various manipulations might positively impact the effectiveness of these Bayesian content filters in

detecting differences between log entries. The unmodified text has been shown sufficiently above, but for comparison, Figure 17 through Figure 19 show graphs where words were chained together in 3-grams (e.g. "Now is the time for" becomes "Now_is_the is_the_time the_time_for") and run through each filter. Note that in all three cases, the graphical attributes are less plain to discern than in the non-chained text case, though the SpamBayes graph is still plain. In the trivial log entry testing, only SpamAssassin benefited significantly from chained words, and it has shown to be ineffective in differentiating records in this test already, so the overall ineffectiveness of chained words in this case is not unexpected.

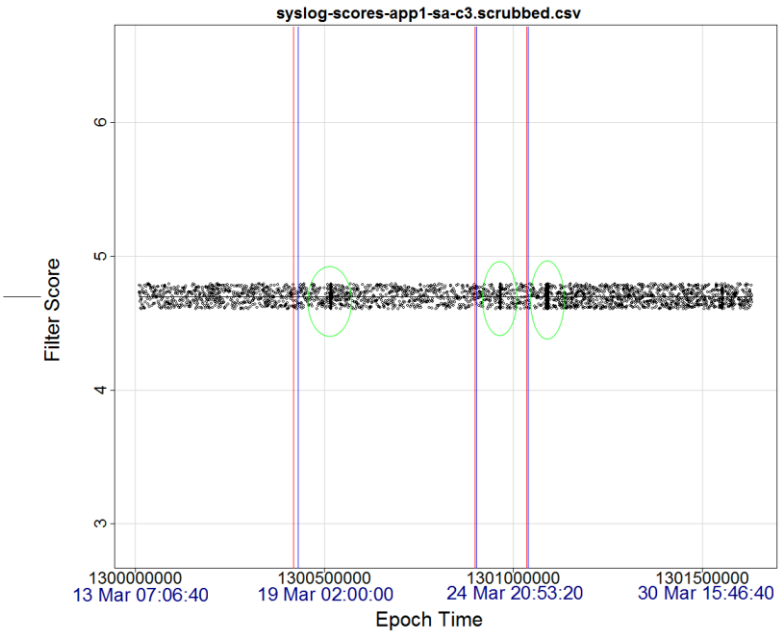

Figure 17 - SpamAssassin (jittered) - App1, Chain-length=3, No Normalization or Stacked Chain

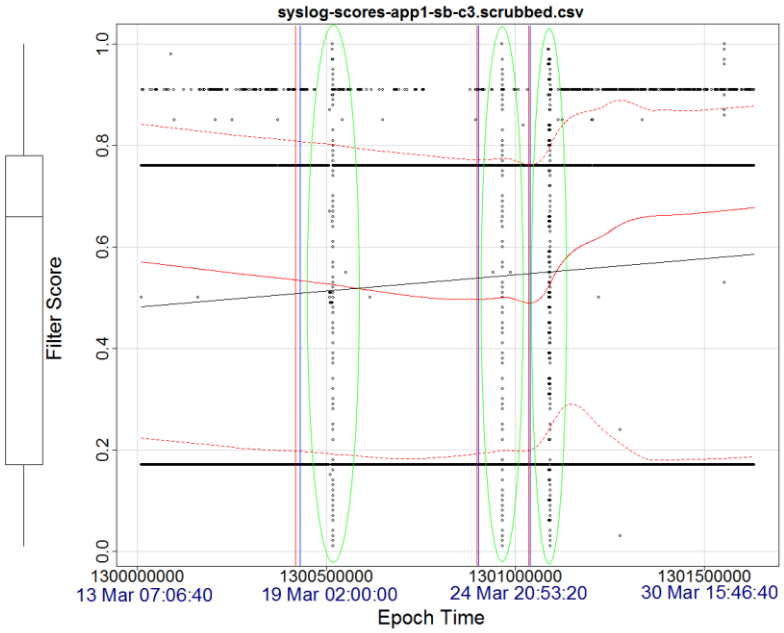

Figure 18 - SpamBayes - App1, Chain-length=3, No Normalization or Stacked Chain

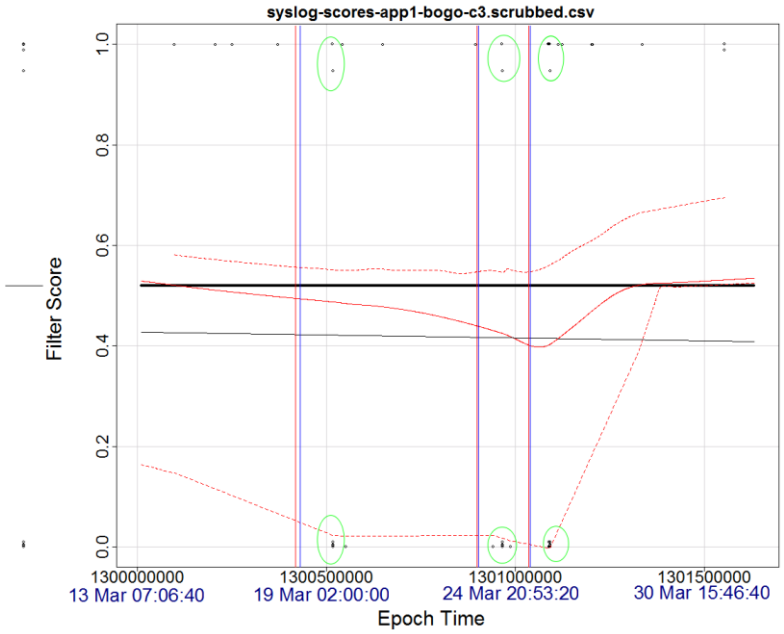

Figure 19 - Bogofilter - App1, Chain-length=3, No Normalization or Stacked Chain

The SpamAssassin scores were in the same bimodal distribution that we saw with the non-manipulated data, so the graph jittering treatment makes sense here as well.

These data manipulations, while increasing the difficulty of integration (dropping integration ease to 1), actually push the differentiation and noise scores for SpamAssassin down, to perhaps 2 and 1 (for a FES score of 4); SpamBayes is largely unaffected (except for ease of integration going to 1, with a FES score of 8), while Bogofilter's apparent effectiveness goes down to a 2 (with noise unaffected, for a FES score of 6). The chained words manipulation does not appear to be helpful for this particular task, especially for SpamAssassin and Bogofilter.

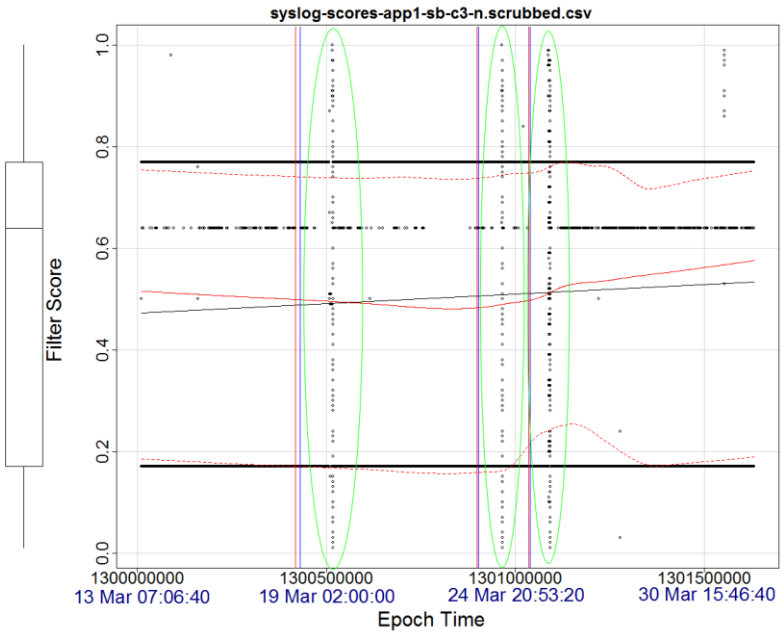

Figure 20 - SpamBayes - App1, Chain-length=3, No Stacked Chains, Numbers Normalized

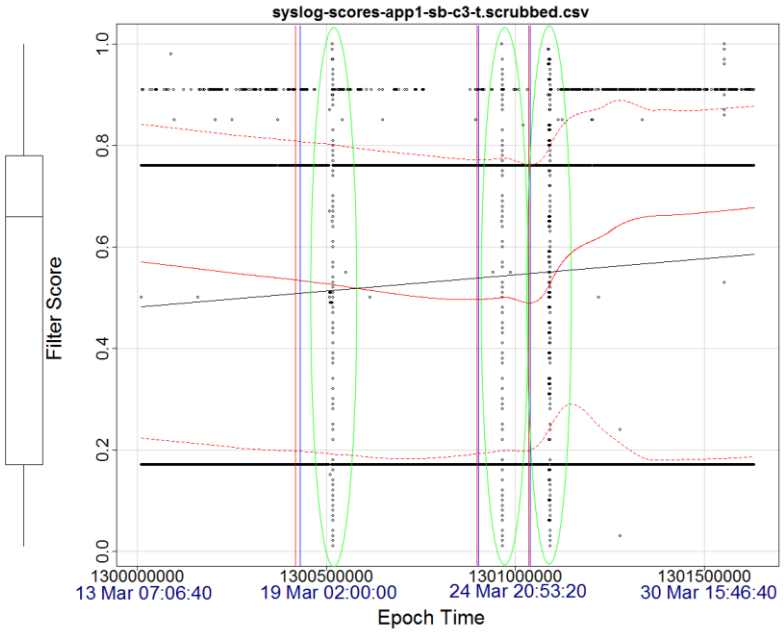

Figure 21 - SpamBayes - App1, Chain-length=3, Stacked Chains, No Numbers Normalized

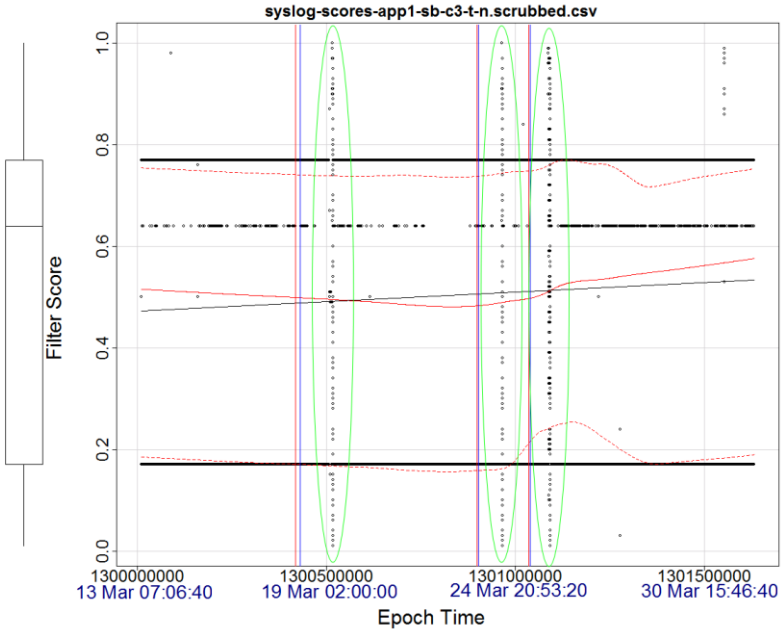

Figure 22 - SpamBayes - App1, Chain-length=3, Stacked Chains, Numbers Normalized

Figure 20 through Figure 22 cut the other way across the manipulations tested, showing the differences between SpamBayes graphs at a chain length of 3 with chains stacked, numbers normalized to zeros and both. SpamBayes graphs were chosen because they showed the most variability overall and thus the differences would be easiest to see with them. Note that stacking the chains made no difference whatever with these, and that the normalized numbers made a very slight difference, moving only one horizontal line of entries up on the graph. Only some entries have relatively differentiable numbers (such as IP addresses, MAC addresses, etc.) which would affect the entry's scores; thus most entries were not affected by normalizing the numbers to zeros. (For examples of these sorts of lines, see the log entries in Figure 2 and Figure 3.) The moved line of points is likely the set of records with IP addresses and the like that are affected by this change. It is interesting that they all move together, suggesting that the training data set causes them to be recognized as more spam-like as they come through as zeros, however the moved points do not clarify when outages occur, and are not directly impacting of this analysis.

The chain stacking and number normalization manipulations again increase the complexity of integration (dropping those scores to 1), but do not appreciably impact the score levels of the filter; the same was true of the other two filters.

In summary of the above scatter plots, almost all of the score scatterplots for SpamBayes and Bogofilter showed very discernible patterns indicating outages related to the training record set. SpamAssassin was similarly effective, but was so consistent to its scoring that the values had to be jittered in the graph before its Bayesian content filter's effectiveness became plain to see in the graphs.

4.1.2.3 **Fall Outage Applicaton Log Analysis**

With the effectiveness of the filters established, the log entries from the four time periods of Fall outages were taken, logs from the 12th, 16th,18th and 21st of November. These days' logs were very large and unwieldy, so they were trimmed to the hour of the error for two logs (the outage having occurred late in the given hour), and to the hour prior to the outage in question as well as the hour of the outage for the other two logs (the outage having occurred early in the given hour). This was intended to bracket the problem time frames sufficiently. For log entries from the 12th, there were 78,779 lines for the hour in which the outage occurred. There were 167,449 log entries for the hour prior to and hour of the outage on the 16th. There were 78,779 entries for the hour of the outage on the 18th. For the log entries on the 21st, there were 18,654 log entries for the hour prior to and the hour of its outage.

The detected outage for the 12th started at 18:48 (24-hour time format), and there were 33 lines in the 18:47 and 18:48 minutes which were errors that may have been associated with the outage. These were used to train the filters as "spam." There were 75,704 other lines in the logs. Bayesian filters are sensitive to imbalanced training sets, so the original 33 lines were removed from the log file and 33 lines were randomly sampled (using randlines.py) from these remaining log lines to represent "ham" in the Bayesian filters.

Again, SpamAssassin, SpamBayes and Bogofilter were used for the filtering work to be done. The lines were again manipulated prior to training and testing: with and without number normalization, and chain lengths from 1 to 4. Chain stacking was not tested due to the prior tests showing it as ineffective in increasing filter accuracy; the same was true for longer chain lengths, as seen in Table IV. SpamAssassin graphs were jittered to increase the visibility of patterns, as with the spring data.

After training each filter with the specifically modified lines, all the lines for the 4 days' unfiltered log entries were processed through the Bayesian filters, with the resulting scores recorded. These scores were then graphed against the timestamps for their entries; timestamps were again converted to Unix epoch times (number of seconds since beginning of day Jan 1, 1970) so that a simple numeric scatterplot could be used. These runs were automated with l_final_run_stage.sh, which wrapped the previously used l_train.py and l_test.py. The filters left some lines empty or with error messages, so scrubfiles.py cleaned the files up. Graphs were generated with so-graphs.R. Actual dates for the reference epoch time values have been added to the graphs to give a better idea of the timeline on the x-axis.

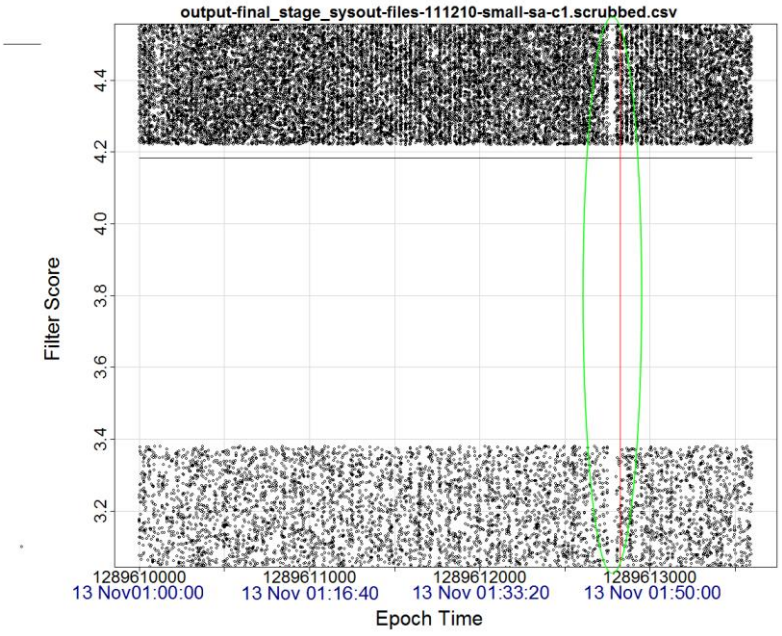

Figure 23 - SpamAssassin (Jittered), Chain-length=1, 11/12

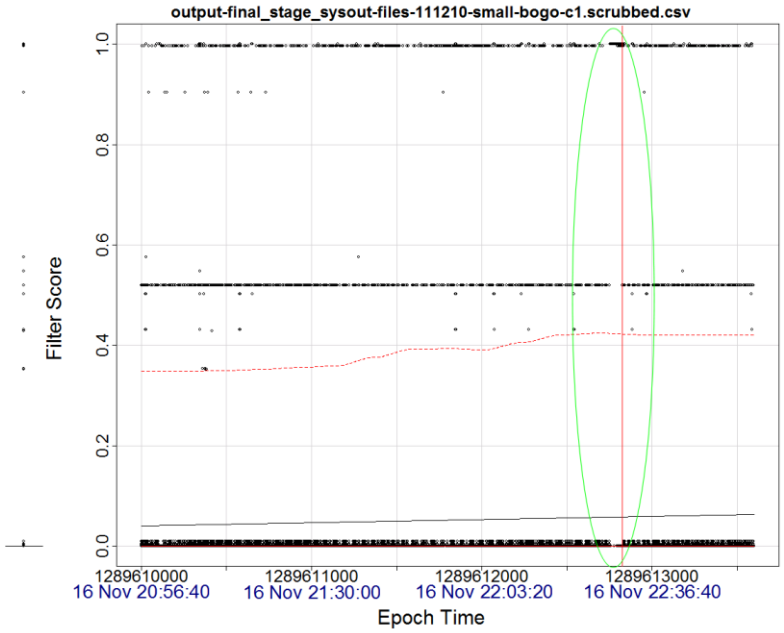

Figure 24 - Bogofilter, Chain-length=1, 11/12

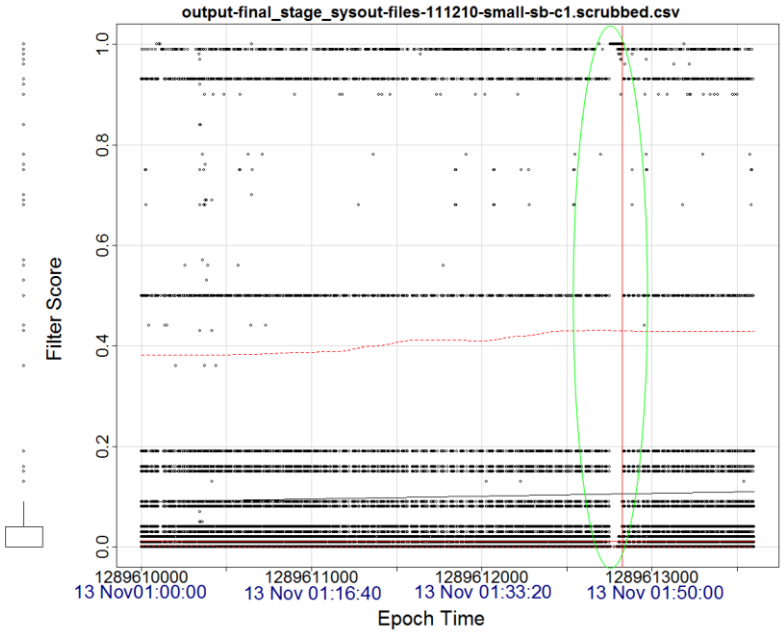

Figure 25 - SpamBayes, Chain-length=1, 11/12

Note in Figure 23 through Figure 25, the change in scores (most plain by the vertical disjunction of points in Figure 25, but also visible in smaller similar disjunctions in Figure 24 and the small break in the areas of points in Figure 23) immediately prior to the outage start time, which is indicated with the vertical red line. Since this was the log set used for training, this was to be expected. This is not what happened on the other days, as shown in Figure 26 through Figure 34. We again see the SpamAssassin bimodal distribution for these scores for all but one of the days, as seen in Table IX.

Table IX - Fall Data SpamAssassin Scores

Log file date	High score	Low score	Number of records with other scores
11/12	4.5	3.1	2
11/16	4.5	3.1	1
11/18	4.5	3.1	1
11/21	1.5	1.5	2

