Log File Filtering with Off-the-shelf Naïve Bayesian Content Filters

Russel W. Havens, Barry Lunt, Chia-Chi Teng
School of Technology
College of Engineering and Technology
Brigham Young University
USA
luntb@byu.edu

ABSTRACT: *As computer systems become more complex, the state of their inner workings become more and more important to the system administrators working to keep them running. Log files provide much needed visibility into these systems, whether they are hardware, operating systems or applications. Unfortunately, systems can easily create overwhelming amounts of data for administrators to comb through. 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 three filters. The 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 log entries from two services, with logs taken from production systems, and 3) the filters were trained with problem-related log entries from system outages and then tested for effectiveness in finding similar outages through similar log files.*

Keywords: Log file Analysis, Bayesian Content Filter, Filter Effectiveness

Received: 13 June 2011, Revised 29 July 2011, Accepted 2 August 2011

1. Introduction

Knowledge workers, accountants, management, sales people, even line workers use computers every day to do their jobs. When critical computer systems crash, these employees often cannot bring in revenue and provide services. This can add up to thousands to millions of dollars in revenue per hour depending on the nature of the outage. Any less downtime per incident can add up to huge savings for some organizations.

Log files written by system components often contain clues pointing to the nature of problems and help administrators to resolve these problems. Unfortunately, even in moderately sized data centers, so many lines of logs are produced that administrators cannot effectively use them. 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. More effective log filtering systems that trim the set to outage-related entries would allow problems to be more quickly resolved, saving trouble, time and money. Thus the question: Is there a widely available, well-understood, advanced filtering and data clustering technology useful for log analysis?

2. Background

2.1 Log Files

The inner workings of the computer are encoded as electrical pulses in a myriad of circuits. Because humans cannot perceive these signals, programmers make those inner workings available to system users through both standard user interfaces and debugging output.

Because displays are already in use with output for end-users, system-internal information is often sent to log fileswhen programmers want to see inside a system. A log of events can provide extremely valuable insights into system operation, not only from visibility of variables and actions, but from visibility to the timing of those variables and actions in relation to one another. Programmers use these log entries to gain insight into potential problems with the code; system administrators use these log entries to guide efforts of system management and troubleshooting.

Generally, each programming project standardizes on a logging framework and a set of conventions for what 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. Unfortunately, even with common methods, log entry styles still vary considerably from program to program.

2.2 Syslog

In the 1980's, Eric Allmandeveloped the syslog logging standard [1]. In addition to a network transport standard, syslog defined entry fields, including severity and logging facility information, a time/date stamp, the source host address or name, the source process name and a message body, called the CONTENT. The CONTENT field contains the specific message logged by the system, in whatever forms the programmer choses, and is notoriously free-formed.

2.3 Syslog Analysis

Syslog is the de facto standard for enterprise logging because of its flexibility and its ease of consolidation (due to its well-defined network transport), syslog is used by many applications, operating systems and devices, in spite of the loose structure of the CONTENT portion of syslog messages. Unfortunately, system administrators are overwhelmed by the sheer volume of loosely structured records.

Each field in the syslog message 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 free-form CONTENT fieldrequires more effort to analyze than other fields, though its payload is most issue-relevant.

2.4 Related Work

Various log analysis techniques have been investigated and tools developed over the last few years. As early as 1996, Doug Hughes [2], described the use of various visualization tools for managing systems and networks, including a simple low/high priority message filter, tklogger.

In 2002, Risto Vaarandi [3] proposed a lightweight event correlation engine, SEC, which used pattern matching rules. In 2003, Vaarandi introduced an enhanced log correlation engine, called SLCT [4].

In 2004, John Stearly [5] described his Sisyphus toolkit, a system to analyze syslogs using a bioinformatics-inspired anomaly-detection algorithm. In 2009, Makanju, et al, [6] presented a technique called IPLoM, or Iterative Partitioning Log Mining, divides the set of log entries through a two or three stage iterative partitioning technique.

In 2008, Wei Xu, et al [7], proposed analyzing a program's source code for all possible log output lines, then simplifying down that data set using Principle Components Analysis (PCA), and then mining console logs for these lines. The source code access and PCA transform knowledge requiredis probably too limiting and too knowledge-heavy to be widely used widely.

Numerous log visualization techniques have been used as well [8][9][10]. However, 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.

2.5 Syslog Analysis Tools and Products

Because syslogs' potential value, many tools have sprung up for analyzing them. Some, like Microsoft's Log Parser [11] are simple log parsers and formatters. Others, like Apache ChainSaw [12] and Octo pussy [13], provide viewers, reporting and alerting as well. Still others, Splunk [14], XpoLog[15], Novell Sentinel Log Manager [16] and LiquidLabs LogScape [17], provide log management, forwarding, viewing, searching, alerting and many other features.

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 largely 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.

2.6 Spam Control

E-mail spam, or unwanted e-mail messages, is a great plague to today's information society. The ITU's 2005 legal analysis of spam law put spam as 88% of e-mail that traverses the Internet [18]. Symantecstated in May of 2009 that over 90% of all e-mails being spam [19].

Great effort has gone into battling the spam problem. There is now a sort of spam arms race, with the see-saw tipping towards the spammers, then the blockers, then the spammers, back and forth. Bayesian spam filters have proven themselves in the anti-spam arena, and areof interest in this research. These filters are quite resilient to variations in text as presented by spammers. This resiliency should also make spam filters useful with log entries, which are generally similar, but rarely exactly the same across machines and over time and software upgrades.