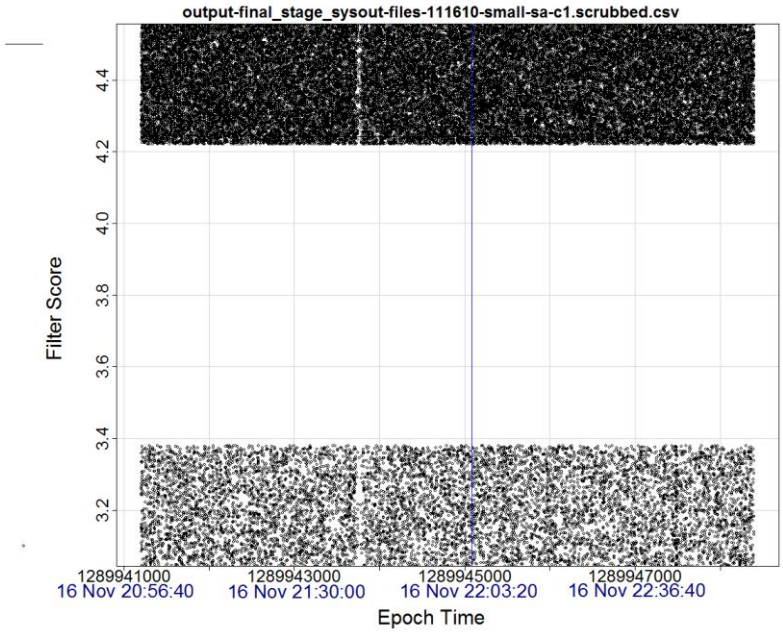

Figure 26 - SpamAssassin, Chain-length=1, 11/16

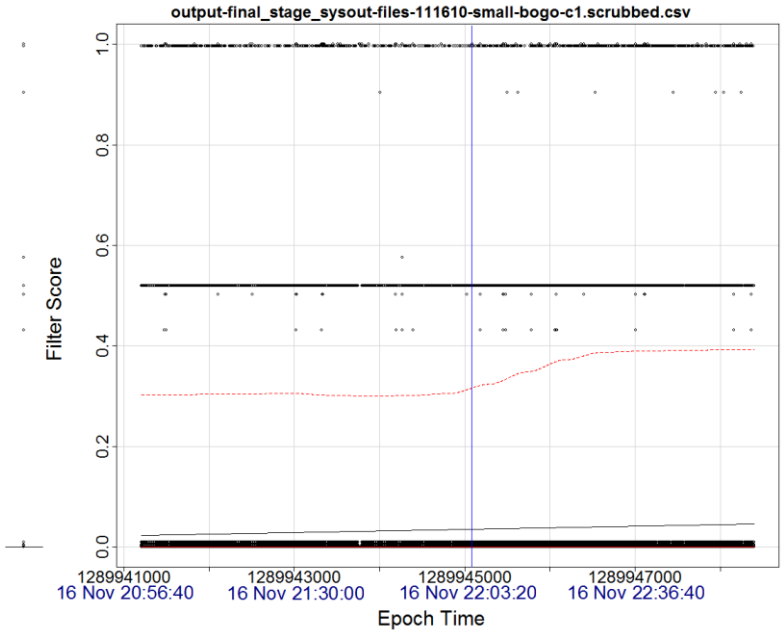

Figure 27 - Bogofilter, Chain-length=1, 11/16

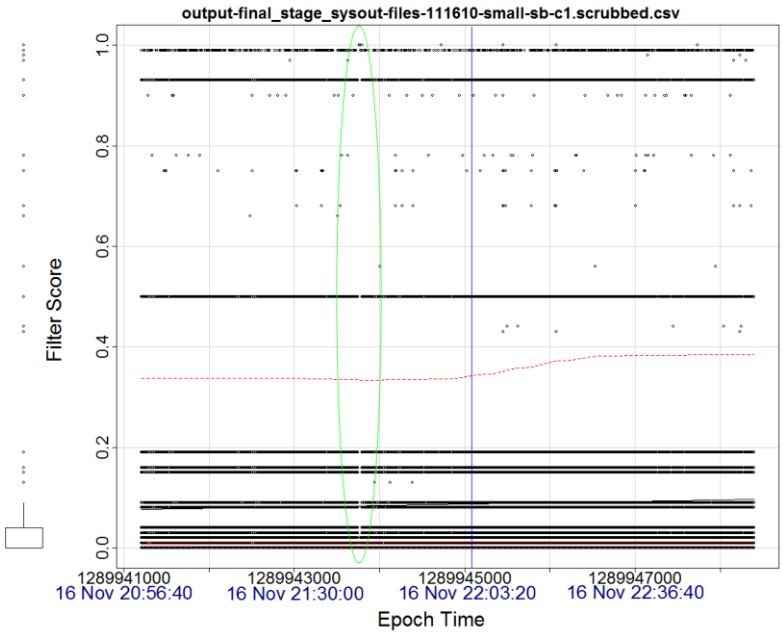

Figure 28 - SpamBayes, Chain-length=1, 11/16

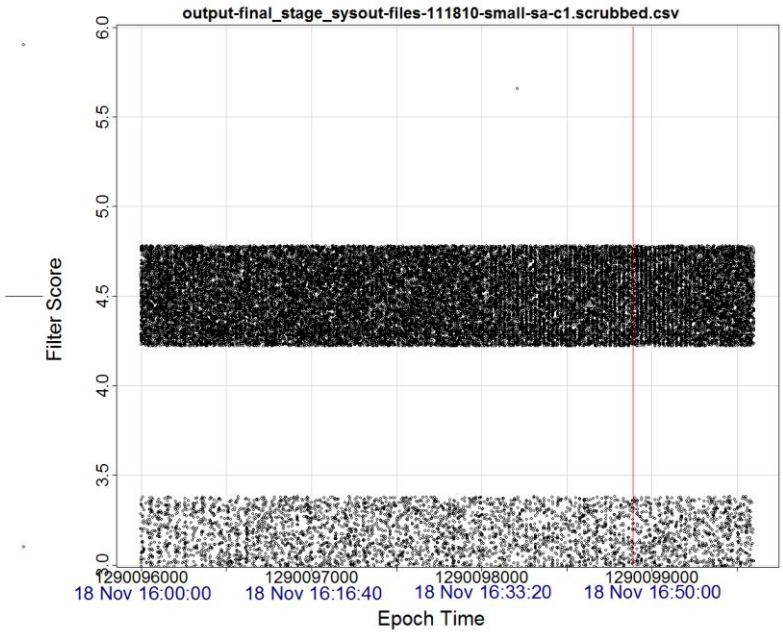

Figure 29 - SpamAssassin, Chain-length=1, 11/18

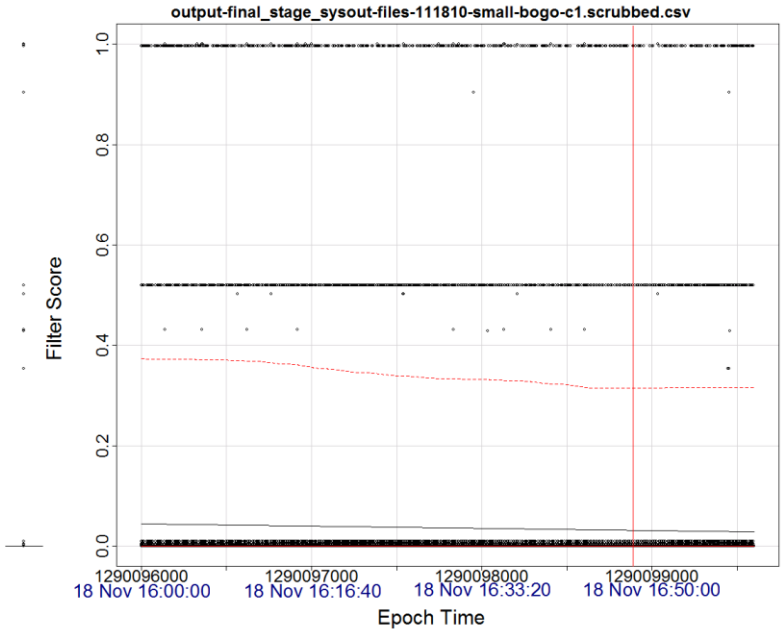

Figure 30 - Bogofilter, Chain-length=1, 11/18

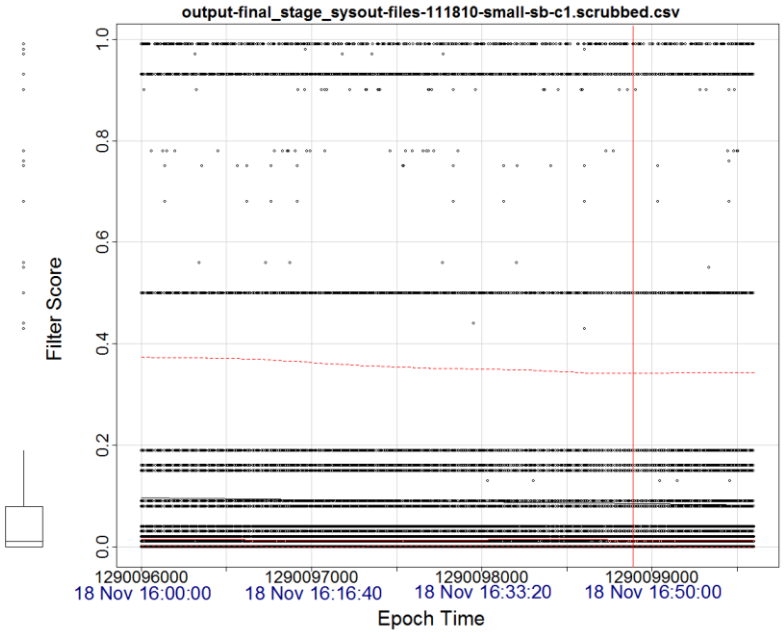

Figure 31 - SpamBayes, Chain-length=1, 11/18

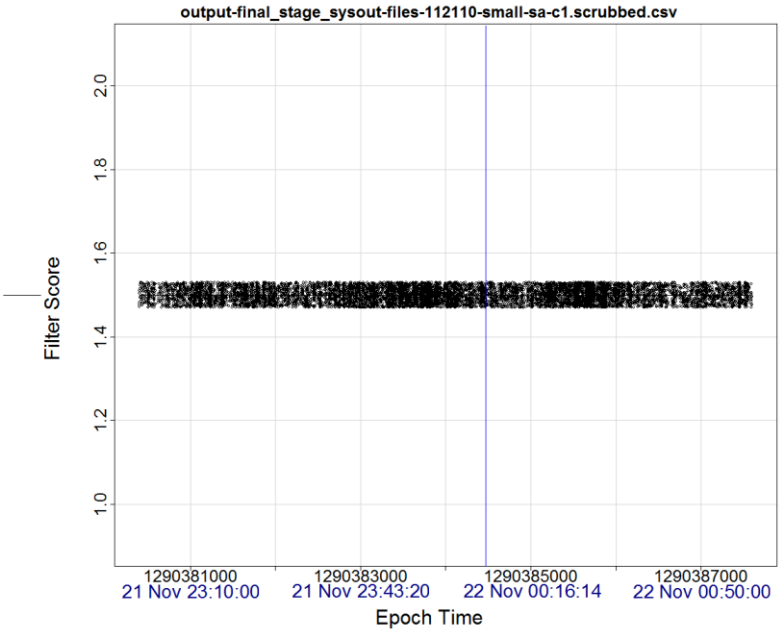

Figure 32 - SpamAssassin, Chain-length=1, 11/21

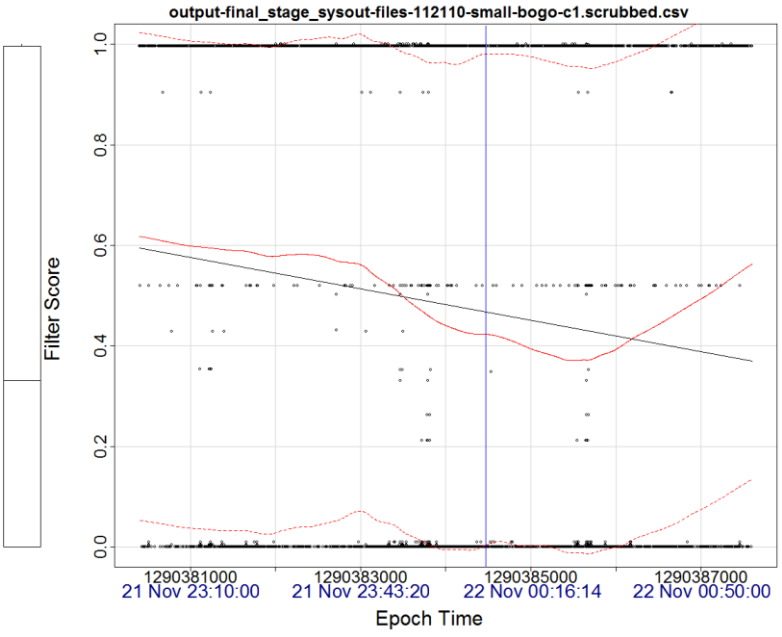

Figure 33 - Bogofilter, Chain-length=1, 11/21

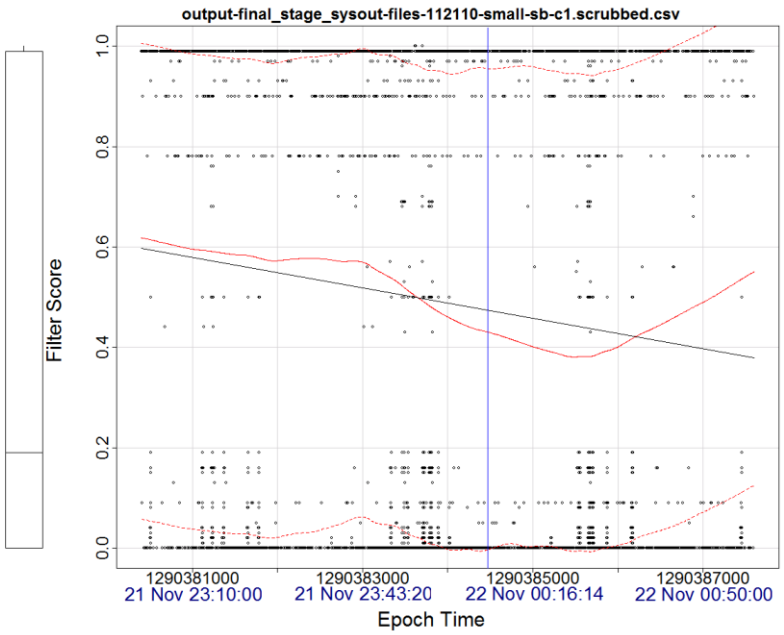

Figure 34 - SpamBayes, Chain-length=1, 11/21

This data set had more data points and more similar records (these being application logs rather than syslogs), and the "score noise" levels reflected these facts. The jittered SpamAssassin graph in particular would have dropped on that portion of the FES scale, to a 6 (3 for differentiation, 1 for noise and 2 for integration ease). All three filters would have dropped in apparent ability to differentiate records, though the score patterns are still visible. SpamBayes and Bogofilter would score 8 and 8 (3 for differentiation, 3 for noise and 2 for ease if integration).

Note that in Figure 26 through Figure 34, though there are changes in some of the smoothed regression lines just before or after the outage time for some graphs, there were no distinct graphical difference in the data points just before the outage times for graphs of scores for 11/18 or 11/21. There was a very similar disjunction of points in the 11/16 graphs, from the

hour previous to the recorded outage time, which would indicate a similar set of log entries to the training set, and thus a similar application state to the outage on 11/12.

As with the spring outages, it is interesting to note the differences in the score graphs for the various tools. SpamAssassin's score set shows only a few score values, and lots of them, spread evenly through the log entry set, thus requiring jittering to be seen clearly in the graphs, while SpamBayes, shows vertically clustered score sets across the timeline. One possible explanation for the SpamAssassin pattern is that it uses relatively few differentiators for the scores, which would not be inconsistent with a tool that uses a suite of tests, only one of which is the Bayesian content filter. For the SpamBayes pattern, one explanation could be that it takes message proximity into account when scoring log entries and that it uses more differentiating factors so that it gets more possible scores. Bogofilter seems to be somewhere between these two distributions, with a largely trimodal distribution, corresponding to its spam/ham/unknown scoring concept.

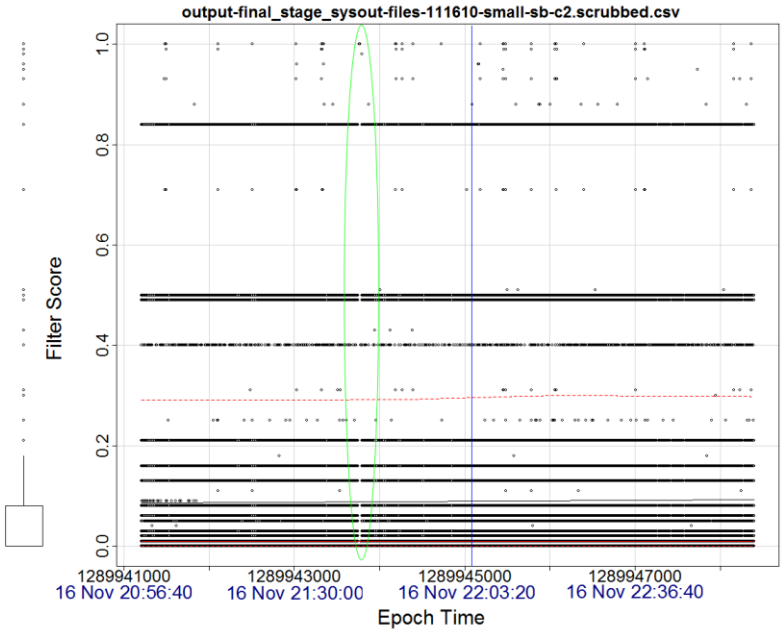

Figure 35 - SpamBayes, Chain-length=2, 11/16

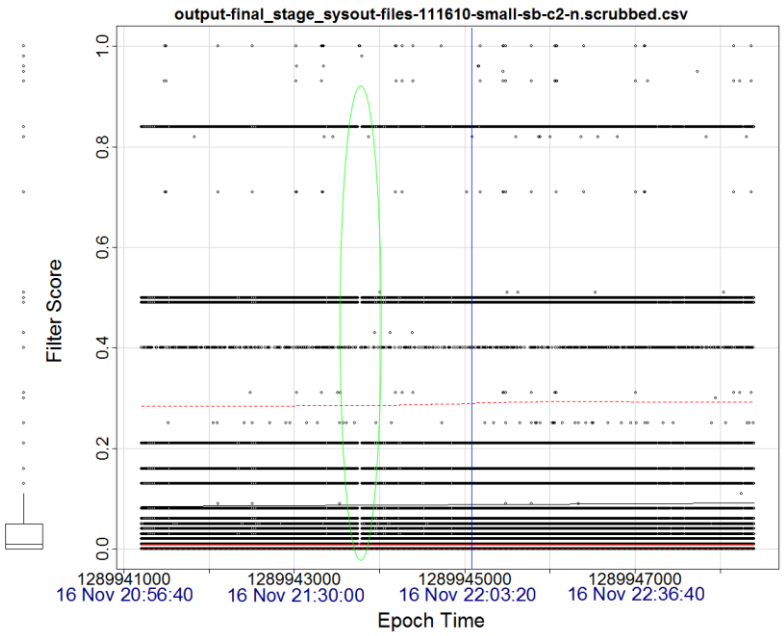

Figure 36 - SpamBayes, Chain-length=2, Normalized Numbers, 11/16

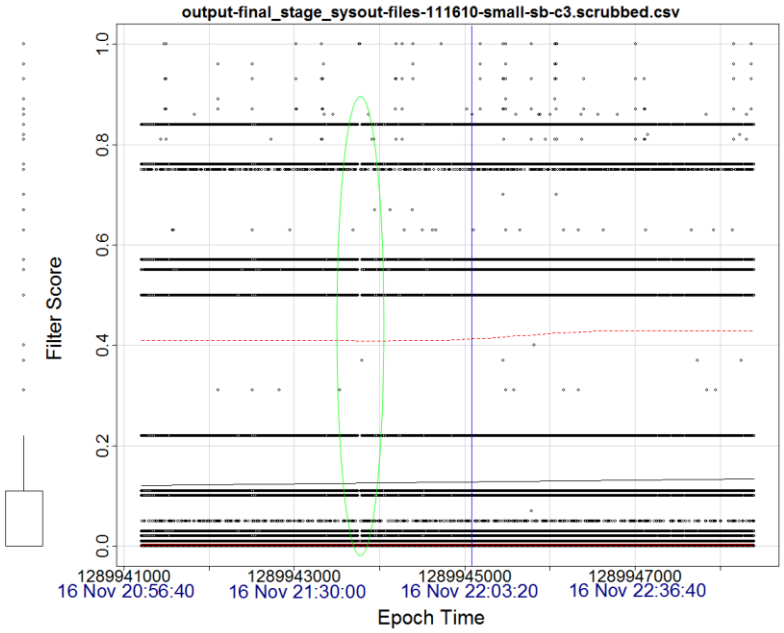

Figure 37 - SpamBayes, Chain-length=3, 11/16

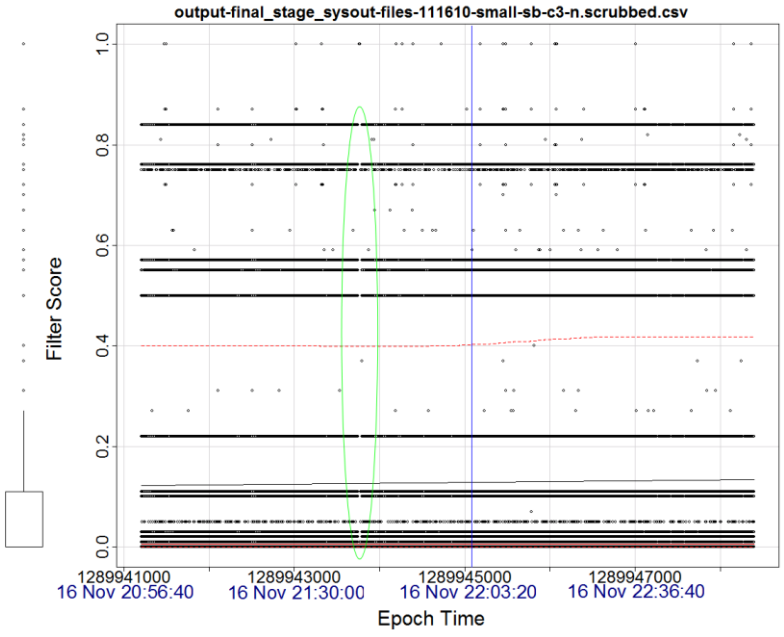

Figure 38 - SpamBayes, Chain-length=3, Normalized Numbers, 11/16

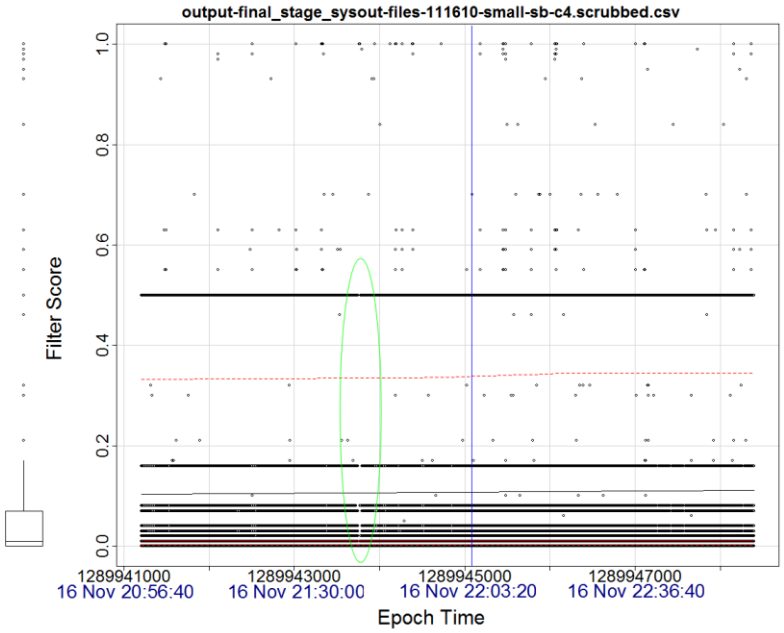

Figure 39 - SpamBayes, Chain-length=4, 11/16

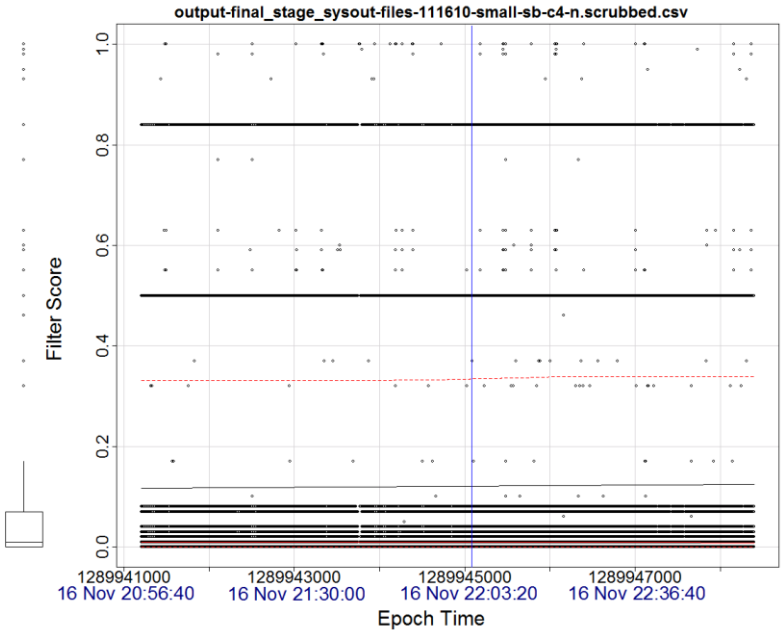

Figure 40 - SpamBayes, Chain-length=4, Normalized Numbers, 11/16

The same held true for the variations in word-chain length, normalization, etc., as seen in Figure 35 through Figure 40, which show these analysis variations for the 11/16 entries under SpamBayes. SpamBayes graphs are shown because they show generally more variation than the other filters' graphs, and its graphs were the most plain in showing the change in filter scores correlating to associated log entries.

These graphs show that the filter did not find log entries from the outages of 11/18 or 11/21 which were the same as those on 11/12. Indeed, the log filter output, with scores appended to the log entries, was closely scrutinized at the times of reported outages by the researcher. The logs entries at those times did not appear to be related to the log entries from the 11/12 time frame. Log entries at the documented outage time on 11/16 were also dissimilar to the training logs of 11/12. However, log entries at the time of the similar pattern, some 20 minutes earlier than the reportage outage time, did show some similarities to the training log set. Taking this as added backing, the filter's findings that the outages on the 16th and 21st were not related does appear to be correct, while the outage on the 16th does appear to be similar, but not at the time reported. The initial assumption that they were all related does appear, in fact, to be mistaken. This conclusion is quite useful, as it could redirect root cause analysis efforts towards more fruitful lines of investigation.

4.1.3 Research Results Summation

Three types of Bayesian content filtering experiments were conducted for this research, utilizing three widely used open source Bayesian spam filters: a spam/ham corpus was manipulated and tested by the filters as a test of the basic functionality of the filters, and as a study of how log entries might be handled differently due to their different lengths and repetitive

natures. Two types of log entries were manipulated and tested by the filters as a trivial test to be sure that these Bayesian spam filters could differentiate log entries and what various manipulations may do to that ability. And, finally, actual production log entries were used to train the filters and logs taken around the times of similar outages were tested for both positive and negative results.

These log sets were quite large, so scatterplots were utilized to show patterns in the scores in a compact and human-accessible way. These scatterplots, seen in Figure 9 through Figure 40, showed patterns for the spring and fall outages which corresponded to the similarities of log entries to the log entries used to train the three filters.

In Figure 9 through Figure 22, the spring outage score graphs were found to to contain patterns which corresponded to the outages reported. Additionally, several unreported outages were detected by the filter and shown as similar vertical point patterns in the logs. These score graph patterns show that these filters can be used for detecting outages, once they are trained for logs from a given type of outage.

In Figure 23 through Figure 40, the fall outage score graphs found that two of the three outages after 11/12 were not similar in log pattern to the first outage (11/18 and 11/21), while the 11/16 outage was similar, but occurred several minutes earlier than reported. These score graphs exhibited dissimilar patterns for two outages following the trained outage, thus demonstrating the utility of these Bayesian spam filters as a mechanism to eliminate possible causes for outages due to the lack of similarity of score patterns when the filters are trained for a particular type of outage.

Using the proposed Filter Effectiveness Scale (FES) scores as a rough guideline for effectiveness with the two data sets, SpamAssassin, SpamBayes and Bogofilter would have all

scored at a 9 out of 10 for the spring outages, while with the fall outages, SpamBayes and Bogofilter would have gone to FES scores of 8 and 8, and SpamAssassin would have dropped to a 7, mostly due to the noise from the need to jitter scores for visibility. Thus the research question of this thesis ("Is it possible to find a widely available, advanced filtering and data clustering technology that is useful for log analysis?") can be answered yes, for each of the three filters tested.

5 SUMMARY AND FUTURE WORK

5.1 Motivation

Computer system outages can be very expensive for organizations and frustrating to IT workers. Log entries written by computer systems are often the main recourse for compter technicians and engineers as they attempt to resolve system problems. These logs are "information ore" for system administrators; this ore must be sifted, filtered and refined, but the end result is potentially of great value, but one of the biggest problems with these logs is the extremely poor richness of the "information ore". The primary purpose of this research was to answer the question, "Is it possible to find a widely available, advanced filtering and data clustering technology that is useful for log analysis?"

5.2 Work Summary

One set of filtering tools that is widely used and freely available today is the spam filter. In particular, spam filters which utilize statistical text analysis, such as those filters using Bayesian content filters, could be used with other text sources, such as log files.

One trains a Bayesian content filter by giving it records which belong to each of several categories, allowing it to build a model representing the likelihood of a given word to belong to a given category. Spam filters commonly distinguish only two categories, called spam and ham. For log filtering, a set of logs related to a type of outage could be used as "spam" training entries,

while a random sampling of non-related entries could be used as "ham" training entries, allowing a filter to build a statistical model for a log set, and allowing it to filter those log entries.

In this research, three spam filters utilizing Bayesian content filters (SpamAssassin, SpamBayes and Bogofilter) were tested for this type of scenario, and their results graphed. The effectiveness of these filters was scored on a Filter Effectiveness Scale (FES) for comparison with each other.

In the first stage of the research, the effectiveness of these filters was tested with the SpamAssassin email corpus. Some minor manipulations were made with these emails to make them more similar to log entries (in particular, message headers and bodies were removed or manipulated). SpamAssassin was generally the most effective, able to properly group the emails (from its own corpus) above 90% of the time; Bogofilter struggled with the corpus, with some manipulations confusing it and pushing its recognition rates below 50% in many cases. SpamBayes took the middle tier, doing its best work with unmanipulated messages (though not as well as SpamAssassin), and its worst with heavily manipulated ones (though not as poorly as Bogofilter).

In the second stage of the research, the log entries from two particular applications were pulled from production systems at the School of Technology, and the effectiveness of the filters in properly grouping these entries was measured. Various manipulations of the log entries were tested and the associated scores were compared. In particular, short word chains (or n-grams) were found to help SpamAssassin and SpamBayes, but long word chains were generally detrimental.

In the third stage of the research, log files from production systems with actual outages were tested with the filters. A set of related outages from the Spring of 2011 were used to train

and test the filters to determine if the filters could find related outages; then a set of possibly related outages from the Fall of 2010 were tested to determine if the filters could show whether or not they were related.

In the Spring set, patterns of record scores related to outage times were easily seen in Figure 9 through Figure 16 (with the necessity of jittering the SpamAssassin scores to see them plainly).

In the Fall set, a possible relation was found between two of the outages, but the other two were determined to be unrelated, as seen in Figure 23 through Figure 34. The greater number of records for the Fall data significantly increased the noise level of output score graphs, making even the patterns found more difficult to see, making the filters show lower FES scores for the Fall data set than the Spring data set.

Today's information systems produce daunting numbers of log entries. Is there utility in using widely available, advanced filtering and data clustering technologies for analyzing these logs? This research shows that today's widely available Bayesian spam filters can be used effectively in filtering and highlighting records of interest amongst tens of thousands of records. Graphs of filter scores from each of the three filters tested showed that logs can be analyzed effectively by Bayesian content filters both for recognizing patterns and for filtering out guessed patterns.

5.3 **Recommendations and Future Work**

This sort of statistical text analysis and filtering of log entries is not widely used, and yet based on these results, it shows a great deal of promise as a method for "refining the ore" in log file analysis. The Bayesian content filters used in this research were, for the most part, open

source and liberally licensed. One of these engines, or something like them, could be included as options for filtering and categorizing logs in one of the many log analysis or log handling tools available freely or commercially. Log analysis can be quite challenging for several reasons, copious data quantities not least amongst these; log analysists need more tools and more powerful tools going forward.

For some sorts of Bayesian content filters, n-gram or word-chaining can be useful for increasing the effectiveness of the filter, but this depends on the characteristics of the given filter. In a few cases in this research, normalizing numbers was somewhat helpful for increasing pattern recognition. Variations on these manipulations could be contrived which would be worth using for some filters.

In this research, only message bodies were used to train filters, but program names, facilities and severities of log entries could be used for future analyses, as these additional pieces of information could give "hints" to an analysis, helping a filter to determine more categorically whether a given entry should be highlighted or filtered, much like the a priori in a Bayesian statistic.

Bayesian content filters, while powerful and efficient, are very simple software devices. Some types of message manipulation were attempted here, with mixed results. These manipulations should be investigated more carefully, particularly after a clearer understanding of the intricacies of a given Bayesian content filter. Other types of text manipulations may be tested, but more powerful statistical tools, such as hidden Markov models, are likely to be more useful in future research.

The basic premise of using a simple Bayesian content filter for filtering log entries has been shown here to be well worth the integration effort required, even without a great deal of

message manipulation to aid these off-the-shelf filters. The ability to train a filter with a small set of problem-related log entries and then sift through massive numbers of additional logs could be a great boon to administrators looking for similar problems occurring at other times. In the debugging of a complex application issue, knowing when similar problems occurred can be quite valuable. Further, issues found during the testing of an application could be used to train a filter that would be used during the production phase of that application, giving operations staff a "heads up" when potential issues arrive. With a certain amount of discipline, systems administrators could use filters like these to proactively watch for previously encountered system problems. Any log analysis package offering these sorts of intelligent filtering, reporting and alerting capabilities is sure to find a ready market as administrators realize the greatly increased power of statistical text analysis over simple text searching and record-type-counting capabilities on the market today,

With the exploding interest in text and data mining in the current knowledge economy, more powerful analysis tools are becoming available each year. As these tools appear and mature, they will eventually either lend themselves to integrations as straight forward as the spam filter integrations used in this research or have such capabilities built-in, offering users greater power of log analysis. As these nascent capabilities mature those integration methods will prove a fruitful line of research: there is value in making more and more powerful tools available, and perhaps even more value in making these sorts of valuable tools easy to utilize. Today's computer technologies provide vast amounts of data to end users. Similarly, the inner workings of those technologies provide vast amounts of run-time data to system administrators. If we are to use that data effectively, we must find ways to utilize the best technologies and best

analysis techniques available. We have little choice but to stand on the shoulders of giants. This will take effort, but the view will be worth it.

REFERENCES

Aharon, M., Barash, G., Cohen, I, Mordechai, E. "One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs." *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases.* 2009. Part I, 243.

Allison, B. "An improved hierarchical Bayesian model of language for document classification." *Proceedings of the 22nd International Conference on Computational Linguistics.* 2008. Volume 1, 25-32.

Andrews, J.H. "Testing usering log file analysis: Tools methods and issues." *Proceedings of the 1th IEEE International Conference on Automated Software Engineering.* 2008. 157.

Apache Foundation. *Apache Chainsaw.* http://logging.apache.org/chainsaw/index.html (accessed April 17, 2010).

—. *SpamAssassin - Welcome to SpamAssassin.* http://spamassassin.apache.org/ (accessed June 25, 2011).

—. "SpamAssassin public corpus readme." *SpamAssassin.* http://spamassassin.apache.org/publiccorpus/ (accessed February 13, 2019).

Axelsson, S. "Intrusion detection systems: A survey and taxonomy." *Depart.of Computer Engineering, Chalmers University, Tech.Rep*, 2000: 99-15.

Bambauer, D., Palfrey, J., Abrams, D. "A Comparative Analysis of Spam Laws: The Quest for Model Law." *ITU WSIS Thematic Meeting on Cybersecurity.* Geneva: International Telecommunication Union, 2005.

Berkhin, P. "A Survey of Clustering Data Mining Techniques." *Grouping Multidimensional Data: Recent Advances in Clustering*, 2996: 25--71.

Brin, S., Motwani, R., Silverstein, C. "Beyond market baskets: Generalizing association rules to correlations." *ACMSIGMOD Record.* 1997. 26(2), 276.

Cavnar, W.B. and Trenkle, J.M. "N-gram-based text categorization." *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval.* 1994. 161--175.

Chang, H.J. and Hung, L.P. and Ho, C.L. "An anticipation model of potential customers' purchasing behavior based on clustering analysis and association rules analysis." *Expert systems with applications*, 2007: 32(3), 753-764.

Chou, A., Yang, J., Chelf, B., Hallem, S., Engler, D. "An empirical study of operationg systems errors." *18th ACM Symposium on Operating Systems Principles (SOSP'01), October 21 2001-October 24.* 2002. 35(5) 73-88.

Dai, W., Xue, G., Yang, Q., Yu, Y. "Transferring naive Bayes classifiers for text classification." *Proceedings of the National Conference on Artificial Intelligence.* 2007. 22(1) 540.

Forte, D. "Log management for effective incident response." *Network Security*, 9 2005: 4-7.

Foundation, Apache. "SpamAssassin: Welcome to SpamAssassin." *SpamAssassin.* http://spamassassin.apache.org (accessed February 13, 2010).

Fox, J. and Weisberg, S. *An {R} Companion to Applied Regression.* Thousand Oaks: Sage, 2011.

Genkin, A., Lewis, D. D., Madigan, D. "Large-scale Bayesian logistic regression for text categorization." *Technometrics*, 2007: 49(3) 291-304.

Gerhards, R. "RFC 5424 - The Syslog Protocol." *IETF*. http://tools.ietf.org/html/rfc5424 (accessed February 13, 2010).

Graham, P. *Hackers & Painters: Big Ideas from the Computer Age.* Beijing: O'Reilly, 2004.

Gunter, D., Tierney, B. L., Brown, A., Swany, M., Bresnahan, J., Schopf, J. M. "Log summarization and anomaly detection for troubleshooting distributed systems." *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing.* 2007. 226-234.

Hellerstein, J. L., Ma, S., Perng, C. S. "Discovering actionable patterns in event data." *IBM Systems Journal*, 2002: 41(3), 475-493.

Hochheiser, H., Schneiderman, B. "Using interactive visualizations of WWW log data to characterize access patterns and informa site design." *Journal of the American Society for Information Science and Technology*, 2001: 52(4), 331-343.

Huang, C., Cohen, I., Symons, J., Abdelzaher, T. "Achieving scalable automated diagnosis of distributed systems performance problems." *Enterprise Systems and Software Laboratory, HP Laboratories Palo Alto. Palo Alto, CA. Rep. HPL-2006-160*, 2007: 1.

Hughs, D. "Using visualization in system and network administration." *Proc. 10th Systems Administration Conference (LISA'96).* 1996. 59--66.

Hulshof, C. D. "Log file analysis." *Encyclopedia of Social Measurement*, 2004: 577--583.

IBM. *IBM - automated problem diagnosis and resolution - Tivoli Enterprise Console - software.* http://www-01.ibm.com/software/tivoli/products/enterprise-console/ (accessed March 13, 2010).

—. *IBM Tivoli Software.* http://www-01.ibm.com/software/tivoli (accessed March 20, 2010).

IETF. *RFC 3195 - Reliable Delivery for Syslog.* https://tools.ietf.org/tools/rfcmarkup/rfcmarkup.cgi?rfc=3195 (accessed April 19, 2010).

—. *RFC 3464 - An Extensible Message Format for Delivery Status Notifications.* http://tools.ietf.org/html/rfc3464 (accessed April 19, 2010).

IETF Syslog Working Group. *IETF Syslog Working Group Home Page.* http://www.employees.org/~lonvick/index.shtml (accessed March 13, 2010).

Journal, Linux. "A Statistical Approach to the Spam Problem." *Linux Journal.* http://www.linuxjournal.com/article/6467 (accessed April 10, 2010).

Juan, A. and Vilar, D. and Ney, H. "Bridging the gap between naive Bayes and maximum entropy text classification." *Pattern Recognition in Information Systems.* 2007. 59--65.

Lewis, D. "Naive (Bayes) at forty: The independence assumption in information retrieval." *Pattern Recognition in Information Systems*, 1998: 4--15.

linux.org. *The Linux Home Page at Linux Online.* http://www.linux.org (accessed April 17, 2010).

Liquidlabs. *Logscape.* http://www.liquidlabs-cloud.com/products/logscape.html (accessed April 17, 2010).

Lonvick, C. *RFC 3164.* http://www.ietf.org/rfc/rfc3164.txt (accessed March 13, 2010).

—. *RFC 3164 notes.* ftp://ftp.rfc-editor.org/in-notes/rfc3164.txt (accessed March 13, 2010).

Lunt, T. F. "Automated audit trail analysis and intrusion detection: A survey." *In Proceedings of the 11th National Computer Security Conference.* 1988.

Ma, H., Hellerstein, J. L. "Mining partially periodic event patterns with unknown periods." *Data Engineering, 2001. Proceedings. 17th International Conference on.* 2001. 205--214.

Makanju, A. A. O., Zincir-Heywood, A. N., Milios, E. E. "Clustering event logs using iterative partitioning." *Proceedings of the 15th ACMSIGKDD International Conference on Knowledge Discovery and Data Mining.* 2009. 1255-1264.

Meyer, T. A., Whateley, B. "Spambayes: Effective open-source, Bayesian based, email classification system." *Proceedings of the First Conference on Email and Anti-Spam (CEAS).* 2004. 98.

Microsoft Corp. *Download details: Log parser 2.2.* http://www.microsoft.com/downloads/en/details.aspx?familyid=890cd06b-abf8-4c25-91b2-f8d975cf8c07&displaylang=en (accessed 4 17, 2010).

Nagios Enterprises, LLC. *Nagios - The Industry Standard in IT Infrastructure Monitoring.* http://www.nagios.org/ (accessed March 13, 2010).

Nawyn, K. E. "A Security Analysis of System Event Logging with Syslog." *SANS Institute, no. As part of the Information Security Reading Room*, 2003.

New, D., Rose, M. *RFC 3195.* ftp://ftp.rfc-editor.org/in-notes/rfc3195.txt (accessed March 13, 2010).

Nicholas, D., Huntington, P., Lievesley, N., Withey, R. "Cracking the code: web log analysis." *Online Information Review,* 1999: 23(5) 263--269.

Novell, Inc. *Log Management software | Sentinel Log Manager.* http://www.novell.com/products/sentinel-log-manager/ (accessed April 17, 2010).

Palfrey, J. and Abrams, D. and Bambauer, D. "A comparative analysis of spam laws: The quest for a model law." *Buscalegis,* 2005.

Pike, R., Dorward, S., Griesemer, R., Quinlan, S. "Interpreting the data: Parallel Analysis with Sawzall." *Scientific Programming,* 2005: 13(4), 277--298.

Quest Software, Inc. "Network monitoring software tools with Big Brother by Quest Software." *Quest Software.* http://www.quest.com/bigbrother (accessed March 13, 2010).

Quest Software, LLC. *Quest Software - Smart Systems Management.* http://www.quest.com (accessed March 20, 2010).

R Development Core Team. *R: A Language and Environment for Statistical Computing.* http://www.R-project.org (accessed November 20, 2010).

Ragan, S. "Spam levels now at 90 percent says Symantec - junk mail arriving like clockwork - Security." *The Tech Herald.* http://www.thetechherald.com/article.php/200922/3756/Spam-levels-now-at-90-percent-says-Symantec-junk-mail-arriving-like-clockwork (accessed March 13, 2010).

Raymond, E. S. *Bogofilter home page.* http://bogofilter.sourceforge.net (accessed 4 10, 2010).

Rigoutsos, I., Floratos, A. "Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm." *Bioinformatics-Oxford,* 1998: 14(1), 55--67.

Rish, I. "An Empirical Study of the Naive Bayes Classifier." *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence,* 2001: 41-46.

Robinson, G. *Old Spam Detection.* http://radio-weblogs.com/0101454/stories/2002/09/24/oldSpamDetection.html (accessed April 10, 2010).

Sahami, M., Dumais, S., Heckerman, D., Horvitz, E. "A Bayesian Approach to Filtering Junk E-Mail." *Learning for Text Categorization: Papers from the 1998 Workshop,* 1998: 62 98-05.

Seewald, A. K. "An Evaluation of Naive Bayes Variants in Content-based Learning for Spam Filtering." *Intelligent Data Analysis,* 2007: 11(5), 497--524.

SpamBayes. "SpamBayes: Bayesian anti-spam classifier written in Python." *SpamBayes.* http://spambayes.sourceforge.net/ (accessed February 13, 2010).

Splunk Inc. "Splunk | IT Search for Log Management, Operations Security and Compliance." *Splunk.* http://www.splunk.com/ (accessed March 13, 2010).

Stearly, J. "Towards Informatic Analysis of Syslogs." *Proceedings of IEEE International Conference on Cluster Computing.* 2004.

Swatch. *Simple log watcher | get simple log watcher at SourceForge.net.* http://sourceforge.net/projects/swatch (accessed March 13, 2010).

Takada, T., Koike, H. "Tudumi: Information Visualization System for Monitoring and Auditiong Computer Logs." *Proceedings of the 6th International Conference on Information Visualization.* 2002.

Thebert, S. *Octopussy [home]*. http://www.8pussy.org/dokuwiki/doku.php (accessed April 17, 2010).

Thompson, K. *LogSurfer & LogSurfer+ = real time log monitoring and alerting.* http://www.crypt.gen.nz/logsurfer (accessed March 13, 2011).

Torvalds, L. *The Linux Kernel Archives.* http://kernel.org (accessed June 25, 2011).

Vaarandi, R. "A Data Clustering Algorithm for Mining Patterns from Event Logs." *Proceedings of the 2003 IEEE Workshop on IP Operations and Management.* 2003. 119-126.

—. "Sec - A Lightweight Event Correlation Tool." *2002 IEEE Workshop on IP Operations and Management.* 2002. 111-115.

XpoLog Ltd. *XpoLog log management and log analysis platform.* http://xpolog.com (accessed April 17, 2010).

Xu, W. and Huang, L. and Fox, A. and Patterson, D. and Jordan, M. "Mining console logs for large-scale system problem detection." *Proceedings of the Third conference on Tackling computer systems problems with machine learning techniques.* USENIX Association, 2008. 4--3.

Zdziarski, J. A. *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification.* San Francisco, CA, USA: No Starch Press, 2005.

Zenoss Inc. *Zenoss Open Source Server and Network Monitoring - Core and Enterprise.* http://www.zenoss.com/ (accessed March 13, 2010).

APPENDIX. PROGRAM CODE AND TEMPLATES

Spam Testing Programs and Scripts

shtrim.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
# shtrim.py - spam/ham header trimmer/splitter
#       - Divides the header from the body for spam and ham mailbox entries, then creates
transformed mailbox entries of the following types:
#               1. Headers preserved, generic body
#               2. Body preserved, generic header
#               3. Messages created for each sentence in body; generic header
#
#       Current version: 0.7
#
#       Version info:
#               0.7 - 5/1/10 - rwh - Initial working version
#

import sys
import os.path as ospath
import os
import getopt
import glob

class shtrim:
        _DEFAULT_TARGET_DIR_TAIL = '.'+os.sep+'messages.trimmed'
        _DEFAULT_HEADER_TARGET_DIR_TAIL = '.hpreserved'
        _DEFAULT_BODY_TARGET_DIR_TAIL = '.bpreserved'
        _DEFAULT_BODY_SPLIT_TARGET_DIR_TAIL = '.bsplit'
        _DEFAULT_HEADER_TEMPLATE_FILE = 'shtrim.header.template.txt'
        _DEFAULT_BODY_TEMPLATE_FILE = 'shtrim.body.template.txt'
        _DEFAULT_BODY_SPLIT_TEMPLATE_FILE = _DEFAULT_BODY_TEMPLATE_FILE
        _verbose = False
        _DELETE_MODE = True#False
        _TEXT_TEMPLATE_HEADER='__INSERT_BODY_HERE__'
        _TEXT_TEMPLATE_BODY='__INSERT_BODY_HERE__'

        def usage(self):
                '''Show the usage of the program'''
                print '''Usage: shtrim.py -i <inputdir> [options]
                -i, --inputfilespec <filespec>   Source file glob (required)
                -o, --outputdir <basedirname>    Base   name   for   output   dirs   (default   =
'''+self._DEFAULT_TARGET_DIR_TAIL+''')
                -t, --headertemplatefile <filename>      Tempate    for    header    (default    =
'''+self._DEFAULT_HEADER_TEMPLATE_FILE+'''
```

```
                        -b, --bodytemplatefile <filename>        Tempate     for     body     (default     =
'''+self._DEFAULT_BODY_TEMPLATE_FILE+'''
                        -s, --splittemplatefile <filename>       Tempate   for   split-body   (default   =
'''+self._DEFAULT_BODY_SPLIT_TEMPLATE_FILE+'''
                        -v, --verbose    Turn on verbose logging
                        -h, --help       This help

        Specified input files will be transformed into messages of 3 types (each with its own directory):
        1. Headers preserved, generic body
        2. Generic header; body preserved
        3. Generic header; messages created for each sentence in body
        '''


                def get_params(self):
                        '''Parse out the command-line parameters'''
                        try:
                                opts,      args      =      getopt.getopt(sys.argv[1:],      "i:o:t:b:s:hv",
["inputfilespec=","outputdir","headertemplatefile=","bodytemplatefile=","splittemplatefile=","help","verbo
se"])
                        except getopt.GetoptError, err:
                                # print help information and exit:
                                print str(err) # will print something like "option -a not recognized"
                                self.usage()
                                sys.exit(1)

                        infilelist = None
                        outdirbasename = None
                        header_template_file = self._DEFAULT_HEADER_TEMPLATE_FILE
                        body_template_file = self._DEFAULT_BODY_TEMPLATE_FILE
                        split_template_file = self._DEFAULT_BODY_SPLIT_TEMPLATE_FILE

                        for opt, arg in opts:
                                if opt in ('-i','--inputfilespec'):
                                        inspec = arg
                                        if inspec.startswith('~'):
                                                inspec = os.path.expanduser(inspec)
                                                #print '-',inspec
                                        infilelist = glob.glob(inspec)
                                        #print len(infilelist)
                                elif opt in ('-o','--outputdir'):
                                        outdirbasename = arg
                                        #print arg

                                elif opt in ('-t', '--headertemplatefile'):
                                        header_template_file = arg
                                        if not ospath.exists(header_template_file):
                                                print 'The specified template file does not exist.'
                                                self.usage()
                                                sys.exit(1)

                                elif opt in ('-b', '--bodytemplatefile'):
                                        body_template_file = arg
                                        if not ospath.exists(body_template_file):
                                                print 'The specified template file does not exist.'
                                                self.usage()
                                                sys.exit(1)

                                elif opt in ('-s', '--splittemplatefile'):
                                        split_template_file = arg
                                        if not ospath.exists(split_template_file):
                                                print 'The specified template file does not exist.'
                                                self.usage()
                                                sys.exit(1)

                                elif opt in ('-v','--verbose'):
                                        self._verbose = True
                                elif opt in ('-h', '--help'):
                                        self.usage()
```

```
                                        sys.exit()
                                else:
                                        assert False, "Unsupported option specified"
                                # ...

                        if infilelist is None or len(infilelist) == 0:
                                print 'Please specify at least one filename. '#,infilelist
                                self.usage()
                                sys.exit(1)
                        if outdirbasename is None or outdirbasename == '':
                                outdirbasename = self._DEFAULT_TARGET_DIR_TAIL
                                if self._verbose:
                                        print 'Defaulting to output dir name: ',outdirbasename

                        header_template = self.read_file_to_string(split_template_file)
                        body_template = self.read_file_to_string(split_template_file)
                        split_template = self.read_file_to_string(split_template_file)

                        if header_template == None:
                                print       'Could       not       read       required       header_template
file:',header_template_file
                                sys.exit(2)
                        if body_template == None:
                                print 'Could not read required body_template file:',body_template_file
                                sys.exit(2)
                        if split_template == None:
                                print 'Could not read required split_template file:',split_template_file
                                sys.exit(2)

                        return infilelist, outdirbasename, header_template, body_template, split_template


                def fix_target_dir(self,outdir):
                        if ospath.isdir(outdir):
                                outfiles = os.listdir(outdir)
                                if len(outfiles) > 0:
                                        print     'Output     dir     ('+outdir+')     is     not     empty.
',len(outfiles),'file(s)/dir(s) found in directory.'
                                        resp = raw_input('Overwrite '+str(outdir)+'? (y/N) ').lower()
                                        if resp != 'y':
                                                print 'Leaving output directory intact.'
                                                sys.exit(5)
                                        else:
                                                print 'Deleting files from target dir...'
                                                for thisfile in outfiles:
                                                        this_full_file = outdir+os.sep+thisfile
                                                        if ospath.isfile(this_full_file):
                                                                if self._DELETE_MODE:
                                                                        if self._verbose:
                                                                                print
'Deleting',this_full_file

                                                                        os.remove(this_full_file)
                                                                else:
                                                                        if self._verbose:
                                                                                print      'Would      be
deleting',this_full_file,', but delete mode is turned off during testing.'
                                                        else:
                                                                print  thisfile,'is  not  a  regular  file.
Bypassing...'
                                else:
                                        print 'Output dir ('+outdir+') exists but is empty.'
                        else:
                                print 'Creating output dir,',outdir+'.'
                                os.makedirs(outdir)


                def write_msg(self, fullname, new_msg, out_dir):
                        fname = ospath.basename(fullname)
```

```
                f_out = open(out_dir+os.sep+fname,'w')
                try:
                        f_out.write(new_msg)
                finally:
                        f_out.close()


        def split_message(self, message):
                _SPLIT_STRING = '\n\n'
                if _SPLIT_STRING in message:
                        smessage = message.split(_SPLIT_STRING)
                        if len(smessage) > 1000:
                                print 'More than 1000 sentences in message.  Skipping message.'
                                return
                        #header= message.split(_SPLIT_STRING)[0]
                        header = smessage[0]
                        if len(smessage) == 2:
                                body = smessage[1]
                        else:
                                body = _SPLIT_STRING.join(smessage[1:])
                else:
                        print 'Message invalid: cannot be split into header and body.'
                        print 'Message is',len(message),'characters long'
                        print 'Message[0:100]--->',message[0:100],'<---'

                        header = None
                        body = None
                return header, body


        def apply_template(self, text_in, template, replacement_string):
                #print template
                #print msg_in

                return template.replace(replacement_string, text_in)


        def read_file_to_string(self, this_file, exclude_comment_lines=True):
                if not ospath.exists(this_file):
                        return None
                f_in = open(this_file)
                contents = None
                try:
                        templatelines = f_in.readlines()
                        if exclude_comment_lines:
                                templatelines = [line for line in templatelines if not
line.startswith('#')]
                        contents = ''.join(templatelines)
                        #print contents

                finally:
                        f_in.close()
                return contents

        def split_message_lines(self, message):
                lines = []
                if '.' in message:
                        lines = message.split('.')
                        last_entry = lines[-1]
                        lines = [x+'.' for x in lines[:-1] if len(x.strip()) > 0]
                        lines.append(last_entry)
                return lines

        def get_split_body_messages(self, body, template, replacement_string):
                '''Returns a list of messages the body if each being one sentence from the body
of the original message'''
                messages = []
                split_message = self.split_message_lines(body)
                for body in split_message:
```

```
                                        messages.append(self.apply_template(body,                template,
replacement_string))
                        return messages


                def processfiles(self, filelist, outdirbase, h_template, b_template, s_template):
                        print 'Processing files...'
                        outdir_header = outdirbase+self._DEFAULT_HEADER_TARGET_DIR_TAIL
                        self.fix_target_dir(outdir_header)
                        outdir_body = outdirbase+self._DEFAULT_BODY_TARGET_DIR_TAIL
                        self.fix_target_dir(outdir_body)
                        outdir_split = outdirbase+self._DEFAULT_BODY_SPLIT_TARGET_DIR_TAIL
                        self.fix_target_dir(outdir_split)

                        bad_records = 0
                        for each_file in filelist:
                                old_msg = self.read_file_to_string(each_file)
                                if old_msg == None:
                                        print 'Could not read old message from',each_file
                                        continue
                                header, body = self.split_message(old_msg)
                                if header == None or body == None:
                                        print 'Message not parsed:',each_file
                                        bad_records += 1
                                        continue
                                header_message       =       self.apply_template(header,       h_template,
self._TEXT_TEMPLATE_HEADER) #self.get_header_message(header, template)
                                body_message        =        self.apply_template(body,        b_template,
self._TEXT_TEMPLATE_BODY) #self.get_body_message(body, template)
                                split_messages     =     self.get_split_body_messages(body,     s_template,
self._TEXT_TEMPLATE_BODY)
                                self.write_msg(each_file, header_message, outdir_header)
                                self.write_msg(each_file, body_message, outdir_body)
                                count = 0;
                                for split_message in split_messages:
                                        count += 1
                                        self.write_msg(each_file+str(count), split_message, outdir_split)
                                #new_msg = self.applytemplate(each, template)
                                #self.write_msg(each, new_msg, outdir)
                        print '...done processing files. ',bad_records,'bad records were found.'


                def start(self):
                        infilelist,  outdirbasename,  header_template,  body_template,  split_template  =
self.get_params()
                        self.processfiles(infilelist,  outdirbasename,  header_template,  body_template,
split_template)


        if __name__ == '__main__':
                sht = shtrim()
                try:
                        sht.start()
                except KeyboardInterrupt:
                        print 'Interrupted by user...'
```

shtrim.body.template.txt

```
#shtrim.py header template file -- commented lines are removed from the template by default
__INSERT_HEADER_HERE__


Generic message body text.
```

shtrim.header.template.txt

```
#shtrim.py body template file -- commented lines are removed from the template by default
Return-Path: skip@pobox.com
Delivery-Date: Sat May  1 20:47:01 2010
From: spamtest.rhavens@byu.com (Russel Havens)
Date: Sat, 1 May 2010 19:47:01 -0600
Subject: Test Message

__INSERT_BODY_HERE__
```

stestgen.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
#       stestgen.py - spam test generator
#               - Generates lists of message files as well as shell scripts to run tests against
SpamAssassin
#               - Selects a random subset of spam and ham messages for training and testing,
creating files of names like "[train|test]filelist[1-5]-5percent.txt".
#                       [By default, the "train" files are the sample size and the "test" files
are (totalsize-samplesize)]
#

import os.path as ospath
import os
import sys
import glob
import random
import ConfigParser

_DEFAULT_CONFIG_FILE = 'stestgen.properties'
_DEFAULT_SPAM_DIRS = '/home/rhavens/spam/sa/*spam*'
_DEFAULT_HAM_DIRS = '/home/rhavens/spam/sa/*ham*'
_DEFAULT_TRAIN_SCRIPT_NAME = 'train_sa.sh'
_DEFAULT_TEST_SCRIPT_NAME = 'test_sa.sh'

_DNAME_BASE='filelist.txt'
_HAM = 'ham'
_SPAM = 'spam'
_TRAIN = 'train'
_TEST = 'test'
_UND = '_'
_DEFAULT_INCLUDE_TRAINING_DATA_FOR_TEST=False
_DEFAULT_TRAINING_PERCENT = 10.0
_BALANCE_SET_SIZES = False
_DEBUG_MODE=False
_CONFIG = 'config'
_OS = sys.platform

def read_properties(configfile):
        defaults = {'spam_dirs':_DEFAULT_SPAM_DIRS, 'ham_dirs':_DEFAULT_HAM_DIRS,
                        'train_script_name':_DEFAULT_TRAIN_SCRIPT_NAME,
'test_script_name':_DEFAULT_TEST_SCRIPT_NAME,
                        'name_base':_DNAME_BASE,
'include_training_data_for_test':_DEFAULT_INCLUDE_TRAINING_DATA_FOR_TEST,
                        'training_percent':10.0,
'balance_set_sizes':_DEFAULT_TRAINING_PERCENT,'debug_mode':_DEBUG_MODE}
        config = ConfigParser.ConfigParser(defaults)
        #config.readfp(configfile)
        config.read(configfile)
        return config
```

```
	def get_dir_lists(spam_glob, ham_glob):
		spam_dirs = glob.glob(spam_glob)
		spam_dirs = [x for x in spam_dirs if ospath.isdir(x)]
		ham_dirs = glob.glob(ham_glob)
		ham_dirs = [x for x in ham_dirs if ospath.isdir(x)]
		return spam_dirs, ham_dirs

	def get_one_dir_listing(dirname):
		full_path_listing = [dirname+os.sep+x for x in os.listdir(dirname)]
		return full_path_listing

	def get_file_lists(spam_dirs, ham_dirs):
		spam_files = []
		ham_files = []

		for each in spam_dirs:
			spam_files.extend(get_one_dir_listing(each))
		for each in ham_dirs:
			ham_files.extend(get_one_dir_listing(each))
		return spam_files, ham_files

	def separate_train_test(filelist, sample_percent, incl_train_in_test, src_type):
		if sample_percent >= 100 or sample_percent <=0:
			print 'Sample size must be between 0 and 100'
			sys.exit(1)
		sample_size = int((sample_percent/100.0)*len(filelist))

		print  src_type,'file  list  size='+str(len(filelist))+';  '+str(sample_percent)+'%  sample;
Sample size='+str(sample_size)

		#print len(filelist),'-',abs_sample_size
		listlen = len(filelist)
		randlines = []
		while len(randlines) < sample_size:
			nextval = random.randint(1,listlen)
			if nextval not in randlines:
				randlines.append(nextval)
		randlines.sort()
		#print randlines#[0:15]


		train = []
		test = []
		loopnum = 0
		for item in filelist:
			if len(randlines) == 0:
				break
			loopnum += 1
			if loopnum == randlines[0]:
				train.append(item)
				randlines.pop(0)
			else:
				if not incl_train_in_test:
					test.append(item)
		if incl_train_in_test:
			test = filelist
		#print len(train)
		#print len(test)
		return train, test

	def write_data(filename, data):

		if _DEBUG_MODE:
			print 'Write',len(data),'lines to',filename
			return

		f_out = open(filename,'w')
```

```
                try:
                        for each in data:
                                f_out.write(each)
                                f_out.write('\n')
                finally:
                        f_out.close()


        def write_train_script(script_name, spam_data_name, ham_data_name):
                train_script = '''#!/bin/sh

        #Clearing training tables
        sa-learn --clear

        echo 'Spam training...'
        for each in `cat '''+spam_data_name+'''`;
        do
                sa-learn --spam $each
        done

        echo 'Ham training...'
        for each in `cat '''+ham_data_name+'''`;
        do
                sa-learn --ham $each
        done

        echo 'Done training'
        '''
                write_data(script_name, train_script.split('\n'))
                print 'Training script written as',script_name

        def write_test_script(script_name, spam_data_name, ham_data_name):
                test_script = '''#!/bin/sh

        echo 'expected_type      score   threshold'
        for each in `cat '''+spam_data_name+'''`;
        do
                retval=`spamc -c < $each`
                echo "spam      ${retval/\//    }"
        done

        echo 'expected_type      score   threshold'
        for each in `cat '''+ham_data_name+'''`;
        do
                retval=`spamc -c < $each`
                echo "ham       ${retval/\//    }"
        done

        echo 'Done testing'
        '''
                write_data(script_name, test_script.split('\n'))
                print 'Testing script written as',script_name

        def  write_files(ham_train,  ham_test,  spam_train,  spam_test,  tr_script_name,  te_script_name,
tr_h_data_name, te_h_data_name, tr_s_data_name, te_s_data_name):
                write_train_script(tr_script_name, tr_s_data_name, tr_h_data_name)
                write_test_script(te_script_name, te_s_data_name, te_h_data_name)
                write_data(tr_h_data_name, ham_train)
                write_data(te_h_data_name, ham_test)
                write_data(tr_s_data_name, spam_train)
                write_data(te_s_data_name, spam_test)

        def rebalance_sets(s_tr, s_te, h_tr, h_te):
                if len(s_tr) == len(h_tr):
                        return
                elif len(s_tr) > len(h_tr):
                        while len(s_tr) > len(h_tr):
                                rand_entry = random.randint(0,len(s_tr)-1)
                                s_te.append(s_tr.pop(rand_entry))
```

```
                else:
                        while len(h_tr) > len(s_tr):
                                rand_entry = random.randint(0,len(h_tr)-1)
                                h_te.append(h_tr.pop(rand_entry))
                #Sanity check
                if len(s_tr) != len(h_tr):
                        print 'Unequal set lengths after rebalance: s_tr=',len(s_tr),';',len(h_tr)
                print 'Sets rebalanced to',len(s_tr),'entries each'

        def start():
                config_file = _DEFAULT_CONFIG_FILE
                if len(sys.argv) > 1:
                        if ospath.exists(sys.argv[1]):
                                print 'Reading specified configuration file'
                                config_file = sys.arg[1]
                props = read_properties(config_file)
                #print props.defaults()
                #print props.items('config')


                spam_dirs,        ham_dirs        =        get_dir_lists(props.get('config','spam_dirs'),
props.get('config','ham_dirs'))
                #print 'SPAM_DIRS=',spam_dirs
                spam_files, ham_files = get_file_lists(spam_dirs, ham_dirs)
                #print 'SPAM_FILES=',spam_files
                spam_train,             spam_test             =             separate_train_test(spam_files,
props.getfloat('config','training_percent'),
props.getboolean('config','include_training_data_for_test'),'Spam')
                ham_train,              ham_test              =              separate_train_test(ham_files,
props.getfloat('config','training_percent'),
props.getboolean('config','include_training_data_for_test'),'Ham')
                if props.getboolean('config','balance_set_sizes'):
                        rebalance_sets(spam_train, spam_test, ham_train, ham_test)
                else:
                        print 'Set sizes: Spam=',len(spam_train),'; Ham=',len(ham_train)

                #print len(spam_train),',',len(spam_test),'.',len(ham_train),',',len(ham_test)

                tr_h_name = _TRAIN+_UND+_HAM+_UND+props.get('config','name_base')
                tr_s_name = _TRAIN+_UND+_SPAM+_UND+props.get('config','name_base')
                te_h_name = _TEST+_UND+_HAM+_UND+props.get('config','name_base')
                te_s_name = _TEST+_UND+_SPAM+_UND+props.get('config','name_base')

                write_files(ham_train,             ham_test,             spam_train,             spam_test,
props.get('config','train_script_name'),   props.get('config','test_script_name'),   tr_h_name,   te_h_name,
tr_s_name, te_s_name)


        if __name__ == '__main__':
                start()

        stestgen.properties
        #stestgen.properties
        [config]
        #Name of training script to write
        #---default: train_sa.sh
        train_script_name=train_sa5-10percent.sh

        #Name of test script to write
        #---default: test_sa.sh
        test_script_name=test_sa5-10percent.sh

        #Base name used for data output files (i.e. data files with lists of spam and ham files used for
the scripts)
        #---default: filelist.txt
        name_base=filelist5-10percent.txt

        #Glob of directories which hold spam mail messages
        #---default: /home/rhavens/spam/sa/*spam*
```

```
        #Linux
        spam_dirs=/home/rhavens/spam/sa/*spam*
        #Windows
        #spam_dirs=/spam/*spam*

        #Glob of directories which hold ham mail messages
        #---default: /home/rhavens/spam/sa/*ham*
        #Linux
        ham_dirs=/home/rhavens/spam/sa/*ham*
        #Windows
        #ham_dirs=/spam/*ham*

        #Include the training data in both the training and testing sets
        #-If this is True, the training records will be left in the testing data set.  This will slightly
reduced memory utilization
        #---default: False
        include_training_data_for_test=False

        #Sample size (in percent) for training data set
        #---default: 10
        training_percent=10

        #Ensure that spam and ham training data sets are the same size
        #--reduces the size of the larger of the two to the size of the smaller
        #---default: False
        balance_set_sizes=False

        #If debug_mode=True, do not write output files
        #---default: False
        debug_mode=False
```

gen-altered-list.py

```
        #!/usr/bin/python
        #
        #
        import sys
        import os
        import os.path as ospath
        import glob

        _TXT='.txt'
        basedir='/home/rhavens/spam/sa/'
        modified_files_dir=basedir+'modified_files/'#'/home/rhavens/spam/sa/modified_files/'
        dirset='''easy_ham
        easy_ham_2
        hard_ham
        spam
        spam_2'''.split()

        #modified_files='/modified_files/'
        extensions = {'.bpreserved':'-b','.hpreserved':'-h','.bsplit':'-bs'}


        listfile_dir = '/home/rhavens/Dropbox/code/python/source/'#previously, I added '/runs/thirdrun/'

        #for each in os.listdir(listfile_dir):
        for listfile in glob.glob(listfile_dir+"*10percent.txt"):#use "*percent.txt" to catch ALL text
files
                #if not each.endswith('.sh'):
                #       print each
                for ext in extensions.keys():
                        #print listfile,'--',listfile.replace(_TXT,extensions[ext]+_TXT)
                        #continue
                        f_out = open(listfile.replace(_TXT,extensions[ext]+_TXT),'w')
                        try:
```

```
                        f_in = open(listfile)
                        try:
                                for line in f_in:
                                        newline = line.replace(basedir,modified_files_dir)

f_out.write(newline[:newline.rfind('/')]+ext+'/'+newline[newline.rfind('/')+1:])
                        finally:
                                f_in.close()
                finally:
                        f_out.close()
```

train_sabs1.sh

```
#!/bin/sh
#TRAIN_LIST_LOC="/home/rhavens/spam/sa/temp/"
TRAIN_LIST_LOC="/home/rhavens/Dropbox/code/python/source/runs/thirdrun"
#SAMPLESIZE=""
SAMPLESIZE="-5percent"

export SA='spamassassin'
export BOGO='bogofilter'
export SB='spambayes'

export TOOL=$SA
#export TOOL=$BOGO
#export TOOL=$SB
echo "Training with ${TOOL}"

RUN=1
if [ "x$1" != "x" ];
then
        if [ $1 -ge 1 -a $1 -le 5 ];
        then
                RUN=$1
        fi
fi

OUTFILE_TRAIN=train_spam_bs.run.${TOOL}.${RUN}.out
OUTFILE_TEST=test_spam_bs.test.${TOOL}.${RUN}.csv

echo "Test: bodysplit - $RUN - $TOOL" > $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN

#Clearing training tables and output file
if [ $TOOL == $SA ];
then
        sa-learn --clear >> $OUTFILE_TRAIN
else
        if [ $TOOL == $BOGO ];
        then
                #bogofilter --db-remove-environment > $OUTFILE_TRAIN
                rm ~/.bogofilter/wordlist.db
                echo "Removed ~/.bogofilter/wordlist.db" >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $SB ];
                then
                        sb_filter.py -n >> $OUTFILE_TRAIN
                else
                        echo "Invalid tool specified: $TOOL"
                        exit 2
                fi
        fi
fi

echo 'Spam training BODY_SPLIT...'
for each in `cat ${TRAIN_LIST_LOC}train_spam_bs_filelist${RUN}.txt`;
```

```
do
        for f in `ls ${each}*`;
        do
                if [ $TOOL == $SA ];
                then
                        sa-learn --spam $f >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $BOGO ];
                        then
                                bogofilter -s < $f >> $OUTFILE_TRAIN
                        else
                                if [ $TOOL == $SB ];
                                then
                                        sb_filter.py -s < $f > /dev/null
                                fi
                        fi
                fi
        done
done

echo 'Ham training...'
for each in `cat ${TRAIN_LIST_LOC}train_ham_bs_filelist${RUN}.txt`;
do
        for f in `ls ${each}*`;
        do
                if [ $TOOL == $SA ];
                then
                        sa-learn --ham $f >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $BOGO ];
                        then
                                bogofilter -n < $f >> $OUTFILE_TRAIN
                        else
                                if [ $TOOL == $SB ];
                                then
                                        sb_filter.py -s < $f > /dev/null
                                fi
                        fi
                fi
        done
done

echo 'Done training'
echo 'Done training' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN

echo "Running test for SPLIT_BODY sample ${RUN}"
./test_sa1.sh ${RUN} > ${OUTFILE_TEST}

echo 'Done testing' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN
```

train_sah1.sh

```
#!/bin/sh
TRAIN_LIST_LOC="/home/rhavens/spam/sa/temp/"

export SA='spamassassin'
export BOGO='bogofilter'
export SB='spambayes'

#export TOOL=$SA
#export TOOL=$BOGO
export TOOL=$SB
echo "Training with ${TOOL}"
```

```
RUN=1
if [ "x$1" != "x" ];
then
        if [ $1 -ge 1 -a $1 -le 5 ];
        then
                RUN=$1
        fi
fi

OUTFILE_TRAIN=train_spam_h.run.${TOOL}.${RUN}.out
OUTFILE_TEST=test_spam_h.test.${TOOL}.${RUN}.csv

echo "Test: header - $RUN - $TOOL" > $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN

#Clearing training tables and output file
if [ $TOOL == $SA ];
then
        sa-learn --clear >> $OUTFILE_TRAIN
else
        if [ $TOOL == $BOGO ];
        then
                #bogofilter --db-remove-environment > $OUTFILE_TRAIN
                rm ~/.bogofilter/wordlist.db
                echo "Removed ~/.bogofilter/wordlist.db" >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $SB ];
                then
                        sb_filter.py -n >> $OUTFILE_TRAIN
                else
                        echo "Invalid tool specified: $TOOL"
                        exit 2
                fi
        fi
fi


echo 'Spam train HEADER_ONLY...'
for each in `cat ${TRAIN_LIST_LOC}train_spam_h_filelist${RUN}.txt`;
do
        if [ $TOOL == $SA ];
        then
                sa-learn --spam $each >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $BOGO ];
                then
                        bogofilter -s < $each >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $SB ];
                        then
                                sb_filter.py -s < $each > /dev/null
                        fi
                fi
        fi
done

echo 'Ham training...'
for each in `cat ${TRAIN_LIST_LOC}train_ham_h_filelist${RUN}.txt`;
do
        if [ $TOOL == $SA ];
        then
                sa-learn --ham $each >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $BOGO ];
                then
                        bogofilter -n < $each >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $SB ];
                        then
```

```
                                sb_filter.py -g < $each > /dev/null
                        fi
                fi
        fi
done

echo 'Done training'
echo 'Done training' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN

echo "Running test for HEADER_ONLY sample ${RUN}"
./test_sa1.sh ${RUN} > $OUTFILE_TEST

echo 'Done testing' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN
```

train_sab1.sh

```
#!/bin/sh
#TRAIN_LIST_LOC="/home/rhavens/spam/sa/temp/"
TRAIN_LIST_LOC="/home/rhavens/Dropbox/code/python/source/runs/thirdrun"
#SAMPLESIZE=""
SAMPLESIZE="-5percent"

export SA='spamassassin'
export BOGO='bogofilter'
export SB='spambayes'

export TOOL=$SA
#export TOOL=$BOGO
#export TOOL=$SB
echo "Training with ${TOOL}"

RUN=1
if [ "x$1" != "x" ];
then
        if [ $1 -ge 1 -a $1 -le 5 ];
        then
                RUN=$1
        fi
fi

OUTFILE_TRAIN=train_spam_b.run.${TOOL}.${RUN}.out
OUTFILE_TEST=test_spam_b.test.${TOOL}.${RUN}.csv

echo "Test: body - $RUN - $TOOL" > $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN

#Clearing training tables and output file
if [ $TOOL == $SA ];
then
        sa-learn --clear >> $OUTFILE_TRAIN
else
        if [ $TOOL == $BOGO ];
        then
                #bogofilter --db-remove-environment > $OUTFILE_TRAIN
                rm ~/.bogofilter/wordlist.db
                echo "Removed ~/.bogofilter/wordlist.db" >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $SB ];
                then
                        sb_filter.py -n >> $OUTFILE_TRAIN
                else
                        echo "Invalid tool specified: $TOOL"
                        exit 2
                fi
```

```
        fi
fi

echo 'Spam training BODY_ONLY...'
for each in `cat ${TRAIN_LIST_LOC}train_spam_b_filelist${RUN}.txt`;
do
        if [ $TOOL == $SA ];
        then
                sa-learn --spam $each >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $BOGO ];
                then
                        bogofilter -s < $each >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $SB ];
                        then
                                sb_filter.py -s < $each > /dev/null
                        fi
                fi
        fi
done

echo 'Ham training...'
for each in `cat ${TRAIN_LIST_LOC}train_ham_b_filelist${RUN}.txt`;
do
        if [ $TOOL == $SA ];
        then
                sa-learn --ham $each >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $BOGO ];
                then
                        bogofilter -n < $each >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $SB ];
                        then
                                sb_filter.py -g < $each > /dev/null
                        fi
                fi
        fi
done

echo 'Done training'
echo 'Done training' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN

echo "Running test for BODY_ONLY sample ${RUN}"
./test_sa1.sh ${RUN}> $OUTFILE_TEST

echo 'Done testing' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN
```

train_safull1.sh

```
#!/bin/sh
FULL_TRAIN_LIST_LOC="/home/rhavens/Dropbox/code/python/source/runs/firstrun/"

export SA='spamassassin'
export BOGO='bogofilter'
export SB='spambayes'

export TOOL=$SA
#export TOOL=$BOGO
#export TOOL=$SB
echo "Training with ${TOOL}"

RUN=1
```

```
if [ "x$1" != "x" ];
then
        if [ $1 -ge 1 -a $1 -le 5 ];
        then
                RUN=$1
        fi
fi

OUTFILE_TRAIN=train_spam_full.run.${TOOL}.${RUN}.out
OUTFILE_TEST=test_spam_full.test.${TOOL}.${RUN}.csv

echo "Test: full - $RUN - $TOOL" > $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN

#Clearing training tables and output file
if [ $TOOL == $SA ];
then
        sa-learn --clear >> $OUTFILE_TRAIN
else
        if [ $TOOL == $BOGO ];
        then
                #bogofilter --db-remove-environment > $OUTFILE_TRAIN
                rm ~/.bogofilter/wordlist.db
                echo "Removed ~/.bogofilter/wordlist.db" >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $SB ];
                then
                        sb_filter.py -n >> $OUTFILE_TRAIN
                else
                        echo "Invalid tool specified: $TOOL"
                        exit 2
                fi
        fi
fi

echo 'Spam training FULL MESSAGES...'
for each in `cat ${FULL_TRAIN_LIST_LOC}train_spam_filelist${RUN}.txt`;
do
        if [ $TOOL == $SA ];
        then
                sa-learn --spam $each >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $BOGO ];
                then
                        bogofilter -s < $each >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $SB ];
                        then
                                sb_filter.py -s < $each > /dev/null
                        fi
                fi
        fi
done

echo 'Ham training...'
for each in `cat ${FULL_TRAIN_LIST_LOC}train_ham_filelist${RUN}.txt`;
do
        if [ $TOOL == $SA ];
        then
                sa-learn --ham $each >> $OUTFILE_TRAIN
        else
                if [ $TOOL == $BOGO ];
                then
                        bogofilter -n < $each >> $OUTFILE_TRAIN
                else
                        if [ $TOOL == $SB ];
                        then
                                sb_filter.py -g < $each > /dev/null
                        fi
```

```
                fi
        fi
done

echo 'Done training'
echo 'Done training' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN


echo "Running test for FULL_MESSAGE sample ${RUN}"
./test_sa1.sh ${RUN} > $OUTFILE_TEST

echo 'Done testing' >> $OUTFILE_TRAIN
date >> $OUTFILE_TRAIN
```

test_sa1.sh

```
#!/bin/sh
#FIRSTRUNDIR="/home/rhavens/Dropbox/code/python/source/runs/firstrun/"
FIRSTRUNDIR="/home/rhavens/Dropbox/code/python/source/runs/thirdrun"
#SAMPLESIZE=""
SAMPLESIZE="-5percent"

if [ -z $TOOL ];
then
        echo "TOOL environment variable is not set.  Exiting."
        exit 1
fi

RUN=1
if [ "x$1" != "x" ];
then
        if [ $1 -ge 1 -a $1 -le 5 ];
        then
                RUN=$1
        fi
fi

echo 'Filter testing...'
echo 'These should all be spam...'
for each in `cat ${FIRSTRUNDIR}test_spam_filelist${RUN}${SAMPLESIZE}.txt`;
do
        if [ $TOOL == 'spamassassin' ];
        then
                spamc -c < $each
        else
                if [ $TOOL == 'bogofilter' ];
                then
                        bogofilter --verbosity < $each
                else
                        if [ $TOOL == 'spambayes' ];
                        then
                                sb_filter.py < $each  | grep 'X-Spambayes-Classification:'
                        fi
                fi
        fi
done

echo 'These should all be ham...'
for each in `cat ${FIRSTRUNDIR}test_ham_filelist${RUN}${SAMPLESIZE}.txt`;
do
        if [ $TOOL == 'spamassassin' ];
        then
                spamc -c < $each
        else
                if [ $TOOL == 'bogofilter' ];
```

```
                then
                        bogofilter --verbosity < $each
                else
                        if [ $TOOL == 'spambayes' ];
                        then
                                sb_filter.py < $each  | grep 'X-Spambayes-Classification:'
                        fi
                fi
        fi
done

echo 'Done testing'
```

runset.sh

```
#!/bin/sh
#runset.sh - run a set of spam tests

export SA1='spamassassin'
export BOGO1='bogofilter'
export SB1='spambayes'

runsetvertical(){
        ./train_sabs-2.sh
        ./train_sabs-2.sh 2
        ./train_sabs-2.sh 3
        ./train_sabs-2.sh 4
        ./train_sabs-2.sh 5
}

runsethorizontal() {
        if [ -z $1 ];
        then
                DATASET=1
        else
                DATASET=$1
        fi
        ./train_safull-2.sh $DATASET
        ./train_sah-2.sh $DATASET
        ./train_sab-2.sh $DATASET
        ./train_sabs-2.sh $DATASET
}


runsa() {
        export TOOL=$SA1
#       runsetvertical 1
        runsethorizontal 1
        runsethorizontal 2
        runsethorizontal 3
        runsethorizontal 4
        runsethorizontal 5
}

runbogo(){
        export TOOL=$BOGO1
        runsethorizontal 1
        runsethorizontal 2
        runsethorizontal 3
        runsethorizontal 4
        runsethorizontal 5
}
runsb(){
        export TOOL=$SB1
        runsethorizontal 1
        runsethorizontal 2
```

```
        runsethorizontal 3
        runsethorizontal 4
        runsethorizontal 5
}

for each in '-5percent' '-1percent' '-halfpercent' '-tenthpercent';
do
        export SAMPLESIZE=${each}
        #echo "$SAMPLESIZE"
        #runbogo
        #runsb
        runsa
done
#./train_sabs-2.sh
#./train_sabs-2.sh 2
#./train_sabs-2.sh 3
#./train_sabs-2.sh 4
#./train_sabs-2.sh 5
#./train_safull-2.sh
#./train_sah-2.sh
#./train_sab-2.sh
#./train_sabs-2.sh

#./train_safull-2.sh 2
#./train_sah-2.sh 2
#./train_sab-2.sh 2
#./train_sabs-2.sh 2

#./train_safull-2.sh 3
#./train_sah-2.sh 3
#./train_sab-2.sh 3
#./train_sabs-2.sh 3

#./train_safull-2.sh 4
#./train_sah-2.sh 4
#./train_sab-2.sh 4
#./train_sabs-2.sh 4

#./train_safull-2.sh 5
#./train_sah-2.sh 5
#./train_sab-2.sh 5
#./train_sabs-2.sh 5

echo "Done with set"
```

data_normalizer.py

```
#!/usr/bin/python
#
#       data_normalizer.py - normalize the output from the 3 spam filtering tools (bogofilter,
spamassassin and spambayes) to csv formats that will be analyzed with R
#
#

#Sample Bogofilter:
#Filter testing...
#These should all be spam...
#X-Bogosity: Unsure, tests=bogofilter, spamicity=0.500000, version=1.2.0
#X-Bogosity: Unsure, tests=bogofilter, spamicity=0.500000, version=1.2.0

#Sample SpamBayes
#

#Sample SpamAssassin
#
#Filter testing...
```

```
        #These should all be spam...
        #X-Spambayes-Classification: spam; 1.00
        #X-Spambayes-Classification: spam; 1.00
        #X-Spambayes-Classification: spam; 1.00
        #X-Spambayes-Classification: spam; 1.00

        import sys
        import glob
        import os.path as ospath

        #_DATA_FILE_GLOB="/home/rhavens/Dropbox/code/python/source/runs/fourthrun/test_*_*.test-*.*.*.csv"
        _DATA_FILE_GLOB="/home/rhavens/Dropbox/code/python/source/runs/fourthrun/test_*_*.test-
*.spamassassin.?.csv"
        _HEADER = ('msg_type','verdict','score')

        def parse_bogo_line(msg_type, line):
                if 'X-Bogosity' not in line:
                        return []
                parsed_line = None

                sline = line.replace('X-Bogosity:','').split(', ')

                verdict = sline[0].lower()
                #tests = sline[2].split('=')[1]
                score = sline[2].split('=')[1]
                #X-Bogosity: Unsure, tests=bogofilter, spamicity=0.500000, version=1.2.0
                parsed_line = [msg_type, verdict, score]

                return parsed_line


        def parse_sa_line(msg_type, line):
                if '/' not in line:
                        return []
                parsed_line = None

                sline = line.split('/')
                score = sline[0]
                nscore = float(score)
                #I won't store threshold, as it's always 5 and not very meaningful for my study.
                threshold = sline[1]
                nthreshold = float(threshold)
                verdict = 'ham'
                if nscore >= nthreshold:
                        verdict = 'spam'

                #17.7/5.0
                parsed_line = [msg_type, verdict, score]

                return parsed_line


        def parse_sb_line(msg_type, line):
                if 'X-Spambayes' not in line:
                        return []

                parsed_line = None

                sline = line.replace(';',':').split(':')
                #nscore = float(sline[2].strip())
                verdict = sline[1].strip()
                score = sline[2].strip()

                #X-Spambayes-Classification: spam; 1.00
                parsed_line = [msg_type, verdict, score]

                return parsed_line

        def write_out_file(filename, data):
```

```
                print 'Writing',len(data),'lines to',filename
                #if True: return

                f_out = open(filename,'w')
                try:
                        f_out.write('\t'.join(_HEADER))
                        f_out.write('\n')

                        for line in data:
                                if line == None or len(line) < 1:
                                        continue
                                f_out.write('\t'.join(line))
                                f_out.write('\n')
                finally:
                        f_out.close()


        def parse_file(finfo):
                print 'Parsing',finfo["filename"]
                #test_spam_full.test.spamassassin.1-scrubbed-HAMONLY.csv
                newname_SPAM = finfo["filename"].replace(".csv",".parsed.SPAMONLY.csv")
                newname_HAM = finfo["filename"].replace(".csv",".parsed.HAMONLY.csv")
                spamlines = []
                hamlines = []
                f_in = open(finfo['filename'])
                try:
                        msg_type = ''
                        linenum = 0
                        curr_lines = None
                        for line in f_in:
                                linenum += 1
                                parsedline = ''
                                if line == None or "Filter testing" in line or 'Done testing' in line or
len(line) < 5:
                                        continue
                                if "should all be spam" in line:
                                        curr_lines = spamlines
                                        msg_type = 'spam'
                                        continue
                                if "should all be ham" in line:
                                        curr_lines = hamlines
                                        msg_type = 'ham'
                                        continue
                                if curr_lines == None:
                                        print 'Type identifier missing before line:',linenum,'-',line

                                        continue

                                if finfo['spamfilter'] == 'bogofilter':
                                        curr_lines.append(parse_bogo_line(msg_type, line))
                                elif finfo['spamfilter'] == 'spamassassin':
                                        curr_lines.append(parse_sa_line(msg_type, line))
                                elif finfo['spamfilter'] == 'spambayes':
                                        curr_lines.append(parse_sb_line(msg_type, line))
                                else:
                                        print 'Invalid spam filter name set:',finfo['spamfilter']
                                        continue
                                #Now, output the parsed line with the new data filename
                finally:
                        f_in.close()
                #print len(hamlines),'hamlines'
                #print len(spamlines),'spamlines'
                write_out_file(newname_SPAM, spamlines)
                write_out_file(newname_HAM, hamlines)



        def start():
                fileparts = []
```

```
                filenames = glob.glob(_DATA_FILE_GLOB)
                for each in filenames:
                        if each.endswith('ONLY.csv'):
                                continue
                        splitname = each.split('_')[2].split(".")
                        fileparts.append({"filename":each,                    "manpulation":splitname[0],
"samplesize":splitname[1].replace("test",""), "spamfilter":splitname[2], "samplenumber":splitname[3]})
                        #print splitname

                for thisfile in fileparts:
                        parse_file(thisfile)


        if __name__ == '__main__':
                start()
```

fourthrun-analysis.R

```
        #!/usr/bin/Rscript
        #       R analysis of fourth run for SpamAssassin data
        #
        #       ver 1.7a
        #
        #       Version info
        #       1.0 - 05/15/10 - initial version
        #       1.01 - 05/30/10 - Updated comments and text output to reflect SpamAssassin data vs other
tools' data
        #       1.5 - 5/30/10 - Refactored to use internal functions for all work
        #       1.6 - 6/28/10 - Reworked VERBOSE and non-VERBOSE output
        #       1.7 - 6/29/10 - Changed to write output directly to a file; added min/max/mean in output
columns; 3)[in progress] report percentage of correct/incorrect categorizations for each data set.
        #File naming convention
        #test_spam_h.test-tenthpercent.spambayes.5.csv
        #test_<ham/spam>_<test_type>.test-<samplesize>.<spamfilter>.<sample_number>.csv
        #test_[sp|h]am_[full|h|b|bs].test-[5percent|1percent|halfpercent|tenthpercent].[1-5].csv
        #
        VERBOSE=FALSE;#TRUE;
        SPAM_EXT="SPAMONLY.csv";
        HAM_EXT="HAMONLY.csv";
        OUTPUT_FILENAME="fourthrun-analysis-output.csv";
        #options(digits=15)

        fn_getpath = function()
        {
                # ------ Set up main variables ------
                lpath="/home/rhavens/Dropbox/code/python/source/runs/fourthrun/";
                wpath="c:\\db\\My Dropbox\\code\\python\\source\\runs\\fourthrun\\";

                if (.Platform$OS.type == "unix")
                {
                        if (VERBOSE) {
                                print("Linux platform");
                        }
                        work_path=lpath;
                # The else keyword below must come on the same line as the closing brace from the
associated if statement
                #retval = system("uname", wait=TRUE) #This picks up CYGWIN uname, so use DOS ver and
reverse logic instead
                } else {
                        #only two possible values are 'unix' and 'windows'
                        if (VERBOSE) {
                                print("most likely Windows platform");
                        }
                        work_path=wpath;
                }
                return(work_path)
```

```
	}

	fn_generate_filenames = function(base_dir) {
		if (VERBOSE) {
			print("-----------------------------------");
			print("Generating filename lists...");
		}
		manipulation=c("full","h","b","bs");
		samplesize=c("5percent","1percent","halfpercent","tenthpercent");
		spamfilter=c("bogofilter","spambayes","spamassassin");
		samplenumber=1:5;
		filefilter=c("HAMONLY","SPAMONLY")

		f_exists=vector();
		f_not_exists=vector();

		for ( m in manipulation) {
			for ( ss in samplesize) {
				for (sf in spamfilter) {
					for ( sn in samplenumber) {
						filename_root	=	paste("test_spam_",m,".test-
",ss,".",sf,".",sn,".parsed.", sep="");
						filename_spam = paste(filename_root,SPAM_EXT,sep="");
						filename_ham = paste(filename_root,HAM_EXT,sep="");


	fa_spam=file.access(names=paste(base_dir,filename_spam,sep=""));

	fa_ham=file.access(names=paste(base_dir,filename_ham,sep=""));

						if (fa_spam == 0 && fa_ham == 0) {
							f_exists = c(f_exists,filename_root);

						} else {

							f_not_exists = c(f_not_exists,filename_root);
						}
					}
				}
			}
		}
		if (VERBOSE) {
			print(sprintf("%d files do not exist.",length(f_not_exists)));
			print(sprintf("%d files do exist.",length(f_exists)));
		}
		return(list(exist_list=f_exists,nonexist_list=f_not_exists));
	}


	fn_loaddata = function(full_filename,sep="\t")
	{
		loaded_data = read.table(full_filename, header=TRUE, sep=sep, na.strings="NA", dec=".",
strip.white=TRUE)
		return(loaded_data)
	}


	fn_summary_stats = function(data_spam, data_ham)
	{
		sum_spam=summary(data_spam);
		sum_ham=summary(data_ham);
		if (VERBOSE) {
			print("-------");
			print('Summary statistics for each spam and ham:');
			print(sum_spam);
			print(sum_ham);
		}
```

```
                summary_stats=list("spam_mean"=mean(data_spam$score),     "ham_mean"=mean(data_ham$score),
"spam_min"=min(data_spam$score),      "spam_max"=max(data_spam$score),      "ham_min"=min(data_ham$score),
"ham_max"=max(data_spam$score));

                return(summary_stats);
        }


        fn_basic_stats = function(dataspam, dataham){
                sds=sd(dataspam$score);
                sdh=sd(dataham$score)
                vs=var(dataspam$score);
                vh=var(dataham$score);
                if (VERBOSE) {
                        print("-------");
                        print('Std Deviation & Variance:');
                        print(paste("spam-stddev: ",sds,"; variance",vs,sep=""));
                        print(paste("ham-stddev: ",sdh,"; variance",vs,sep=""));
                }
                return(list("sds"=sds,"sdh"=sdh,"vs"=vs,"vh"=vh));
        }

        fn_calc_hitrate = function(this_data){
                total_len = nrow(this_data)#length(this_data$type);
                correct = 0;
                for (i in 1:total_len) {
                        thisrow = this_data[i,];
                        t1=as.character(thisrow$msg_type);
                        if (is.null(t1)){
                                t1=as.character(thisrow$type);
                        }

                        v1=as.character(thisrow$verdict);
                        if (!is.null(t1) && !is.null(v1)) {
                                if (paste(t1,"x") == paste(v1,"x")) {
                                        correct = correct + 1;
                                }# else {
                                        #if (v1 != "unsure"){
                                        #       print(paste(v1,"!=",t1));
                                        #}
                                #}
                        } #else {
                        #       print(paste(v1,"-",t1));
                        #}
                }
                if (VERBOSE) {
                        print(paste("correct/total: ",correct,'/',total_len,sep=""));
                }

                hitrate = 100.0*(correct/total_len);
                return(hitrate);
        }

        fn_analyze = function(base_path, filebase) {
                test_type="none";
                test_score=-100;
                hamname=paste(filebase,HAM_EXT,sep="")
                spamname=paste(filebase,SPAM_EXT,sep="")
                if (VERBOSE) {
                        print(paste("---Analyzing",spamname,'and',hamname));
                }

                spam_filename=paste(base_path, spamname, sep="");
                ham_filename=paste(base_path, hamname, sep="");
                data_spam = fn_loaddata(spam_filename);
                data_ham = fn_loaddata(ham_filename);

                summary_stats = fn_summary_stats(data_spam, data_ham);
                hitrate_spam = fn_calc_hitrate(data_spam)
```

```
                hitrate_ham = fn_calc_hitrate(data_ham);
                var_info = fn_basic_stats(data_spam,data_ham);
                var_ratio = var_info$vh/var_info$vs;

                #print(var_ratio);
                #if ((is.nan(var_ratio)) ||(var_ratio < 0.1) || (var_ratio >= 10)) {
                        wtest = wilcox.test(data_spam$score, data_ham$score);
                        if (VERBOSE) {
                                print("Wilcoxcon test");
                                #print(paste("Wilcoxon p-value:",wtest$p.value));
                                print(wtest);
                        }
                        test_type="Wilcoxon";
                        test_score=wtest$p.value;
                #} else {
                #       ttest = t.test(data_spam$score, data_ham$score);
                #       if (VERBOSE) {
                #               print("T test");
                #               print(paste("T-test p-value:",ttest$p.value));
                #       }
                #       test_type="Student's T";
                #       test_score=ttest$p.value;
                #}

                return(list("spam_filename"=spamname,                                "ham_filename"=hamname,
"percent_correct_spam"=hitrate_spam,       "percent_correct_ham"=hitrate_ham,       "test_type"=test_type,
"test_score"=test_score,      "spam_mean"=summary_stats$spam_mean,      "ham_mean"=summary_stats$ham_mean,
"spam_min"=summary_stats$spam_min,   "spam_max"=summary_stats$spam_max,   "ham_min"=summary_stats$ham_min,
"ham_max"=summary_stats$ham_max,      "sds"=var_info$sds,      "sdh"=var_info$sdh,      "vs"=var_info$vs,
"vh"=var_info$vh));
        }


        fn_start = function(){
                base_path = fn_getpath();

                file_lists = fn_generate_filenames(base_path);
                exists_list = file_lists$exist_list;
                nonexists_list = file_lists$nonexist_list;
                if (VERBOSE && length(nonexists_list) > 0) {
                        print("These generated filenames do not exist:");
                        print(paste(nonexists_list,sep="\n"));
                }
                if (VERBOSE) {
                        print("-------");
                        print('Analysis:');
                        print("Are the Spam and Ham sets for any given test statistically different?");
                        print("If the variance is less than an order of magnitude different between the
pairs, ");
                        print("Student's T test is used.  Otherwise the Wilcoxcon test is used.");
                        print("");
                }

                f_out=file(OUTPUT_FILENAME,"w");
                cat(c("manipulation","samplesize","tool","samplenumber","percent_correct_spam","percent_co
rrect_ham","test_type","test_score","spam_min","spam_mean","spam_max","spam_std_dev","spam_variance","ham_
min","ham_mean","ham_max","ham_std_dev","ham_variance","spam_filename","ham_filename"),file=f_out,sep="\t"
);
                cat("\n", file=f_out);
                for (this_filebase in exists_list) {
                        #strsplit returns a list.  To change this to an array of strings ("character"
classes), use unlist on the strsplit output.
                        split_filebase = unlist(strsplit(this_filebase,"\\."));
                        manipulation=unlist(strsplit(split_filebase[1],"_"))[3];
                        samplesize=unlist(strsplit(split_filebase[2],'-'))[2];
                        tool=split_filebase[3];
                        samplenumber=split_filebase[4];
                        retval = fn_analyze(base_path, this_filebase);
```

```
                        cat(c(manipulation, samplesize, tool, samplenumber, retval$percent_correct_spam,
retval$percent_correct_ham,   retval$test_type,   retval$test_score,   retval$spam_min,   retval$spam_mean,
retval$spam_max,  retval$sds,  retval$vs,  retval$ham_min,  retval$ham_mean,  retval$ham_max,  retval$sdh,
retval$vs, retval$spam_filename, retval$ham_filename),file=f_out, sep="\t");
                        cat("\n", file=f_out);
                }
                close(f_out);
        }

        #---------Flow starts here----------
        fn_start();

        w=warnings();
        if (length(w) > 0){
                warnings();
        }

        quit();
```

randlines.py

```
        #!/usr/bin/python
        #
        # Randomly select a specific number of lines from one text file into another text file
        #
        # Version 1.1
        #
        # 1.0 - 12/23/09 - rwh - Initial version
        # 1.01 - 12/24/09 - rwh - Cleaned up output for stdout; added verbose output of line numbers to be
sampled
        # 1.02 - 01/04/10 - rwh - Added output to tell what percentage of file was sampled; changed code
to UNIX newlines
        #  1.03  -  01/04/10  -  rwh  -  Moved  percentage  size  output  after  checks  for  bogus
sample_size/file_size values
        # 1.04 - 03/27/10 - rwh - Fixed an error string for when a non-existent input file is specified;
fixed date stamps for version info.
        # 1.05 - 08/03/10 - rwh - Updated help
        # 1.06 - 08/14/10 - rwh - the previous update had removed a needed exception handler line, causing
it not to run
        # 1.1 - 08/14/10 - rwh - Added ability to specify samplines as a floating-point percentage.
        # 1.1 - 11/13/10 - rwh - Added a hidden parameter to enable the cautious sampling checks I
originally made mandatory.
        #               These checks were supposed to keep the sampler from continuing for a long time,
but ended up causing less than full samples
        #               to be taken if the number of samples was close to the length of the file.  You
can still specify it, but since I'm fairly sure
        #               it was a bad idea in the first place and yet want to hedge my bets on it, I've
hidden the functionality rather than removed it.
        #
        # Future:
        #       Optimize, optimize, optimize
        #               - Currently just checks [0] of the line array rather than "if number in
lines_list" since the list will be sorted lowest to highest the same as the line counting scheme.
        #       Allow samplines to be specified as a percentage; handle decimal percentages.
        #       Allow the user to specify that the line number sampled will be prepended to the line; need
to determine a separator to use or allow the user to specify one.
        #
        #

        import sys
        import getopt
        import os
        import os.path as ospath
        import random

        _VERSION = 1.11
```

```
        #I had to make this a class so that _verbose could be properly referenced.  Sadly, I don't know
why it would not show up as a global, even though _DEFAULT_SAMPLE_LINES did....
        class randlines:
                '''Select a specified number of lines as a random sample of a specified input text file'''
                _DEFAULT_SAMPLE_LINES = 100
                _DEFAULT_NUM_LINES = -1
                _verbose = False
                _DASH = '-'

                def usage(self):
                        '''Show the usage of the program'''
                        print '''randlines.py '''+str(_VERSION)+'''
        Usage: randlines.py -i <inputfile> [options]
                        -i, --inputfile <filename>      Source file (required)
                        -o, --outputfile <filename>     Output  file  (Use  "-"  for  stdout;  default  =
<srcfile>.<sample_size>.sample)
                        -s, --samplines <number> Number of sample lines to extract (default = 100)
                                        Can also specify as a decimal percent (e.g. 1% or 10.5%)
                        -n, --numlines <number>  Number of lines in file (default = read from input file)
                        -v, --verbose   Turn on verbose logging
                        -h, --help      This help'''
                        #-a, --cautious Cautious  sampling;  might  not  get  a  full  sample  size,  but
minimizes sampling time

                def get_params(self):
                        '''Parse out the command-line parameters'''
                        try:
                                opts,     args     =     getopt.getopt(sys.argv[1:],     "i:o:s:n:ahv",
["inputfile=","outputfile=","samplines=","numlines=","cautious","help","verbose"])
                        except getopt.GetoptError, err:
                                # print help information and exit:
                                print str(err) # will print something like "option -a not recognized"
                                self.usage()
                                sys.exit(1)
                        infilename = None
                        outfilename = None
                        numlines = self._DEFAULT_NUM_LINES
                        samplines = self._DEFAULT_SAMPLE_LINES
                        sample_percent = -1
                        cautious = False
                        for opt, arg in opts:
                                if opt in ('-i','--inputfile'):
                                        infilename = arg
                                elif opt in ('-o','--outputfile'):
                                        outfilename = arg
                                elif opt in ('-a','--cautious'):
                                        cautious = True
                                elif opt in ('-n', '--numlines'):
                                        if arg.isdigit():
                                                numlines = int(arg)
                                        else:
                                                print  'Number  of  file  lines  must  be  specified  as  a
number:',arg
                                                print 'Defaulting to read lines from file.'
                                                self.usage()
                                                sys.exit(1)
                                elif opt in ('-s', '--samplines'):
                                        if '%' in arg:
                                                s = arg.replace('%','')
                                                try:
                                                        sample_percent = float(s)/100.0
                                                except:
                                                        print  'The  specified  sample  percent  is  not  a
number:',arg
                                                        self.usage()
                                                        sys.exit(1)
                                        else:
                                                if arg.isdigit():
```

```
                                                samplines = int(arg)
                                        else:
                                                print 'Number of sample lines must be specified
as a number:',arg
                                                self.usage()
                                                sys.exit(1)
                                elif opt in ('-v','--verbose'):
                                        self._verbose = True
                                elif opt in ('-h', '--help'):
                                        self.usage()
                                        sys.exit()
                                else:
                                        assert False, "Unsupported option specified"
                                # ...
                        if infilename is None:
                                print 'Please specify an input filename'
                                self.usage()
                                sys.exit(1)
                        if outfilename is None or outfilename == '' or outfilename == infilename:
                                if sample_percent > 0:
                                        outfilename                                                    =
infilename+'.'+str(100*sample_percent)+'percent.sample'
                                else:
                                        outfilename = infilename+'.'+str(samplines)+'.sample'
                                if self._verbose:
                                        print 'Defaulting to output file name: ',outfilename
                        if not ospath.exists(infilename):
                                print 'Specified filename does not exist:',infilename
                                self.usage()
                                sys.exit(1)
                        if samplines < 1 and sample_percent < 1:
                                print 'Specified number of sample lines or sample percent < 1, defaulting
to 1 sample line.'
                                samplines = 1

                        return infilename, outfilename, samplines, sample_percent, numlines, cautious


                def get_sample_line_numbers(self, linecount, sample_size, cautious):
                        '''Randomly select the line numbers to sample from the source file'''
                        count = 0;
                        retval = []
                        #for count in range(1,linecount):
                        tries = 0
                        while len(retval) < sample_size:
                                tries += 1
                                #Put a maximum number of samples on it -- 1 billion sounds good enough to
me
                                if (cautious or tries > 1000000000) and tries > linecount*2:
                                        print  'Too  many  duplicate  lines  chosen...breaking  loop.'#this
should never happen
                                        break
                                nextval = random.randint(1,linecount)
                                if nextval not in retval:
                                        retval.append(nextval)
                        retval.sort()
                        if self._verbose:
                                if len(retval) <= 100:
                                        print 'Sampled line numbers:',','.join([str(x) for x in retval])
                                else:
                                        print   'Sampled   line   numbers   (trimmed   to   the   first
100):',','.join([str(x) for x in retval[:100]])
                        return retval


                def   get_lines(self,   in_filename,   out_filename,   sample_line_count,   sample_percent,
num_lines, cautious):
                        '''Get the sample lines and write them to the output'''
                        linecount = 0
```

```
                        if num_lines != self._DEFAULT_NUM_LINES:
                                if self._verbose:
                                        print 'Using specified number of lines in file:',num_lines
                                linecount = num_lines
                        else:
                                if self._verbose:
                                        print 'Getting line count from',in_filename,'...'
                                f = open(in_filename)
                                try:
                                        for line in f:
                                                linecount += 1
                                finally:
                                        f.close()
                                        #if self._verbose:
                                        #       print linecount,'lines in file'
                                        print linecount,'lines in file'

                        #print     '=-->',sample_line_count,'-',sample_percent,'--',round(linecount     *
sample_percent)
                        if sample_percent > 0:
                                sample_line_count = int(round(linecount * sample_percent))
                                if self._verbose:
                                        print    (100*sample_percent),'%    sample    of',linecount,'lines
is',sample_line_count,'lines.'

                        if sample_line_count >= linecount:
                                print  'Line  sample  size  ('+str(sample_line_count)+')  >=  lines  in  file
('+str(linecount)+').  Exiting.'
                                sys.exit(2)
                        print    str(round(float(sample_line_count)/float(linecount),5)*100.0)+'%   sample
size'
                        sample_line_nums   =   self.get_sample_line_numbers(linecount,   sample_line_count,
cautious)

                        #if self._verbose:
                        #       print                 'Sampling',sample_line_count,'                 lines
from',in_filename,'into',out_filename

                        out_tail = ''
                        if out_filename == self._DASH:
                                out_tail = 'to stdout'
                                #print 'Sampling',sample_line_count,' lines from',in_filename,'to stdout'
                        else:
                                out_tail = 'into '+out_filename
                                #print                 'Sampling',sample_line_count,'                 lines
from',in_filename,'into',out_filename
                        if sample_percent > 0:
                                print    'Sampling',sample_line_count,'    lines    (',sample_percent,'%)
from',in_filename,out_tail
                        else:
                                print 'Sampling',sample_line_count,' lines from',in_filename,out_tail
                        #print 'Sampling',sample_line_count,' lines from',in_filename,'into',out_filename
                        count = 0
                        written_lines = 0
                        #Handle stdout vs a filename
                        if out_filename == self._DASH:
                                f_out = sys.stdout;
                        else:
                                f_out = open(out_filename,'w')

                        try:
                                f_in = open(in_filename)
                                try:
                                        for line in f_in:
                                                count += 1
                                                if count == sample_line_nums[0]:#in sample_line_nums:
                                                        written_lines += 1
                                                        f_out.write(line)
                                                        sample_line_nums.remove(count)
```

```
                                                if len(sample_line_nums) == 0:
                                                        break
                                finally:
                                        f_in.close()
                        finally:
                                if out_filename == self._DASH:
                                        f_out = None
                                else:
                                        f_out.close()
                        if out_filename == self._DASH:
                                out_filename = 'stdout'
                        if self._verbose:
                                print written_lines,'lines written to',out_filename
                        #print
in_filename,self._DASH,out_filename,self._DASH,sample_line_count,self._DASH,linecount,self._DASH,sample_li
ne_nums


                def start(self):
                        '''The action of the class starts here'''
                        in_filename,  out_filename,  samplines,  sample_percent,  numlines,  cautious  =
self.get_params()
                        sample_line_numbers   =   self.get_lines(in_filename,   out_filename,   samplines,
sample_percent, numlines, cautious)

        if __name__ == '__main__':
                '''Running from a command line begins here'''
                print 'randlines.py starting...'
                rl = randlines()
                rl.start()
                print '...done.'
```

l_train.py

```
        #!/usr/bin/python
        # -*- coding: utf-8 -*-
        #
        # l_train.py - log file spam-filter trainer
        #
        #
        #       Version info:
        #       1.0 - 07/24/10 (rwh) - Initial version
        #
        #

        import sys
        import subprocess
        import l_salib

        # Pull in the local constants -- used by both l_train and l_test
        from l_check_common import *

        #Controls whether to run the actual spam filter tools or not
        _NO_RUN = False#True

        class l_train():
                verbose = False

                def usage(usage):
                        '''Show the usage of the program'''
                        print '''Usage: l_train.py -s <inputfile> [options]
                        -s, --spamfile <filename>       Spam source file (required)
                        -a, --hamfile <filename> Ham source file (required)
                        -f, --filtertool Spam  filter  tool  [spamassassin|spambayes|bogofilter]  (Default:
spamassassin)
                        -c, --chainwords <number>       Number of words to chain together before
```

```
                        running through spam filter (Default=1]
                -j, --chainjoinchar     character to join words for chainwords (Default=_)
                -t, --stackchains       Use all chain lengths up to chainwords (Default=False)
                -n, --normalizenumbers  Normalize numbers to zeros (0)
                -o, --noclear   Do not clear previous filter before training
                -l, --logtype   Type                            of                              log
[syslog|syslog_b|syslog_c|syslog_i|syslog_n|websphere|websphere_c|applog_fhd] (Default:applog_fhd)
                -h, --help      This help

        Make sure to use the same settings for training and testing.

        Examples:

        l_train.py      --normalizenumbers
                Turn the phrase 192,168.1.1 to 000.000.0.0

        l_train.py --chainwords=3
                Turn the phrase "Now is the time for all" into the line
                "Now_is_the is_the_time the_time_for time_for_all" before analysis

        l_train.py --chainwords=3 --chainjoinchar=Q --stackchains
                Turn the phrase "Now is the time for all" into the line
                "NowQisQthe isQtheQtime theQtimeQfor timeQforQall NowQis isQthe theQtime
                timeQfor forQall Now is the time for all " before analysis
        '''


                def train_line(self, line, cmd_line):
                        '''Train the filter with a given log line'''
                        cat_mail_lines = MAIL_TEMPLATE.replace(MESSAGE_HERE, line)
                        if _NO_RUN:
                                print '-------------------------'
                                print 'Message text:', cat_mail_lines
                                print '--'
                                print 'Command:', cmd_line
                                print '========================='

                        else:
                                p1    =    subprocess.Popen([cat_mail_lines],    stdout=subprocess.PIPE,
shell=True)
                                p2 = subprocess.Popen(cmd_line, stdin=p1.stdout, stdout=subprocess.PIPE,
shell=True)
                                output = p2.communicate()[0]
                                if self.verbose:
                                        print 'OUTPUT:',output

                def train(self, data, msg_type):
                        '''Train for all log lines in a given file'''
                        filename = data[msg_type][l_salib.VAL]
                        tool = data[TOOL][l_salib.VAL]
                        chain_len = data[CHAIN_LEN][l_salib.VAL]
                        parsed_lines = 0
                        non_parsed_lines = 0

                        print 'Training from',filename,'...'
                        line_count = 0
                        f_in = open(filename)
                        try:
                                #WebSphere logs can be multi-line, so state is stored in prevline; start
fresh if this is a WAS log.
                                if data[LOGTYPE][l_salib.VAL] == WEBSPHERE:
                                        l_salib.prevline = None

                                for line in f_in:
                                        if line.startswith('#') or len(line) < 10:
                                                continue
                                        line_count += 1
                                        p_line = None
                                        if data[LOGTYPE][l_salib.VAL] == WEBSPHERE:
```

```
                                                #need to be able to handle multi-line entries in this
code
                                                p_line,                 l_salib.prevline                =
l_salib.parse_line_text(data[LOGTYPE][l_salib.VAL], line)
                                                #If the line is not parsed correctly, just skip it.
                                                if p_line == None:
                                                        print 'Skipping multiline entry...'
                                                        continue
                                        else:
                                                p_line,                 l_salib.prevline                =
l_salib.parse_line_text(data[LOGTYPE][l_salib.VAL], line)
                                        if p_line == None or len(p_line) < 5 or len(p_line[5]) < 5:
                                                print 'Error parsing line:',line
                                                non_parsed_lines += 1
                                                continue
                                        parsed_lines += 1

                                        msg = p_line[5].replace('\r','').replace('\n','').strip()
                                        if data[NORM_NUM][l_salib.VAL]:
                                                msg = l_salib.normalize_numbers(msg)
                                        if self.verbose:
                                                print '_ORIG__>', line
                                                print '_PARSED_>', msg
                                                print 'Training as',msg_type

                                        text = msg
                                        if self.verbose:
                                          print '*Original line:',line
                                          print '*Parsed line:',text
                                        #Get word chains if necessary
                                        text      =      l_salib.get_chained_words(text,      chain_len,
data[CHAIN_JOIN_CHAR][l_salib.VAL])

                                        #Now, do the actual training for this line
                                        self.train_line(text, HAM_SPAM_CMD_LINES[msg_type][tool])
                                        if self.verbose and line_count % 100:
                                                print '.',

                        finally:
                                f_in.close()
                        if self.verbose:
                                print ''
                        print '...done training for',line_count,'non-comment, non-blank',msg_type,'lines'
                        print  parsed_lines,'lines  parsed  correctly;',non_parsed_lines,'lines  not  parsed
correctly.'

                def clear_filter(self, tool):
                        '''Clear the given filter'''
                        try:
                                retcode = subprocess.check_call(CMD_CLEAR_FILTER[tool], shell=True);
                                if retcode != 0:
                                        print    'An    error    occurred    while    clearing',tool,'filter.
Command:',_CMD_CLEAR_FILTER[tool]
                                        sys.exit(1)
                                else:
                                        if self.verbose:
                                                print tool,'filter cleared'
                        except:
                                print    'An    exception    occurred    while    clearing',tool,'filter.
Commmand:',_CMD_CLEAR_FILTER[tool]


                def start(self):
                        '''Start the work here'''
                        #Specify props needed, types, default values, etc.
                        short_params = "a:s:c:j:f:l:tnovh"
                        long_params                                                                     =
["hamfile=","spamfile=","chainwords=","chainjoinchar=","filtertool=","stackchain","logtype=","normalizenum
bers","noclear","verbose","help"]
```

```
                        data = {
                                SPAMFILE:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-s',l_salib.LNAME:'--
spamfile',l_salib.VAL:None, l_salib.REQUIRED:True, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                HAMFILE:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-a',l_salib.LNAME:'--
hamfile',l_salib.VAL:None, l_salib.REQUIRED:True, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                TOOL:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-f',l_salib.LNAME:'--
filtertool',l_salib.VAL:SA, l_salib.REQUIRED:False, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                CHAIN_LEN:{l_salib.TYPE:int.__name__,l_salib.SNAME:'-c',l_salib.LNAME:'--
chainwords',l_salib.VAL:DEFAULT_CHAIN_LEN,       l_salib.REQUIRED:False,       l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                CHAIN_JOIN_CHAR:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-
j',l_salib.LNAME:'--chainjoinchar',l_salib.VAL:DEFAULT_CHAIN_JOIN_CHAR,            l_salib.REQUIRED:False,
l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                CHAIN_STACK:{l_salib.TYPE:int.__name__,l_salib.SNAME:'-
t',l_salib.LNAME:'--stackchain',l_salib.VAL:DEFAULT_CHAIN_STACK,                  l_salib.REQUIRED:False,
l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                LOGTYPE:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-l',l_salib.LNAME:'--
logtype',l_salib.VAL:DEFAULT_LOGTYPE,          l_salib.REQUIRED:False,          l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                NORM_NUM:{l_salib.TYPE:None,l_salib.SNAME:'-n',l_salib.LNAME:'--
normalizenumbers',l_salib.VAL:False,          l_salib.REQUIRED:False,          l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                NOCLEAR:{l_salib.TYPE:None,l_salib.SNAME:'-o',l_salib.LNAME:'--
noclear',l_salib.VAL:False, l_salib.REQUIRED:False, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                VERBOSE:{l_salib.TYPE:None,l_salib.SNAME:'-v',l_salib.LNAME:'--
verbose',l_salib.VAL:self.verbose,           l_salib.REQUIRED:False,           l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                HELP:{l_salib.TYPE:None,l_salib.SNAME:'-h',l_salib.LNAME:'--
help',l_salib.VAL:False, l_salib.REQUIRED:False, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                        }

                        #Get properties
                        props = l_salib.get_params(sys.argv[1:], short_params, long_params, data, self)

                        if props[HELP][l_salib.VAL]:
                                self.usage()
                                sys.exit(0)
                        #----Validate properties
                        #verbose
                        if props[VERBOSE][l_salib.VAL]:
                                self.verbose = True

                        #ham input file
                        if props[SPAMFILE][l_salib.VAL] is None:
                                print 'Please specify a spam input file'
                                self.usage()
                                sys.exit(1)
                        else:
                                print 'Training with spam file:',props[SPAMFILE][l_salib.VAL]

                        #input file
                        if props[HAMFILE][l_salib.VAL] is None:
                                print 'Please specify a ham input file'
                                self.usage()
                                sys.exit(1)
                        else:
                                print 'Training with ham file:',props[HAMFILE][l_salib.VAL]

                        #spam filter tool
                        if props[TOOL][l_salib.VAL] is None:
                                print 'Defaulting to filter tool: SpamAssassin'
                                props[TOOL][l_salib.VAL] = SA
                        else:
                                props[TOOL][l_salib.VAL] = props[TOOL][l_salib.VAL].lower()
                                if props[TOOL][l_salib.VAL] in (SA, SB, BOGO):
                                        print 'Spam filter tool:',props[TOOL][l_salib.VAL]
                                else:
                                        print         'Invalid         spam         filter         tool
specified:',props[TOOL][l_salib.VAL]
```

```
                                        print 'Must be spamassassin, spambayes or bogofilter.'
                                        print 'Default is spamassassin.'
                                        self.usage()
                                        sys.exit(4)

                        #validate logtype
                        if props[LOGTYPE][l_salib.VAL] is None:
                                props[LOGTYPE][l_salib.VAL]=APPLOG_FHD
                        else:
                                props[LOGTYPE][l_salib.VAL] = props[LOGTYPE][l_salib.VAL].lower()
                                valid_vals   =   (SYSLOG,WEBSPHERE,   l_salib.SYSLOG_B,   l_salib.SYSLOG_C,
l_salib.SYSLOG_I, l_salib.SYSLOG_N, l_salib.WEBSPHERE_C, l_salib.APPLOG_FHD)
                                if props[LOGTYPE][l_salib.VAL] not in valid_vals:
                                        print 'Invalid logtype specified:',props[LOGTYPE][l_salib.VAL]
                                        print   'Must   be   '+'   '.join(valid_vals)+'.   (Default   is
'+DEFAULT_LOGTYPE+')'
                                        self.usage()
                                        sys.exit(1)
                                else:
                                        if props[LOGTYPE][l_salib.VAL] == SYSLOG:
                                                props[LOGTYPE][l_salib.VAL] = l_salib.SYSLOG_I
                                        elif props[LOGTYPE][l_salib.VAL] == WEBSPHERE:
                                                props[LOGTYPE][l_salib.VAL] = l_salib.WEBSPHERE_C
                                        if self.verbose:
                                                print 'Log file type:',props[LOGTYPE][l_salib.VAL]

                        #chain length
                        chainlen = props[CHAIN_LEN][l_salib.VAL]
                        if chainlen < 1:
                                if self.verbose:
                                        print 'Chain length < 1; defaulting to 1'
                                props[CHAIN_LEN][_VAL]=1
                        elif chainlen > CHAIN_LEN_MAX:
                                if self.verbose:
                                        print   'Chain   length   >   CHAIN_LENGTH_MAX.       Defaulting
to',CHAIN_LEN_MAX
                                props[CHAIN_LEN][l_salib.VAL]=CHAIN_LEN_MAX

                        if props[CHAIN_LEN][l_salib.VAL] != DEFAULT_CHAIN_LEN:
                                if self.verbose:
                                        print   'Chaining',props[CHAIN_LEN][l_salib.VAL],'words   together
for structure matching.'

                        #normalize numbers
                        if props[NORM_NUM][l_salib.VAL]:
                                if self.verbose:
                                        print 'Normalizing numbers to 0'

                        #Clear the spam filter unless otherwise specified
                        if not props[NOCLEAR][l_salib.VAL]:
                                self.clear_filter(props[TOOL][l_salib.VAL])

                        #print props

                        self.train(props, SPAMFILE)
                        self.train(props, HAMFILE)

        if __name__ == '__main__':
                ltr = l_train()
                try:
                        ltr.start()
                except KeyboardInterrupt, ex:
                        print '\nInterrupted by user.'
```

l_test.py

```
        #!/usr/bin/python
        # -*- coding: utf-8 -*-
        #
        # l_test.py - log file spam-filter tester
        #
        #       Version info:
        #       0.9 - 07/31/10 (rwh) - Initial version
        #       1.0 - 08/20/10 (rwh) - Added rawoutput and printmessage capabilities; tuned for final use.
        #       1.1 - 08/25/10 (rwh) - Added handling of stdin as a log entry source; changed rawoutput to
outputtype to allow 3 output types: raw (r), metrics_only (m) and log_metrics (l);
        #                                                       fixed   a   bug   that   caused   bogofilter
output to be doubled.
        #       1.2 - 09/04/10 (rwh) - Added WebSphere log parsing.   Tricky because WAS logs can be
multiline.
        #       1.3 - 12/4/10 (rwh) - Added printline capability.
        #
        #
        import sys
        import subprocess
        import l_salib
        import re
        import datetime
        import time

        # Pull in the local constants -- used by both l_train and l_test
        from l_check_common import *

        #Version info
        _VERSION = '1.3'

        #Controls whether to run the actual spam filter tools or not
        _NO_RUN = False#True

        class l_test():
                verbose = False
                very_verbose = False
                date_regex = re.compile('\d+\.\d+\d+\s+(\d+\-\d+\-\d+\s\d+\:\d+\:\d+)\,\d+')
                #e.g.: 2010-11-12 18:01:25
                date_format = '%Y-%m-%d %H:%M:%S'

                def usage(usage):
                        '''Show the usage of the program'''
                        print '''Usage: l_test.py -s <inputfile> [options]
                        -i, --inputfile <filename>      Log source file ; use '-' for STDIN (required)
                        -f, --filtertool Spam  filter  tool  [spamassassin|spambayes|bogofilter]  (Default:
spamassassin)
                        -c, --chainwords <number>       Number of words to chain together before
                                running through spam filter (Default=1]
                        -j, --chainjoinchar <char>      character   to   join   words   for   chainwords
(Default='''+DEFAULT_CHAIN_JOIN_CHAR+''')
                        -t, --stackchains       Use all chain lengths up to chainwords (Default=False)
                        -n, --normalizenumbers  Normalize numbers to zeros (0)
                        -l, --logtype   Type                                of                              log
[syslog|syslog_b|syslog_c|syslog_i|syslog_n|websphere|websphere_c|applog_fhd] (Default:applog_fhd)
                        -o, --outputtype <r|m|l|f>      Print the output in raw (r), metric_only (m) or
metric_full_log (f) format (Default='''+DEFAULT_OUTPUT_TYPE+''')
                        -p, --printmessage      Print the message after the score.   Mutually exclusive
from -r (Default=False)
                        -r, --printline Print the raw line after the score.  Mutually exclusive from -p
(Default=False)
                        -v, --verbose
                        -h, --help      This help

        Make sure to use the same settings for training and testing.
```

```
        Examples:

        l_train.py      --normalizenumbers
                Turn the phrase 192,168.1.1 to 000.000.0.0

        l_train.py --chainwords=3
                Turn the phrase "Now is the time for all" into the line
                "Now_is_the is_the_time the_time_for time_for_all" before analysis

        l_train.py --chainwords=3 --chainjoinchar=Q --stackchains
                Turn the phrase "Now is the time for all" into the line
                "NowQisQthe isQtheQtime theQtimeQfor timeQforQall NowQis isQthe theQtime
                timeQfor forQall Now is the time for all " before analysis
        '''

                def get_date_values(self, line):
                        '''Extract the date values from a line'''
                        try:
                                m = re.search(self.date_regex, line)
                                date_text = m.group(1)
                                dt = time.strptime(date_text, self.date_format)
                                retval = (str(long(time.mktime(dt))), m.group(1))
                                # 411386.125 2010-11-12 18:01:25,952 WARN
                        except Exception, ex:
                                retval = ('','')
                        return retval

                def  print_metrics(self,  tool,  input_line,  input_message,  output,  print_line=False,
print_message=False):
                        '''Print the metrics output by a given filter tool'''
                        add_line=''
                        if print_line:
                                add_line = '\t'+input_line.replace('\r','').replace('\n','')
                        elif print_message:
                                add_line = '\t'+input_message.replace('\r','').replace('\n','')
                        if tool == SB:
                                lines = output.replace('\r','')
                                for line in lines.split('\n'):
                                        if 'X-Spambayes-Classification' in line:
                                                sline     =     line.replace('X-Spambayes-Classification:
','').split(';')
                                                date_values = self.get_date_values(add_line)
                                                print
sline[0].strip()+'\t'+sline[1].strip()+'\t'+date_values[0]+'\t'+date_values[1]+'\t'+add_line
                                                break
                        elif tool == SA:
                                lines = output.replace('\r','')
                                for line in lines.split('\n'):
                                        if '/' in line:
                                                sline = line.strip().split('/')
                                                #411386.125 2010-11-12 18:01:25,952 WARN
                                                date_values = self.get_date_values(add_line)
                                                print
sline[0]+'\t'+sline[1]+'\t'+date_values[0]+'\t'+date_values[1]+'\t'+add_line
                                                break
                        elif tool == BOGO:
                                lines = output.replace('\r','')
                                for line in output.split('\n'):
                                        if 'X-Bogosity' in line:
                                                sline = output.strip().split(',')
                                                date_values = self.get_date_values(add_line)
                                                print                          sline[0].replace('X-Bogosity:
','')+'\t'+sline[2].replace('spamicity=','')+'\t'+date_values[0]+'\t'+date_values[1]+'\t'+add_line
                                                break
                                        #print '>',line

                def test_line(self, raw_line, line, cmd_line, tool, output_type):
                        '''Test a line against a given spam filter'''
```

```
                        cat_mail_lines = MAIL_TEMPLATE.replace(MESSAGE_HERE, line)

                        if _NO_RUN:
                                print 'Message text:',cat_mail_lines
                                print 'Command:',cmd_line
                        else:
                                output = None
                                try:
                                        p1  =  subprocess.Popen([cat_mail_lines],  stdout=subprocess.PIPE,
shell=True)
                                        p2       =       subprocess.Popen(cmd_line,       stdin=p1.stdout,
stdout=subprocess.PIPE, shell=True)
                                        output = p2.communicate()[0]
                                except Exception, ex:
                                        sys.stderr.write('Exception while running filter:'+str(ex)+'\n')
                                        sys.stderr.write('Attempted                                    command
1:'+str([cat_mail_lines])+'\n')
                                        sys.stderr.write('Attempted command 2:'+str(cmd_line)+'\n')
                                        return
                                output = output.strip('\n')
                                if output_type == 'r':
                                        print output.strip('\n')
                                elif output_type == 'm':
                                        self.print_metrics(tool, raw_line, line, output, False, False)
                                elif output_type == 'f':
                                        self.print_metrics(tool, raw_line, line, output, True, False)
                                elif output_type == 'l':
                                        self.print_metrics(tool, raw_line, line, output, False, True)
                                else:
                                        #This should never happen!
                                        print 'Invalid output type:',output_type

                def process_one_line(self, line, msg, data):
                        '''Clean up and check one line'''
                        if data[NORM_NUM][l_salib.VAL]:
                                msg = l_salib.normalize_numbers(msg)
                        if self.very_verbose:
                                print '_ORIG__>', line
                                print '_PARSED_>', msg
                        text = msg
                        tool = data[TOOL][l_salib.VAL]
                        #Get word chains if necessary
                        text      =      l_salib.get_chained_words(text,      data[CHAIN_LEN][l_salib.VAL],
data[CHAIN_JOIN_CHAR][l_salib.VAL])

                        #Now, do the actual training for this line
                        self.test_line(line,         text,         TEST_SPAM_CMD_LINES[tool],         tool,
data[OUTPUT_TYPE][l_salib.VAL])

                def test(self, data):
                        '''Run the test'''
                        filename = data[INPUTFILE][l_salib.VAL]
                        tool = data[TOOL][l_salib.VAL]
                        #chain_len = data[CHAIN_LEN][l_salib.VAL]
                        parsed_lines = 0
                        non_parsed_lines = 0
                        if self.very_verbose:
                                print 'Testing...'
                        line_count = 0
                        f_in = None
                        if filename == DASH:
                                f_in = sys.stdin
                        else:
                                f_in = open(filename)
                        try:
                                #WebSphere logs can be multi-line, so state is stored in prevline; start
fresh if this is a WAS log.
                                if data[LOGTYPE][l_salib.VAL] in WEBSPHERE_ENTRIES:
                                        prevline = None
```

```
                                for line in f_in:
                                        if line.startswith('#') or len(line) < 10 or len(line) > 10000:
                                                continue
                                        line_count += 1
                                        p_line = None
                                        correct_line_len = 5
                                        if data[LOGTYPE][l_salib.VAL] in WEBSPHERE_ENTRIES:
                                                correct_line_len = 4
                                                #need to be able to handle multi-line entries in this
code
                                                p_line,                     prevline                    =
l_salib.parse_line_text(data[LOGTYPE][l_salib.VAL], line, prevline)
                                                #If the line is not parsed correctly, just skip it.
                                                if p_line == None:
                                                        if self.verbose:
                                                                print 'Skipping multiline entry...'
                                                        continue
                                        else:
                                                correct_line_len = 5
                                                p_line,                     prevline                    =
l_salib.parse_line_text(data[LOGTYPE][l_salib.VAL], line)

                                        if p_line == None:
                                                #The line is a continuation line for WAS or was not
correctly parsed for SYSLOG
                                                #continuation_lines += 1
                                                continue
                                        elif len(p_line) < correct_line_len or len(p_line[-1]) < 5:
                                                print 'Error parsing line:',line
                                                non_parsed_lines += 1
                                                continue
                                        parsed_lines += 1

                                        if data[LOGTYPE][l_salib.VAL] in WEBSPHERE_ENTRIES:
                                                msg                         =                       p_line[-
1].replace('\r','').replace('\n','').strip()

                                                self.process_one_line(line, msg, data)
                                        else:
                                                msg = p_line[5].replace('\r','').replace('\n','').strip()

                                                self.process_one_line(line, msg, data)

                                        if self.very_verbose and line_count % 100:
                                                print '.',

                                #after the file has been looped, process the last WAS log entry
                                if data[LOGTYPE][l_salib.VAL] in WEBSPHERE_ENTRIES:
                                        if p_line is not None:
                                                msg                         =                       p_line[-
1].replace('\r','').replace('\n','').strip()
                                                self.process_one_line(line, msg, data)

                        finally:
                                f_in.close()
                        if self.very_verbose:
                                print ''
                        if self.verbose:
                                print     '...done     training     for',line_count,'non-comment,     non-
blank',data[LOGTYPE][l_salib.VAL],'lines'
                                print parsed_lines,'lines parsed correctly;',non_parsed_lines,'lines not
parsed correctly.'


                def start(self):
                        '''The main flow starts here'''
                        #Specify props needed, types, default values, etc.
                        short_params = "i:c:f:l:j:to:nvh"
```

```
                        long_params                                                                     =
["inputfile=","chainwords=","filtertool=","logtype=","stackchain","chainjoinchar","`outputtype","normalize
numbers","verbose","help"]
                        data = {
                                INPUTFILE:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-i',l_salib.LNAME:'--
inputfile',l_salib.VAL:None, l_salib.REQUIRED:True, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                TOOL:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-f',l_salib.LNAME:'--
filtertool',l_salib.VAL:SA, l_salib.REQUIRED:False, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                CHAIN_LEN:{l_salib.TYPE:int.__name__,l_salib.SNAME:'-c',l_salib.LNAME:'--
chainwords',l_salib.VAL:DEFAULT_CHAIN_LEN,       l_salib.REQUIRED:False,       l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                CHAIN_JOIN_CHAR:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-
j',l_salib.LNAME:'--chainjoinchar',l_salib.VAL:DEFAULT_CHAIN_JOIN_CHAR,            l_salib.REQUIRED:False,
l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                CHAIN_STACK:{l_salib.TYPE:int.__name__,l_salib.SNAME:'-
t',l_salib.LNAME:'--stackchain',l_salib.VAL:DEFAULT_CHAIN_STACK,                  l_salib.REQUIRED:False,
l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                LOGTYPE:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-l',l_salib.LNAME:'--
logtype',l_salib.VAL:DEFAULT_LOGTYPE,          l_salib.REQUIRED:False,          l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                NORM_NUM:{l_salib.TYPE:None,l_salib.SNAME:'-n',l_salib.LNAME:'--
normalizenumbers',l_salib.VAL:False,          l_salib.REQUIRED:False,          l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                OUTPUT_TYPE:{l_salib.TYPE:str.__name__,l_salib.SNAME:'-
o',l_salib.LNAME:'--outputtype',l_salib.VAL:DEFAULT_OUTPUT_TYPE,                  l_salib.REQUIRED:False,
l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                                VERBOSE:{l_salib.TYPE:None,l_salib.SNAME:'-v',l_salib.LNAME:'--
verbose',l_salib.VAL:self.verbose,           l_salib.REQUIRED:False,           l_salib.SPECIFIED_VAL:None,
l_salib.ERROR:None},
                                HELP:{l_salib.TYPE:None,l_salib.SNAME:'-h',l_salib.LNAME:'--
help',l_salib.VAL:False, l_salib.REQUIRED:False, l_salib.SPECIFIED_VAL:None, l_salib.ERROR:None},
                        }

                        #Get properties
                        props = l_salib.get_params(sys.argv[1:], short_params, long_params, data, self,
self.very_verbose)

                        if props[HELP][l_salib.VAL]:
                                self.usage()
                                sys.exit(0)
                        #----Validate properties
                        #verbose
                        if props[VERBOSE][l_salib.VAL]:
                                self.verbose = True

                        #input file
                        if props[INPUTFILE][l_salib.VAL] is None:
                                print 'Please specify a spam input file'
                                self.usage()
                                sys.exit(1)
                        else:
                                if self.verbose:
                                        if props[INPUTFILE][l_salib.VAL] == DASH:
                                                print 'Testing input from STDIN'
                                        else:
                                                print 'Testing input file:',props[INPUTFILE][l_salib.VAL]

                        #spam filter tool
                        if props[TOOL][l_salib.VAL] is None:
                                print 'Defaulting to filter tool: SpamAssassin'
                                props[TOOL][l_salib.VAL] = SA
                        else:
                                props[TOOL][l_salib.VAL] = props[TOOL][l_salib.VAL].lower()
                                if props[TOOL][l_salib.VAL] in (SA, SB, BOGO):
                                        if self.verbose:
                                                print 'Spam filter tool:',props[TOOL][l_salib.VAL]
                                else:
                                        print         'Invalid         spam         filter         tool
specified:',props[TOOL][l_salib.VAL]
```

```
                                        print 'Must be spamassassin, spambayes or bogofilter.'
                                        print 'Default is spamassassin.'
                                        self.usage()
                                        sys.exit(4)

                        #validate logtype
                        if props[LOGTYPE][l_salib.VAL] is None:
                                props[LOGTYPE][l_salib.VAL]=SYSLOG
                        else:
                                props[LOGTYPE][l_salib.VAL] = props[LOGTYPE][l_salib.VAL].lower()
                                valid_vals   =   (SYSLOG,WEBSPHERE,   l_salib.SYSLOG_B,   l_salib.SYSLOG_C,
l_salib.SYSLOG_I, l_salib.SYSLOG_N, l_salib.WEBSPHERE_C, l_salib.APPLOG_FHD)
                                if props[LOGTYPE][l_salib.VAL] not in valid_vals:
                                        print 'Invalid logtype specified:',props[LOGTYPE][l_salib.VAL]
                                        print   'Must   be   '+'   '.join(valid_vals)+'.   (Default   is
'+DEFAULT_LOGTYPE+')'
                                        self.usage()
                                        sys.exit(1)
                                else:
                                        if props[LOGTYPE][l_salib.VAL] == SYSLOG:
                                                props[LOGTYPE][l_salib.VAL] = l_salib.SYSLOG_I
                                        elif props[LOGTYPE][l_salib.VAL] == WEBSPHERE:
                                                props[LOGTYPE][l_salib.VAL] = l_salib.WEBSPHERE_C
                                        if self.very_verbose:
                                                print 'Log file type:',props[LOGTYPE][l_salib.VAL]

                        #chain length
                        chainlen = props[CHAIN_LEN][l_salib.VAL]
                        if chainlen < 1:
                                if self.very_verbose:
                                        print 'Chain length < 1; defaulting to 1'
                                props[CHAIN_LEN][_VAL]=1
                        elif chainlen > CHAIN_LEN_MAX:
                                if self.very_verbose:
                                        print   'Chain   length   >   CHAIN_LENGTH_MAX.       Defaulting
to',CHAIN_LEN_MAX
                                props[CHAIN_LEN][l_salib.VAL]=CHAIN_LEN_MAX

                        if props[CHAIN_LEN][l_salib.VAL] != DEFAULT_CHAIN_LEN:
                                if self.very_verbose:
                                        print   'Chaining',props[CHAIN_LEN][l_salib.VAL],'words   together
for structure matching.'

                        #output_type
                        output_type = props[OUTPUT_TYPE][l_salib.VAL]
                        if output_type is None or len(output_type) < 1:
                                output_type = DEFAULT_OUTPUT_TYPE
                        elif len(output_type) >  1:
                                        output_type = output_type[0]
                        if output_type not in VALID_OUTPUT_TYPES:
                                output_type = DEFAULT_OUTPUT_TYPE
                                print           'Invalid           output           type           specified
('+str(props[OUTPUT_TYPE][l_salib.VAL])+'); defaulting to',DEFAULT_OUTPUT_TYPE
                        #now put the normalized version back into the props structure
                        props[OUTPUT_TYPE][l_salib.VAL] = output_type

                        #normalize numbers
                        if props[NORM_NUM][l_salib.VAL]:
                                if self.very_verbose:
                                        print 'Normalizing numbers to 0'

                        self.test(props)

        if __name__ == '__main__':
                lte = l_test()
                try:
                        lte.start()
                except KeyboardInterrupt, ex:
                        print '\nInterrupted by user.'
```

```
                        #raise ex
```

l_salib.py

```
        #!/usr/bin/python
        #
        # salib.py - Syslog Analysis library - contains library functions used for other syslog analysis
tools
        #
        #       Version info:
        #       1.0 - 06/10 (rwh) - Initial version
        #       1.1     - 07/17/10 (rwh) - Added the ability to start from a particular line.  This will
allow for incremental handling of large files.
        #
        import getopt
        import sys
        import re

        _SPLIT_TEXT = ']: '
        _TEMPLATE_SRC_TEXT = '__DATA_HERE__'
        _NUMBERS = [str(n) for n in range(1,10)]
        STRIP_QUOTES = True
        TYPE='type'
        SNAME='short_name'
        LNAME='long_name'
        VAL='value'
        REQUIRED='required'
        SPECIFIED_VAL='specified_value'
        ERROR='error'
        SYSLOG_I = 'syslog_i'
        _SYSLOG_I_TEMPLATE    =    re.compile('''^(\w+\s+\d+\s+\d+:\d+:\d+)\s+(\w+)\s+(\S+):?\s+(.*)''')#1-
date;#2-server;#3-process;#4-message
        SYSLOG_N='syslog_n'
        _SYSLOG_N_TEMPLATE                                                                                  =
re.compile('''(\w+\s+\d+\s\d+:\d+:\d+)\s\<(\w+)\.(\w+)\>\s(\S+)\s(\S+):\s(.*)''')#1-date;#2-facility;#3-
severity;#4-server;#5-process;#6-message
        SYSLOG_B = 'syslog_b'
        _SYSLOG_B_TEMPLATE   =   re.compile('''(\w+\s+\d+\s\d+:\d+:\d+)\s(\S+)\s(.+?):\s(.*)''')#1-date;#2-
server;#3-process;#4-message
        SYSLOG_C = 'syslog_c'
        _SYSLOG_C_TEMPLATE = ''#'((\w+)\s\d+\s\d+:\d+:\d+)\s\<(\w+)\.(\w+)\>\s(\S+)\s(.*+):\s(.*)'
        WEBSPHERE_C = 'websphere_c'
        _WEBSPHERE_C_TEMPLATE                                                                                                  =
re.compile('''\[(\d+\/\d+\/\d+\s\d+:\d+:\d+\s\w+)\]\s(\w+)\s(\w+)\s+(\w)\s+(.*)''')
        APPLOG_FHD = 'applog_fhd'
        _APPLOG_FHD_TEMPLATE                    =                    re.compile('''\d+\.\d+\s+(\d+\-\d+\-
\d+\s+\d+:\d+:\d+),\d+\s+\w+\s+\[(.+?)\]\s+\[\d+\]\s+\[(.+?)\](.*)''')#1-date; #2-process; #3-server; #4-
message
        #e.g.: 411386.125 2010-11-12 18:01:25,953 WARN   [PersonReader]   [4468] [brk001.search.prod.ft-
b,b8] Relatives not found for results page. expected=1, missing=1, samples=[-2147514850], primary=-
2147514848, specGroup=api_matchSummary, relationship=mother
        _SPLIT_STRING = re.compile('\s+')


        def get_params(args, short_params, long_params, data, source, verbose=False):
                '''Parse out the command-line parameters'''
                try:
                        opts, args = getopt.getopt(args, short_params, long_params)
                except getopt.GetoptError, err:
                        # print help information and exit:
                        print str(err) # will print something like "option -a not recognized"
                        source.usage()
                        sys.exit(1)
                params=[]

                if verbose:
```

```
                print 'Running with these options:',' '.join(args)

        for opt, arg in opts:
                for key in data:
                        option = data[key]
                        if opt in (option[SNAME],option[LNAME]):
                                option[SPECIFIED_VAL]=arg

                                if option[TYPE] == None:
                                        option[VAL] = True
                                elif option[TYPE] == int.__name__:
                                        try:
                                                option[VAL]=int(arg)
                                        except Exception, ex:
                                                option[ERROR] = ex
                                elif option[TYPE] == float.__name__:
                                        try:
                                                option[VAL]=float(arg)
                                        except Exception, ex:
                                                option[ERROR] = ex
                                elif option[TYPE] == str.__name__:
                                        option[VAL] = arg
                                else:
                                        sys.stderr.write('Unknown          option          type:
'+str(option[TYPE])+'\n')
        return data


def get_template(template_file):
        '''Get the contents of a template file as a template string'''
        template = ''
        f_in = open(template_file)
        try:
                template = f_in.readlines()
        finally:
                try:
                        f_in.close()
                except:
                        sys.stderr.write('Error while closing'+str(template_file)+'\n')
        return ''.join(template)


def fill_template(template, line):
        '''Fill a template using a given line.  The template is usually some mail format for the
spam filters.'''
        sline = line
        if _SPLIT_TEXT in line:
                sline = line.split(_SPLIT_TEXT)[1]

        filled_template = template.replace(_TEMPLATE_SRC_TEXT, sline)
        return filled_template


def get_input_lines(filename, maxlines, startline=0):
        '''Get the content of a file, up to a maximum number of lines.'''
        f_in = open(filename)
        line_count = 0
        lines = []
        try:
                for line in f_in:
                        #skip lines up to specified starting line
                        if startline > line_count:
                                continue
                        line_count += 1
                        if line_count > maxlines:
                                sys.stderr.write('Input file > '+str(maxlines)+' lines.  Halting
input now.\n')
                                break
                        if line.startswith('#') or len(line) == 0:
```

```
                                continue
                        lines.append(line)

        finally:
                try:
                        f_in.close()
                except:
                        sys.stderr.write('Error while closing input file.\n')
        return lines


def get_chained_words(line, maxnum, word_join_char='_', stack_chain=False):
        '''Create a chained-words list from a given phrase and chain length, stacking chains if
specified.'''
        line_orig = line
        if maxnum <= 1:
                return line_orig
        #This one collapses multiple spaces and/or tabs
        sline = re.split(_SPLIT_STRING, line)
        #or
        #This one puts one space per position, thus retaining the actual number or spaces.  It
does not handle tabs.
        #sline = re.split(' ')
        all_words = []
        for length in range(maxnum,maxnum+1):
                cur_set = []
                for startpos in range(0,len(sline)):
                        if startpos+length > len(sline):
                                continue
                        cur_word = []
                        for pos in range(0,length):
                                cur_word.append(sline[startpos+pos])

                        cur_set.append(word_join_char.join(cur_word))
                all_words.extend(cur_set)
        if stack_chain and maxnum > 1:
                next_set = get_chained_words(line, maxnum-1, word_join_char, stack_chain)
                all_words.append(next_set)
        #print all_words
        return ' '.join(all_words)


def parse_syslogs(line, verbose, template):
        '''Parse a given syslog line'''
        parsed_line = None
        try:
                m = re.search(template, line)
                if m is None:
                        if verbose:
                                print '******RE match not found for line'
                                print '**TEMPLATE**',template
                                print '-=LINE-=',line
                else:
                        parsed_line = m.groups()
        except:
                if verbose:
                        print 'Exception while parsing line:',line
                raise
        return parsed_line


#************ Start Display Current Environment ************
#WebSphere   Platform   7.0.0.3   [ND   7.0.0.3   cf030911.09]   running   with   process   name
srvl8055Cell01\srvu8035Node01\DSP_8035 and process id 385114
#Host Operating System is AIX, version 5.3
#Java version = 1.6.0, Java Compiler = j9jit24, Java VM name = IBM J9 VM
#was.install.root = /opt/wsph/AppServer
#user.install.root = /opt/wsph/AppServer/profiles/Appserver01
#Java Home = /opt/wsph/AppServer/java/jre
```

```
        #ws.ext.dirs                                                                    =
/opt/wsph/AppServer/java/lib:/opt/wsph/AppServer/profiles/Appserver01/classes:/opt/wsph/AppServer/classes:
/opt/wsph/AppServer/lib:/opt/wsph/AppServer/installedChannels:/opt/wsph/AppServer/lib/ext:/opt/wsph/AppSer
ver/web/help:/opt/wsph/AppServer/deploytool/itp/plugins/com.ibm.etools.ejbdeploy/runtime
        #Classpath                                                                      =
/opt/wsph/AppServer/profiles/Appserver01/properties:/opt/wsph/AppServer/properties:/opt/wsph/AppServer/lib
/startup.jar:/opt/wsph/AppServer/lib/bootstrap.jar:/opt/wsph/AppServer/lib/jsf-
nls.jar:/opt/wsph/AppServer/lib/lmproxy.jar:/opt/wsph/AppServer/lib/urlprotocols.jar:/opt/wsph/AppServer/d
eploytool/itp/batchboot.jar:/opt/wsph/AppServer/deploytool/itp/batch2.jar:/opt/wsph/AppServer/java/lib/too
ls.jar
        #Java                          Library                          path                          =
/opt/wsph/AppServer/java/jre/lib/ppc:/usr/lib:/opt/wsph/AppServer/java/jre/lib/ppc:/opt/wsph/AppServer/jav
a/jre/lib/ppc/j9vm:/opt/wsph/AppServer/java/jre/lib/ppc/j9vm:/opt/wsph/AppServer/java/jre/lib/ppc:/opt/wsp
h/AppServer/java/jre/../lib/ppc:/usr/lib:/opt/wsph/AppServer/java/jre/lib/ppc:/opt/wsph/AppServer/java/jre
/lib/ppc/j9vm:/opt/wsph/AppServer/java/jre/lib/ppc/j9vm:/opt/wsph/AppServer/java/jre/lib/ppc:/opt/wsph/App
Server/java/jre/../lib/ppc:/opt/wsph/AppServer/bin:/opt/oracle/v10201cli/lib32:/usr/lib
        #************* End Display Current Environment *************
        #[6/10/10  15:39:35:250  MDT]  0000000c  webcontainer    I  com.ibm.ws.wswebcontainer.VirtualHost
addWebApplication SRVE0250I: Web Module WebSphere ASYNC Response Servlet Application has been bound to
default_host[*:9080,*:80,*:9443,*:5060,*:5061,*:443,*:9081,*:9082,*:9083,srvu8201.ldsglobal.net:9081,srvu8
201.ldsglobal.net:80,srvu8201.ldsglobal.net:9444,srvu8201.ldsglobal.net:5063,srvu8201.ldsglobal.net:5062,s
rvu8201.ldsglobal.net:443,srvu8201.ldsglobal.net:9080,srvu8201.ldsglobal.net:9443,srvu8201.ldsglobal.net:5
060,srvu8201.ldsglobal.net:5061,srvu8036.lab.ldsglobal.net:9080,srvu8036.lab.ldsglobal.net:80,srvu8036.lab
.ldsglobal.net:9443,srvu8036.lab.ldsglobal.net:5060,srvu8036.lab.ldsglobal.net:5061,srvu8036.lab.ldsglobal
.net:443,srvu8035.lab.ldsglobal.net:9080,srvu8035.lab.ldsglobal.net:80,srvu8035.lab.ldsglobal.net:9443,srv
u8035.lab.ldsglobal.net:5060,srvu8035.lab.ldsglobal.net:5061,srvu8035.lab.ldsglobal.net:443,srvl8481.ch.or
g:80,srvl8481.ch.org:9443,srvl8481.ch.org:5060,srvl8481.ch.org:5061,srvl8481.ch.org:443,srvu7999.ch.org:90
80,srvu7999.ch.org:80,srvu7999.ch.org:9443,srvu7999.ch.org:5060,srvu7999.ch.org:5061,srvu7999.ch.org:443,s
rvu8816.ldsglobal.net:9080,srvu8816.ldsglobal.net:80,srvu8816.ldsglobal.net:9443,srvu8816.ldsglobal.net:50
60,srvu8816.ldsglobal.net:5061,srvu8816.ldsglobal.net:443,srvu8816.ldsglobal.net:9081,srvu8816.ldsglobal.n
et:9444,srvu8816.ldsglobal.net:5063,srvu8816.ldsglobal.net:5062,srvu8505.ldsglobal.net:9080,srvu8505.ldsgl
obal.net:80,srvu8505.ldsglobal.net:9443,srvu8505.ldsglobal.net:5060,srvu8505.ldsglobal.net:5061,srvu8505.l
dsglobal.net:443,*:9091,*:9086,srvu7965.ch.org:9080,srvu7965.ch.org:80,srvu7965.ch.org:9443,srvu7965.ch.or
g:5060,srvu7965.ch.org:5061,srvu7965.ch.org:443,srvl7042.ch.org:9080,srvl7042.ch.org:80,srvl7042.ch.org:94
43,srvl7042.ch.org:5060,srvl7042.ch.org:5061,srvl7042.ch.org:443,srvl7041.ch.org:9080,srvl7041.ch.org:80,s
rvl7041.ch.org:9443,srvl7041.ch.org:5060,srvl7041.ch.org:5061,srvl7041.ch.org:443,*:9084,*:9085,srvu7530.c
h.org:9060,srvu7530.ch.org:80,srvu7530.ch.org:9062,srvu7530.ch.org:9076,srvu7530.ch.org:9077,srvu7530.ch.o
rg:443,srvu7529.ch.org:9080,srvu7529.ch.org:80,srvu7529.ch.org:9443,srvu7529.ch.org:5060,srvu7529.ch.org:5
061,srvu7529.ch.org:443,*:9088,srvl7023.ch.org:9080,srvl7023.ch.org:80,srvl7023.ch.org:9443,srvl7023.ch.or
g:5060,srvl7023.ch.org:5061,srvl7023.ch.org:443,srvu8132.lab.ldsglobal.net:9080,srvu8132.lab.ldsglobal.net
:80,srvu8132.lab.ldsglobal.net:9443,srvu8132.lab.ldsglobal.net:5060,srvu8132.lab.ldsglobal.net:5061,srvu81
32.lab.ldsglobal.net:443,srvu8173.lab.ldsglobal.net:9080,srvu8173.lab.ldsglobal.net:80,srvu8173.lab.ldsglo
bal.net:9443,srvu8173.lab.ldsglobal.net:5060,srvu8173.lab.ldsglobal.net:5061,srvu8173.lab.ldsglobal.net:44
3,*:9089,srvu8461.ch.org:9080,srvu8461.ch.org:80,srvu8461.ch.org:9443,srvu8461.ch.org:5060,srvu8461.ch.org
:5061,srvu8461.ch.org:443,srvu8462.ch.org:9080,srvu8462.ch.org:80,srvu8462.ch.org:9443,srvu8462.ch.org:506
0,srvu8462.ch.org:5061,srvu8462.ch.org:443,srvu8744.ch.org:9080,srvu8744.ch.org:80,srvu8744.ch.org:9443,sr
vu8744.ch.org:5060,srvu8744.ch.org:5061,srvu8744.ch.org:443,srvl7415.ch.org:9080,srvl7415.ch.org:80,srvl74
15.ch.org:9443,srvl7415.ch.org:5060,srvl7415.ch.org:5061,srvl7415.ch.org:443,srvl7416.ch.org:9080,srvl7416
.ch.org:80,srvl7416.ch.org:9443,srvl7416.ch.org:5060,srvl7416.ch.org:5061,srvl7416.ch.org:443,srvu7530.ch.
org:19060,srvu7530.ch.org:19062,srvu7530.ch.org:19076,srvu7530.ch.org:19077].

        def parse_waslog_line(line, verbose, template, prevline):
                '''Parse WebSphere App Server log line'''
                if line.startswith('***') and 'Start Display Current Environment' in line:
                        if verbose:
                                print 'WAS restart record found'
                        return None, None
                elif not line.startswith('['):
                        if prevline is not None:
                                if type(prevline[3]) != type('str'):
                                        print prevline[3]
                                prevline[3] += ' '+line
                                return None, prevline
                parsed_line = None
                line=line.replace('\r','').replace('\n','')
                try:
                        #For now, throw away all lines from multiline entries, except for the one with
the date.
                        m = re.search(template, line)
```

```
                        if m is None:
                                if verbose:
                                        print 'RE match not found for line'
                                parsed_line = (None, None, None, None, None, line.strip())
                        else:
                                parsed_line = m.groups()
                                prevline = parsed_line
                except:
                        if verbose:
                                print 'Exception while parsing waslog line:',line
                        raise
                return parsed_line, prevline


        def parse_syslog_i(line, verbose):
                '''Parse type i syslog line'''
                #sample lines:
                #Aug  4 16:55:48 slinger dhclient: bound to 192.168.10.150 -- renewal in 3161 seconds.
                #Aug  5 14:05:48 josephus dhclient: DHCPACK from 192.168.10.4
                #Aug  8 22:09:19 goteam ntpd[4503]: kernel time sync status change 4001
                #Aug    9  11:49:04  samwise  kernel:  [       0.000000]    modified:  000000007feff000  -
000000007ff00000 (ACPI NVS)
                #(1=date)(2=facility=NONE)(3=severity=NONE)(4=server)(5=process)(6=message)
                parsed_line = None
                temp_parsed_line = parse_syslogs(line, verbose, _SYSLOG_I_TEMPLATE)
                if len(temp_parsed_line) == 4:
                        #Pad the return value up to 6 entries, matching syslog_n output length
                        parsed_line    =    (temp_parsed_line[0],    None,    None,    temp_parsed_line[1],
temp_parsed_line[2], temp_parsed_line[3])
                else:
                        print 'Incorrect number of elements parsed for syslog_n entry.  Results may be
invalid.'
                        parsed_line = temp_parsed_line
                return parsed_line

        def parse_syslog_n(line, verbose):
                '''Parse type n syslog line'''
                #sample lines:
                #Sep 22 11:10:01 <user.notice> myserv1-n logrotate: ALERT exited abnormally with [1]
                #Nov  11  20:22:09  <daemon.info>  myserv2-n  named[3031]:  unexpected  RCODE  (SERVFAIL)
resolving 'inkcommercial.com/MX/IN': 62.254.254.124#53
                #Jan 4 8:01:04 <daemon.debug> myserv3-b SSLVPN: Sending servlet CONNMAN_STATUS response to
fd 16
                #(1=date)(2=facility)(3=severity)(4=server)(5=process)(6=message)
                parsed_line = None
                temp_parsed_line = parse_syslogs(line, verbose, _SYSLOG_N_TEMPLATE)
                if len(temp_parsed_line) == 6:
                        parsed_line = temp_parsed_line
                else:
                        print 'Incorrect number of elements parsed for syslog_n entry.  Results may be
invalid.'
                return parsed_line

        def parse_syslog_b(line, verbose):
                '''Parse type b syslog line'''
                #Jun 13 04:02:03 aji syslogd 1.4.1: restart.
                #Jun 13 04:02:03 aji rpc.idmapd[3022]: nss_getpwnam: name 'infauser' not found in domain
'localdomain'
                #Jun 13 04:03:28 aji selogrd[16915]: Cannot resolve destination file. Entry ignored.
                #(1=date)(2=facility=NONE)(3=severity=NONE)(4=server)(5=process)(6=message)
                parsed_line = None
                temp_parsed_line = parse_syslogs(line, verbose, _SYSLOG_B_TEMPLATE)
                if temp_parsed_line is not None and len(temp_parsed_line) == 4:
                        #Pad the return value up to 6 entries, matching syslog_n output length
                        parsed_line    =    (temp_parsed_line[0],    None,    None,    temp_parsed_line[1],
temp_parsed_line[2], temp_parsed_line[3])
                else:
                        parsed_line = temp_parsed_line
                return parsed_line
```

```
        def parse_syslog_c(line, verbose):
                '''Parse type c syslog line'''
                print 'parse_syslog_c is not yet implemented'
                sys.exit(5)
                return line

        def parse_applog_fhd(line, verbose):
                '''Parse FHD app log line'''
                #sample lines:
                #411327.594 2010-11-12 18:00:27,440 WARN  [BrokerImpl]  [1371] [brk001.search.prod.ft-
b,b8] match() - Unable to communicate with updaters. Results only reflect baked state.
                #411328.969 2010-11-12 18:00:28,813 ERROR [BrokerImpl]  [23201] [brk001.search.prod.ft-
b,b8] Error matching on updaters. Attempt 1 of 3. Will retry.
                #411386.125 2010-11-12 18:01:25,952 WARN  [PersonReader]  [4468] [brk001.search.prod.ft-
b,b8] Relatives not found for results page. expected=1, missing=1, samples=[-2147514849], primary=-
2147514848, specGroup=api_matchSummary, relationship=father

                #(1=date)(2=facility=NONE)(3=severity=NONE)(4=process)(5=server)(6=message)
                parsed_line = None
                temp_parsed_line = parse_syslogs(line, verbose, _APPLOG_FHD_TEMPLATE)
                if temp_parsed_line == None:
                        sys.stderr.write('Unable to parse line properly: '+str(line)+'\n')
                elif len(temp_parsed_line) == 4:
                        #Pad the return value up to 6 entries, matching syslog_n output length
                        parsed_line    =    (temp_parsed_line[0],    None,    None,    temp_parsed_line[2],
temp_parsed_line[1], temp_parsed_line[3].strip('\r'))
                else:
                        print 'Incorrect number of elements parsed for applog_fhd entry.  Results may be
invalid.'
                        parsed_line = temp_parsed_line
                return parsed_line

        def parse_websphere_c(line, verbose, prevline):
                '''Parse WebSphere log line'''
                #sample log data
                #************ Start Display Current Environment ************
                #WebSphere  Platform  6.1  [ND  6.1.0.23  cf230910.10]    running  with  process  name
cell\srvu4160_AppSvr01\ERS_4160 and process id 466954
                #Detailed IFix information: No IFixes applied to this build
                #Host Operating System is AIX, version 5.3
                #Java version = 1.5.0, Java Compiler = j9jit23, Java VM name = IBM J9 VM
                #was.install.root = /opt/wsph/AppServer
                #user.install.root = /opt/wsph/AppServer/profiles/AppServer01
                #Java Home = /opt/wsph/AppServer/java/jre
                #ws.ext.dirs                                                                                =
/opt/wsph/AppServer/java/lib:/opt/wsph/AppServer/profiles/AppServer01/classes:/opt/wsph/AppServer/classes:
/opt/wsph/AppServer/lib:/opt/wsph/AppServer/installedChannels:/opt/wsph/AppServer/lib/ext:/opt/wsph/AppSer
ver/web/help:/opt/wsph/AppServer/deploytool/itp/plugins/com.ibm.etools.ejbdeploy/runtime
                #Classpath                                                                                  =
/opt/wsph/AppServer/profiles/AppServer01/properties:/opt/wsph/AppServer/properties:/opt/wsph/AppServer/lib
/startup.jar:/opt/wsph/AppServer/lib/bootstrap.jar:/opt/wsph/AppServer/lib/j2ee.jar:/opt/wsph/AppServer/li
b/lmproxy.jar:/opt/wsph/AppServer/lib/urlprotocols.jar:/opt/wsph/AppServer/deploytool/itp/batchboot.jar:/o
pt/wsph/AppServer/deploytool/itp/batch2.jar:/opt/wsph/AppServer/java/lib/tools.jar
                #Java                          Library                          path                       =
/opt/wsph/AppServer/java/jre/bin:/opt/wsph/AppServer/java/jre/bin:/opt/wsph/AppServer/java/jre/bin/classic
:/opt/wsph/AppServer/java/jre/bin:/opt/wsph/AppServer/bin:/opt/oracle/v10201cli/lib32:/opt/wsph/AppServer/
java/jre/bin/j9vm:/opt/wsph/AppServer/java/jre/bin/j9vm:/opt/wsph/AppServer/java/jre/bin/j9vm:/usr/lib:/op
t/wsph/AppServer/lib/WMQ/java/lib
                #************ End Display Current Environment ************
                #[7/6/10 23:01:21:921 MDT] 00003ac8 UserGrant      W    USER_GRANT Insert <<<Insert - Id:
null - OrgId: null - Role: ROLE_USER - User: 157616 - UserProfessionalCenterRights: >>>
                #[7/6/10 23:01:21:952 MDT] 00003ac8 SecurityBean  W   Error with user contact info setup.
Redirecting to LookingTo page for user=157616
                #[7/6/10 23:01:33:640 MDT] 00003ac8 WebContainer  E    SRVE0255E: A WebGroup/Virtual Host
to handle /_WS/PT has not been defined.
                #[7/6/10 23:02:53:688 MDT] 00003593 SecurityBean  W   Error with user contact info setup.
Redirecting to LookingTo page for user=157616
                #[7/6/10 23:03:05:473 MDT] 00003593 JobSearchBean I   ----->Start job search
```

```
                #[7/6/10 23:03:05:489 MDT] 00003593 JobSearchBean I    ------>End proximity.  Elapse Time
in seconds:3.0E-6
                #[7/6/10 23:03:05:492 MDT] 00003593 JobSearchBean I    Job Search criteria: location =
Sugar Hill GA 30518
                #latitude = 34.121634
                #longitude = -84.048862
                #radius = distance-amount.twentyfive
                #unit = MILE
                #
                #[7/6/10 23:03:10:631 MDT] 00003593 JobSearchBean I    ------>End Query.  Elapse Time in
seconds:5.133251
                #[7/6/10 23:03:44:364 MDT] 0000375b JobSearchBean I    Job Search criteria: location =
Sugar Hill GA 30518
                #latitude = 34.121634
                #longitude = -84.048862
                #radius = distance-amount.ten
                #unit = MILE
                #
                #[7/6/10 23:03:47:504 MDT] 0000375b JobSearchBean I    ------>End Query.  Elapse Time in
seconds:3.137292
                #[7/6/10 23:03:47:508 MDT] 0000375b JobSearchBean I    ------>End Search.  Elapse Time in
seconds:3.146098
                #[7/6/10 23:04:22:284 MDT] 0000375b GisServiceImp I   Initializing GIS Service Proxy
                #[7/6/10 23:05:02:395 MDT] 000039d7 JobSearchBean I    ------>End proximity.  Elapse Time
in seconds:2.0E-6
                #[7/6/10 23:05:02:397 MDT] 000039d7 JobSearchBean I    Job Search criteria: location =
Sugar Hill GA 30518
                #latitude = 34.121634
                #longitude = -84.048862
                #radius = distance-amount.ten
                #unit = MILE
                #
                #[7/6/10 23:05:05:463 MDT] 000039d7 JobSearchBean I    ------>End Query.  Elapse Time in
seconds:3.062806
                #[7/6/10 23:05:27:791 MDT] 00003ac8 JobSearchBean I    Job Search criteria: location =
Sugar Hill GA 30518
                #latitude = 34.121634
                #longitude = -84.048862
                #radius = distance-amount.ten
                #unit = MILE
                #
                #[7/6/10 23:07:15:294 MDT] 000039d7 WebContainer  E    SRVE0255E: A WebGroup/Virtual Host
to handle /_WS/PT has not been defined.
                #[7/6/10 23:07:25:505 MDT] 00003ac8 SRTServletReq E    SRVE0133E: An error occurred while
parsing parameters. java.net.SocketTimeoutException: Async operation timed out
                #       at
com.ibm.ws.tcp.channel.impl.AioTCPReadRequestContextImpl.processSyncReadRequest(AioTCPReadRequestContextIm
pl.java:157)
                #       at
com.ibm.ws.tcp.channel.impl.TCPReadRequestContextImpl.read(TCPReadRequestContextImpl.java:109)
                #       at
com.ibm.ws.http.channel.impl.HttpServiceContextImpl.fillABuffer(HttpServiceContextImpl.java:4127)
                #       at
com.ibm.ws.http.channel.impl.HttpServiceContextImpl.readSingleBlock(HttpServiceContextImpl.java:3371)

                #(1=date)(2=facility=NONE)(3=severity)(4=server=NONE?)(5=process)(6=message)

                #Parsing options:
                #1. Only look at rows with time stamps
                #2. Append lines without timestamps to the end of most recent line with timestamp
                #3. duplicate most recent timestamp for each line without a timestamp
                #4. Does this matter?  I'm throwing away the timestamp anyway.
                #4a. It only matters for separating out the log data (timestamp, hex#, class name, log
level letter [I/W/E] from the text of the log entry
                #parsed_line = (None,None,None,None,line)
                parsed_line = None
                temp_parsed_line,  prevline  =  parse_waslog_line(line,  verbose,  _WEBSPHERE_C_TEMPLATE,
prevline)
                if temp_parsed_line is None:
```

```
                        if verbose:
                                print 'Line skipped or part of previous entry'
                elif len(temp_parsed_line) == 5:
                        #Pad the return value up to 6 entries, matching syslog_n output length
                        parsed_line    =    (temp_parsed_line[0],    None,    temp_parsed_line[3],    None,
temp_parsed_line[2], temp_parsed_line[3], temp_parsed_line[4])
                else:
                        sys.stderr.write('Incorrect    number    of    elements    parsed    for    waslog_c
entry('+str(len(temp_parsed_line))+', should be 5).  Results may be invalid.\n')
                        #parsed_line = (None,None,None,None,None,temp_parsed_line)
                        parsed_line = temp_parsed_line
                return parsed_line, prevline


        def parse_line_text(logtype, line, prevline=None, verbose=False):
                '''Clean up line and call correct method to parse for the given type of log line'''
                p_line = None
                if STRIP_QUOTES:
                        line = line.replace('\"','')
                if logtype == APPLOG_FHD:
                        p_line = parse_applog_fhd(line, verbose)
                elif logtype == SYSLOG_I:
                        p_line = parse_syslog_i(line, verbose)
                elif logtype == SYSLOG_N:
                        p_line = parse_syslog_n(line, verbose)
                elif logtype == SYSLOG_B:
                        p_line = parse_syslog_b(line, verbose)
                elif logtype == SYSLOG_C:
                        p_line = parse_syslog_c(line, verbose)
                elif logtype == WEBSPHERE_C:
                        p_line, prevline = parse_websphere_c(line, verbose, prevline)
                else:
                        print 'Unknown log file type:',logtype
                return p_line, prevline

        def normalize_numbers(text):
                '''Convert all digits to 0'''
                for n in _NUMBERS:
                        text = text.replace(n,'0')
                return text
```

l_check_common.py

```
        #!/usr/bin/python
        #f
        # l_check_common.py - Common variables and methods for log file spam-filter training and testing
systems
        #
        #
        #       Version info:
        #       1.0 - 07/31/10 (rwh) - Initial version
        #       1.1 - 08/20/10 (rwh) - Added vars for rawoutput and printmessage
        #       1.2 - 12/04/10 (rwh) - Added var for printline
        #
        #

        import sys
        import os
        import l_salib

        program_name=sys.argv[0].split(os.sep)[-1]

        DASH = '-'
        HAMFILE = 'hamfile'
        SPAMFILE = 'spamfile'
        INPUTFILE = 'inputfile'
```

```
        NORM_NUM = 'normalize_numbers'
        CHAIN_LEN = 'chain_length'
        CHAIN_JOIN_CHAR='chain_join_char'
        CHAIN_STACK = 'chain_stack'
        TOOL = 'tool'
        DEFAULT_CHAIN_LEN = 1
        DEFAULT_CHAIN_JOIN_CHAR = '_'
        DEFAULT_CHAIN_STACK = False
        CHAIN_LEN_MAX = 10
        NOCLEAR = 'noclear'
        OUTPUT_TYPE = 'outputtype'
        PRINTMESSAGE = 'printmessage'
        PRINTLINE = 'printline'
        VERBOSE = 'verbose'
        HELP = 'help'
        SA = "spamassassin"
        SB = "spambayes"
        BOGO = "bogofilter"
        LOGTYPE = 'logtype'
        SYSLOG = 'syslog'
        APPLOG_FHD = 'applog_fhd'
        WEBSPHERE = 'websphere'
        WEBSPHERE_C = 'websphere_c'
        WEBSPHERE_ENTRIES = (WEBSPHERE, WEBSPHERE_C)
        #DEFAULT_LOGTYPE = SYSLOG
        DEFAULT_LOGTYPE = APPLOG_FHD
        VALID_OUTPUT_TYPES = ('r','m','l','f')
        DEFAULT_OUTPUT_TYPE = VALID_OUTPUT_TYPES[1]

        CMD_CLEAR_FILTER    =    {SA:'sa-learn    --clear',    SB:'sb_filter.py    -n',    BOGO:'if    [    -f
~/.bogofilter/wordlist.db ]; then rm ~/.bogofilter/wordlist.db; fi'}
        HAM_SPAM_CMD_LINES   =   {HAMFILE:{SA:'sa-learn   --ham',SB:'sb_filter.py   -g',BOGO:'bogofilter   -
n'},SPAMFILE:{SA:'sa-learn --spam',SB:'sb_filter.py -s',BOGO:'bogofilter -s'}}
        TEST_SPAM_CMD_LINES = {SA:'spamc -c',SB:'sb_filter.py',BOGO:'bogofilter --verbosity'}

        VAL=l_salib.VAL
        TYPE=l_salib.TYPE
        SNAME=l_salib.SNAME
        LNAME=l_salib.LNAME
        REQUIRED=l_salib.REQUIRED
        SPECIFIED_VAL=l_salib.SPECIFIED_VAL
        ERROR=l_salib.ERROR

        MESSAGE_HERE = '__MESSAGE__HERE__'

        MAIL_TEMPLATE = '''cat <<END_TEXT
        Return-Path: skip@pobox.com
        Delivery-Date: Sat May  1 20:47:01 2010
        From: spamtest.rhavens@byu.edu (Russel Havens)
        Date: Sat, 1 May 2010 19:47:01 -0600
        Subject: Test Message

        __MESSAGE__HERE__

        END_TEXT
        '''
```

l_runtest.sh

```
        #!/bin/sh
        # Test Harness: run tests for sa, sb and bogofilter, with various word chainings
        # Train all 3 filters for a given chain level, then test all 3 filters
        #

        #N_SAMPLE_NAMED="/home/rhavens/Dropbox/data/syslog/nov/sample-named.txt"
        #N_SAMPLE_NNTP="/home/rhavens/Dropbox/data/syslog/nov/sample-nntp.txt"
```

```
        #Full test document: "~/Dropbox/data/syslog/nov/named-typhoond.log"
        I_SAMPLE_KERNEL="/home/rhavens/Dropbox/data/syslog/itet/syslogs-itet-kernel.txt.25.sample"
        I_SAMPLE_DHCLIENT="/home/rhavens/Dropbox/data/syslog/itet/syslogs-itet-dhclient.txt.25.sample"
        I_KERNEL_DHCLIENT="/home/rhavens/Dropbox/data/syslog/itet/kernel-dhclient.txt"

        do_test() {
                if [ -f ${OUTPUTFILE} ];
                then
                        TODAY=`date +'%Y-%m-%d_%H-%M-%S'`
                        cp -f ${OUTPUTFILE} ${OUTPUTFILE}.bak.${TODAY}
                fi
                echo "Running test for ${TOOL}, chainlevels=${CHAINLEVEL}"
                ./l_test.py  -i  ${TEST_SET}  ${NORM_NUMBERS}  -l  ${LOG_TYPE}  -c  ${CHAINLEVEL}  -j
${CHAIN_JOIN_CHAR} -f ${TOOL} -t > ${OUTPUTFILE}
                #This makes it easier for a ctrl-c to break out of the script
                sleep 1
        }

        train_test() {
                if [ -z "${CHAINLEVEL}" ];
                then
                        CHAINLEVEL=1
                fi
                #sample set:
                ./l_train.py  -s  ${SAMPLE_SPAM}  -a  ${SAMPLE_HAM}  ${NORM_NUMBERS}  -l  ${LOG_TYPE}  -c
${CHAINLEVEL} -j ${CHAIN_JOIN_CHAR} -f spamassassin ${STACKPARAM}
                ./l_train.py  -s  ${SAMPLE_SPAM}  -a  ${SAMPLE_HAM}  ${NORM_NUMBERS}  -l  ${LOG_TYPE}  -c
${CHAINLEVEL} -j ${CHAIN_JOIN_CHAR} -f spambayes ${STACKPARAM}
                ./l_train.py  -s  ${SAMPLE_SPAM}  -a  ${SAMPLE_HAM}  ${NORM_NUMBERS}  -l  ${LOG_TYPE}  -c
${CHAINLEVEL} -j ${CHAIN_JOIN_CHAR} -f bogofilter ${STACKPARAM}

                #Don't just stomp on an existing output filed -- it may have taken a long time to build!
                TOOL="spamassassin"
                STACKPARAM='-t'
                OUTPUTFILE="dhclient_kernel-sa-c${CHAINLEVEL}${NORM_NUMBERS}-j${CHAIN_JOIN_CHAR}-
stack.csv"
                do_test
                STACKPARAM=''
                OUTPUTFILE="dhclient_kernel-sa-c${CHAINLEVEL}${NORM_NUMBERS}-j${CHAIN_JOIN_CHAR}-
nostack.csv"
                do_test
                TOOL="spambayes"
                STACKPARAM='-t'
                OUTPUTFILE="dhclient_kernel-sb-c${CHAINLEVEL}${NORM_NUMBERS}-j${CHAIN_JOIN_CHAR}-
stack.csv"
                do_test
                STACKPARAM=''
                OUTPUTFILE="dhclient_kernel-sb-c${CHAINLEVEL}${NORM_NUMBERS}-j${CHAIN_JOIN_CHAR}-
nostack.csv"
                do_test
                TOOL="bogofilter"
                STACKPARAM='-t'
                OUTPUTFILE="dhclient_kernel-bogo-c${CHAINLEVEL}${NORM_NUMBERS}-j${CHAIN_JOIN_CHAR}-
stack.csv"
                do_test
                STACKPARAM=''
                OUTPUTFILE="dhclient_kernel-bogo-c${CHAINLEVEL}${NORM_NUMBERS}-j${CHAIN_JOIN_CHAR}-
nostack.csv"
                do_test
                echo 'Done with test.'
                date
        }

        echo 'Starting...'
        date
        #kernel records reported as SPAM
        #dhclient records reported as HAM
        NORM_NUMBERS=""
        #"-n"
```

```
#NORM_NUMBERS=""
LOG_TYPE=syslog_i
CHAIN_JOIN_CHAR=_
SAMPLE_SPAM=$I_SAMPLE_KERNEL
SAMPLE_HAM=$I_SAMPLE_DHCLIENT
TEST_SET=$I_KERNEL_DHCLIENT
CHAINLEVEL=1
train_test
CHAINLEVEL=3
train_test
CHAINLEVEL=5
train_test
CHAINLEVEL=7
train_test
CHAINLEVEL=9
train_test
```

matchrate.py

```
#!/usr/bin/python
#
#       matchrate.py  -  Report  the  percentage  of  correct  message  type  matches  for  output  of
l_test.py
#

import sys
import os
import os.path as ospath

_DIR_NAME = '/home/rhavens/Dropbox/code/python/source/loganalysis/output/'
_SA = '-sa-'
_SB = '-sb-'
_BOGO = '-bogo-'

def process_file(filename, tool):
        match = 0
        miss = 0
        f_in = open(filename)
        try:
                for line in f_in:
                        sline = line.lower().split('\t')
                        if len(sline) < 3:
                                print 'Invalid line:',line
                                print filename
                                continue
                        if 'detected_type' in line or 'threshold' in line:
                                continue
                        if tool == _SA:
                                if (sline[0] == 'spam' and float(sline[1]) >= 3.0) or (sline[0]
== 'ham' and float(sline[1]) < 3.0):
                                        match += 1
                                else:
                                        miss += 1
                        else:
                                if sline[0] == sline[1]:
                                        match += 1
                                else:
                                        miss += 1
        finally:
                f_in.close()
        return match, miss

def start():
        print 'filename  match_rate'
        filelist = os.listdir(_DIR_NAME)
        for filename in filelist:
```

```
                        if filename.startswith('dhclient_kernel') and '-scrubbed' in filename:
                                tool = None
                                if _BOGO in filename:
                                        tool = _BOGO
                                elif _SB in filename:
                                        tool = _SB
                                elif _SA in filename:
                                        tool = _SA
                                else:
                                        print 'File type not recognized by name:',filename
                                        continue

                                match,miss =process_file(_DIR_NAME+filename, tool)
                                if match+miss < 1:
                                        print 'Problem parsing records in file:',_DIR_NAME+filename
                                print filename+'\t'+str(round((float(match)/float(match+miss)*100.00),3))


        if __name__ == '__main__':
                start()
        l_final_run_stage.sh
        #!/bin/sh
        do_training() {
                if [ ${samplesize} == "small" ];
                then
                        spamfile="sysout-files-111210-18_46-48_training.log"
                        hamfile="sysout-files-111210-18_NO_46-48_training-full.log.33.sample"
                else
                        spamfile="sysout-files-111210-18_45-48-all-spam-training.log"
                        hamfile="sysout-files-111210-without_18_45-48.log.348.sample"
                fi
                ./l_train.py        -s        /home/rhavens/Dropbox/data/fhd/applog/${spamfile}        -a
/home/rhavens/Dropbox/data/fhd/applog/${hamfile}  -c ${chain} ${normalize} -l applog_fhd -f spamassassin
                ./l_train.py        -s        /home/rhavens/Dropbox/data/fhd/applog/${spamfile}        -a
/home/rhavens/Dropbox/data/fhd/applog/${hamfile}  -c ${chain} ${normalize} -l applog_fhd -f spambayes
                ./l_train.py        -s        /home/rhavens/Dropbox/data/fhd/applog/${spamfile}        -a
/home/rhavens/Dropbox/data/fhd/applog/${hamfile}  -c ${chain} ${normalize} -l applog_fhd -f bogofilter
        }

        do_tests() {
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111210.log -c ${chain}
${normalize} -l applog_fhd -f spamassassin -o f > output-final_stage_sysout-files-111210-${samplesize}-sa-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111210.log -c ${chain}
${normalize} -l applog_fhd -f spambayes -o f > output-final_stage_sysout-files-111210-${samplesize}-sb-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111210.log -c ${chain}
${normalize} -l applog_fhd -f bogofilter -o f > output-final_stage_sysout-files-111210-${samplesize}-bogo-
c${chain}${normalize}.csv

                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111610.log -c ${chain}
${normalize} -l applog_fhd -f spamassassin -o f > output-final_stage_sysout-files-111610-${samplesize}-sa-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111610.log -c ${chain}
${normalize} -l applog_fhd -f spambayes -o f > output-final_stage_sysout-files-111610-${samplesize}-sb-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111610.log -c ${chain}
${normalize} -l applog_fhd -f bogofilter -o f > output-final_stage_sysout-files-111610-${samplesize}-bogo-
c${chain}${normalize}.csv

                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111810.log -c ${chain}
${normalize} -l applog_fhd -f spamassassin -o f > output-final_stage_sysout-files-111810-${samplesize}-sa-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111810.log -c ${chain}
${normalize} -l applog_fhd -f spambayes -o f > output-final_stage_sysout-files-111810-${samplesize}-sb-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-111810.log -c ${chain}
${normalize} -l applog_fhd -f bogofilter -o f > output-final_stage_sysout-files-111810-${samplesize}-bogo-
c${chain}${normalize}.csv
```

```
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-112110.log -c ${chain}
${normalize} -l applog_fhd -f spamassassin -o f > output-final_stage_sysout-files-112110-${samplesize}-sa-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-112110.log -c ${chain}
${normalize} -l applog_fhd -f spambayes -o f > output-final_stage_sysout-files-112110-${samplesize}-sb-
c${chain}${normalize}.csv
                ./l_test.py -i /home/rhavens/Dropbox/data/fhd/applog/sysout-files-112110.log -c ${chain}
${normalize} -l applog_fhd -f bogofilter -o f > output-final_stage_sysout-files-112110-${samplesize}-bogo-
c${chain}${normalize}.csv
        }

        for samplesize in "small" "big";
        do
                for chain in 1 2 3 4;
                do
                        for normalize in "-n" "";
                        do
                                echo "----Training for ${chain} ${normalize} ${samplesize} - `date`"
                                do_training
                                echo "****Testing for ${chain} ${normalize} ${samplesize}- `date`"
                                do_tests
                        done
                done
        done

        scrubfiles.py

        #!/usr/bin/python

        import os

        files = os.listdir("c:/data/ThesisFeb")
        for filename in files:
                if   (not   filename.endswith('.csv'))   or   '.scrubbed.'   in   filename   or   (not
filename.startswith('output-final')):
                        continue
                newfilename = filename.replace('.csv','.scrubbed.csv')
                print 'Processing',filename
                f_in = open(filename)
                print 'newfilename=',newfilename
                try:
                        f_out = open(newfilename,'w')
                        try:
                                for line in f_in:
                                        if len(line.strip()) < 11:
                                                continue
                                        elif 'Unable to parse line properly' in line:
                                                continue
                                        else:
                                                f_out.write(line)
                                f_out.flush()
                        finally:
                                f_out.close()
                finally:
                        f_in.close()
```

so-graphs.R

```
        #!/cygdrive/c/Program Files/R/R-2.12.0/bin/Rscript
        # so-graphs.R - Generate Spam filter Output GRAPHs from .csv files
        #
        #       Version info
```

```
#       1.0 - 02/11 (rwh) - Initial version - used for November FHD applogs.
#       1.1     - 03/26/11 (rwh) - Bumped up size of axis labels; fixed numerous small issues.
#       1.2 - 04/04/11 (rwh) - Updated to analyze FHD syslogs from March.
#       1.3 - 04/09/11 (rwh) - Updated to handle both Nov and Mar files.
#       1.4 - 05/07/11 (rwh) - Added minset directory handling (to allow redoing just a few
graphs)
#

require(car);
require(lattice);

utc_offset = 7*60*60;
log_set = 'minset'; #could be fall or spring or minset

#Function to generate and save out a graph of the specified dataset
generate_graph = function(pngname, dataset, outage_times, filename, doJitter)
{
        #Generate graph from spam set only
        print(paste('Generated .png file:',pngname));

        #dataset$dt = strptime(dataset$datetime, '%Y-%m-%d %H:%M:%S');
        #dataset$dtepoch = as.integer(as.POSIXct(spamrows$dt));

        tryCatch({
                png(pngname,  width=1200,  height=1000,  unit='px',  pointsize=12,  bg='white',
res=NA, restoreConsole=TRUE);
                lefttick=min(dataset$date_epoch);
                righttick=max(dataset$date_epoch);

                par(cex=1.25, cex.lab=2.5, cex.axis=2.5,cex.main=2, mar=c(8,9,4,4), ann=FALSE);
                #scatterplot(dataset$date_epoch, dataset$score, xaxt='n');
                #scatterplot(dataset$date_epoch, dataset$score, xaxp=c(lefttick,righttick,6));
                ##smoothScatter gives a Very different view of the data (more like a heatmap):
smoothScatter(dataset$date_epoch, dataset$score);

                #scatterplot(dataset$date_epoch, dataset$score, jitter=list(x=1, y=1));
                if (doJitter == TRUE) {
                        scatterplot(dataset$date_epoch,  dataset$score,  jitter=list(x=1,  y=1),
boxplots="xy");
                        #plot(jitter(dataset$date_epoch), jitter(dataset$score));
                } else {
                        scatterplot(dataset$date_epoch, dataset$score, boxplots="xy");
                        #plot(dataset$date_epoch, dataset$score);
                }

                line_count = 0;
                for (vline in outage_times) {
                        line_count = line_count+1;
                        if (log_set == 'fall') {
                                abline(v=vline[1],col='red');
                        } else {
                                if ((line_count %% 2) == 0) {
                                        abline(v=vline,col='blue');
                                } else {
                                        abline(v=vline,col='red');
                                }
                        }
                }
                #title(main=fname,   sub='Epoch   Time   vs.   Filter   Score   ',ylab='Filter
Score',xlab='Epoch Time');
                title(main=fname,ylab='Filter Score',xlab='Epoch Time');

        }, finally = {
                dev.off();
        })#End of tryCatch
}

get_col_headers = function(fname)
{
```

```
		skiplines = 0;
		#Set the column headers according to the filter tool type, since they're not in these raw
files
		col_headers = NULL;
		if (regexpr('avg.csv',fname) > 0) {
			col_headers = c('period','score_total','score_count','score_mean');
			skiplines = 1;
		} else {
			if (regexpr('-sa-',fname) > 0) {
				col_headers                                                                    =
c('score','threshold','date_epoch','datetime','emptycol','message');
				threshold = '-3.0-';
			} else {
				if (regexpr('-sb-',fname) > 0) {
					col_headers                                                            =
c('verdict','score','date_epoch','datetime','emptycol','message');
				} else {
					if (regexpr('-bogo-',fname) > 0) {
						col_headers                                                    =
c('verdict','score','date_epoch','datetime','emptycol','message');
					} else {
						#It isn't one of the expected files, so skip it
						col_headers = Null;#or is that next();?
					}
				}
			}
		}
		if (is.null(col_headers)) {
			print(paste('get_col_headers did not properly handle file named',fname));
		}

		#The result of the last line in a function is that function's return value
		list(col_headers,skiplines);
	}

	print('Starting so-graphs.R ');
	#Just change to the directory with the files so that image handling is pathless
	oldpath = getwd();
	workingpath_fallfiles = 'C:/data/ThesisFeb/';
	workingpath_springfiles = 'C:/data/ThesisMar/';
	workingpath_minfileset = 'C:/data/ThesisMinCSVSet';
	workingpath = workingpath_springfiles;
	if (log_set=='fall') {
		workingpath = workingpath_fallfiles;
	} else {
		if (log_set == 'minset') {
		workingpath = workingpath_minfileset
		}
	}
	print(paste('Working path:',workingpath));
	setwd(workingpath);

	#Don't forget: list.files() takes a REGEX, NOT a GLOB.
	filepath = '.';
	filenames_fall = list.files(path='.', 'output-final_stage_sysout-files-11.*\\.csv');
	#This  is  the  normal  one!#filenames  =  list.files(path='.',  'output-final_stage_sysout-files-
11.*.csv')
	#filenames = list.files(path='.', 'output-final_stage_sysout-files-112.*.csv')
	#filenames = list.files(path='.', 'output-final_stage_sysout-files-11.*-sb-c.*.csv')
	#filenames = list.files(path='.', 'output-final_stage_sysout-files-11.*\\.scrubbed.csv')
	#filenames = list.files(path='.', 'output-final_stage_sysout-files-11.*avg.csv')
	filenames_spring = list.files(path='.', 'syslog.*scrubbed.*csv');
	filenames_minfileset = list.files(path='.', '.*scrubbed\\.csv');
	filenames = filenames_spring;
	if (log_set == 'fall') {
		filenames = filenames_fall;
	} else {
		if (log_set == 'minset') {
			filenames = filenames_minfileset;
```

```
		}
	}
	#print(paste('Filespec:',filenames));

	#print('-----1');
	#print(length(filenames[]));

	#eg
	#New FamilySearch Site Down - 12 Nov 2010 - Outage: 18:48 - 19:28   -> 1289587680 through
1289590080

	outage_dates = c('111210','111610','111810','112110')
	#actual:
	#outage_times = c(1289587680,1289920080,1290073740,1290359460)
	#recalculated:
	#outage_times + 7 hours -- it appears that the timestamps are being rolled to forward to UTC (+7
hours), even though they are already in UTC, so the abline must correct for this
	outage_times_fall                                                                                =
c(1289587680+utc_offset,1289920080+utc_offset,1290073740+utc_offset,1290359460+utc_offset)
	#outage_times                                                                                    =
c(1289587680+utc_offset,1289920080+utc_offset,1290073740+utc_offset,1290359460+utc_offset)#129038[1-7]000
	outage_times = c(1300487400,1300942980,1301072580,1301444220)

	#####outages_app1 = c(1300487400,1300498200,1300942980,1300946340);
	outages_app1 = c(1300487400,1300498200,1300942980,1300946340,1301072580,1301076660);
	####outages_app2 = c(1301072580,1301076660,1301444220,1301445780);
	outages_app2 = c(1301444220,1301445780);

	for (fname in filenames) {
		print(paste('Processing file:',fname));
		if (regexpr('.png',fname) > 0) {
			next;
		}
		if (regexpr('.svg',fname) > 0) {
			next;
		}

		if (!file.exists(fname)) {
			print(paste('Specified file does not exist:',fname));
			next;
		}

		header_data = get_col_headers(fname);
		col_headers = header_data[1]
		skiprows = header_data[2]
		if (is.null(col_headers)) {
			next;
		}

		#TODO: remember that the lines in the files are not tab-separated except for the first 3
columns
		#-- the 3rd column is really just everything else

		#Read the file's data
		#fill=TRUE allows it to read lines that are not the right length -- unfortunately, that
means some rows might have bogus data in the later columns
		if (skiprows > 0) {
			print(paste('File (skiprows):',fname));
			allrows  =  read.table(fname,  header=TRUE,  sep='\t',  na.strings='NA',  dec='.',
strip.white=TRUE, blank.lines.skip=TRUE, fill=TRUE);
		} else {
			allrows  =  read.table(fname,  header=FALSE,  sep='\t',  na.strings='NA',  dec='.',
strip.white=TRUE, col.names=col_headers[[1]], blank.lines.skip=TRUE, fill=TRUE);
		}

		if (skiprows > 0) {
			allrows$score = as.double(allrows$score_mean);
		}
		else {
```

```
                allrows$score = as.double(allrows$score);
        }
        #The epoch time for March records was incorrectly converted offset to UTF a second time
(when it was parsed from the date field), so undo that mistake
        if (log_set == 'spring') {
                #print(paste("Correcting        date_epoch        for        allrows$date_epoch;
avg:",mean(allrows$date_epoch)));
                allrows$date_epoch = allrows$date_epoch-utc_offset
                #print(paste("Corrected        date_epoch        for        allrows$date_epoch;
avg:",mean(allrows$date_epoch)));
        }

        #Generate graph from full set
        print(paste('Generating graph from full data set with',length(allrows$score),'rows'));
        pngname = paste(fname,'-allrows-','.png',sep='');
        if (length(grep('app1', fname)) >=1)  {
                #print(outages_app1);
                outage_times = outages_app1;
        } else {
                if (length(grep('app2', fname)) >= 1) {
                        #print(outages_app2);
                        outage_times = outages_app2;
                } else {
                        outage_times = outage_times_fall;
                }
        }
        generate_graph(pngname, allrows, outage_times, fname, FALSE);

        #Generate graph with jitter
        pngname = paste(fname,'-allrows-jitter-','.png',sep='');
        if (length(grep('app1', fname)) >=1)  {
                print(outages_app1);
                outage_times = outages_app1;
        } else {
                if (length(grep('app2', fname)) >= 1) {
                        print(outages_app2);
                        outage_times = outages_app2;
                } else {
                        outage_times = outage_times_fall;
                }
        }
        generate_graph(pngname, allrows, outage_times, fname, TRUE);

}
#[1] '***Processing file: output-final_stage_sysout-files-111210-big-sa-c2-n.scrubbed.csv'
#[1] 'Generating graph from full data set with 39765 rows'
#[1]  'Generated  .png  file:  output-final_stage_sysout-files-111210-big-sa-c2-n.scrubbed.csv-
allrows-.png'
#[1] 'Generating graph from spam-scored data set with 0 rows'
#[1]  'Generated  .png  file:  output-final_stage_sysout-files-111210-big-sa-c2-n.scrubbed.csv-
above3.0-.png'
#Error in plot.window(...) : need finite 'xlim' values
#Calls: generate_graph ... vbox -> plot -> plot.default -> localWindow -> plot.window
#In addition: There were 25 warnings (use warnings() to see them)
#Execution halted
warnings()
```