2.7 Bayesian Spam Filtering

In 1998, Mehran Sahami, et al, [20] wrote a seminal article proposing the use of Bayesian probabilistic machine learning techniques, particularly the naïve Bayesian classifier, on the then-new spam classification problem.

In 2002, Paul Graham's [21] similarly influential article, "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.

A Bayesian spam filteruses Bayes' theorem to combine knowledge of previously categorized messages with analysis of incoming messages, categorizing e-mails as spam or "ham" (non-spam messages) [22]. An administrator trains the filter by giving it a certain number of spam, as a spam category, and a similar number of ham, as a ham category. Then the filter compares the words found in each new email to the words in its two categories, combining the probabilities of the new message's words with the probabilities of similar words in the two categories, and determining how likely it is that this new 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' theorem, but all of them are lumped together as Bayesian because of their similar properties. The best known of the open source products is SpamAssassin [23]. Two other well-known filters of this type are SpamBayes [24] and Bogofilter [25].

SpamAssassin is a very rich spam filtering tool which uses multiple techniques to recognize and filter spam from a mail stream. All filters can be enabled as needed. For this research, only its Bayesian filter remained enabled.

The open source Bogofilter, based in part on Graham's work, was started by Eric Raymond in 2002. Bogofilter additionally utilizes a geometric mean algorithm with Fisher's method modification from Gary Robinson, attempting to make the filter more effective.

3. Methodology

Most log analysis tools use very simple keyword searching and filtering engines, the richest using regular expressions. While statistical text analysis tools are available, they are generally either very expensive commercial tools or hard-to-use research tools. However, Bayesian content filtering is very commonly used in widely distributed spam filtering tools.

Each of the three filters used here were evaluated with three different datasets of various characteristics:

1. SpamAssassin Corpus testing: A known spam corpus was run through the filters to verify that the filters work for a known data set. The messages in the set were manipulated variously to make them more similar to log entries.

2. Contrived Log File Testing: Two applications' log entries were pulled from a production system's syslog repository and the filters ability to differentiate these entries from each other was tested. These entries were manipulated variously to potentially increase the effectiveness of the filter.

3. Outage Log Entry Testing:

a. Syslog entries were found to contain outage-related entries at several time frames. One of those time frames was used to train the filters and the rest were tested to determine if the filter could find the outages.

b. Application log entries correlating to another set of outage were tested with the filter to determine if they were related to one another. The logs were then manually searched to determine the effectiveness of the filter.

The output scores of these tests were analyzed statistically and graphically.

4. Results

4.1 SpamAssassin Corpus Testing

SpamAssassin, SpamBayes and Bogofilter were installed and run on a Linux testing platform.

The SpamAssassin public corpus was pulled from the SpamAssassin corpus repository site at Apache.org [27]. It contains 6047 messages, 31% of which are spam.

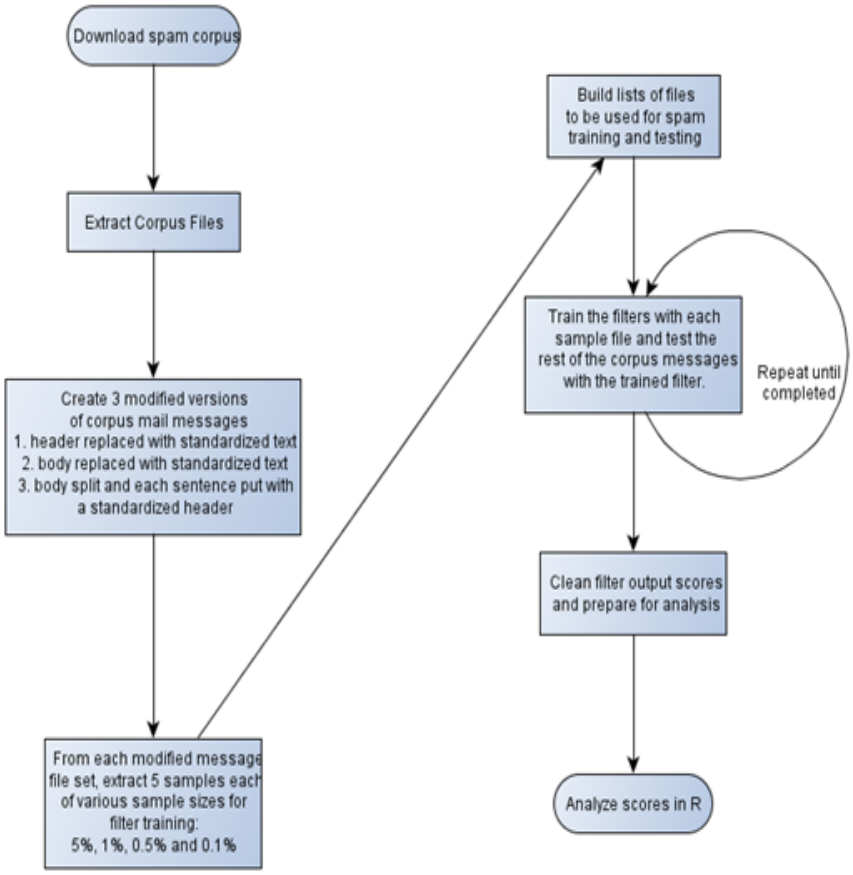

Figure 1. Spam Testing Flowchart

A simple flowchart of the work done in the spam corpus test is shown in Figure 1.

This work was labor-intensive, partially due to the number of variables independently controlled for: filtering tools (SpamAssassin, SpamBayes and Bogofilter), message manipulations (retain header data only, retain body data only, split body sentences into individual messages), sample sizes (10%, 5%, 1%, .1%, .5%), 5 separate random samplings for training, and ham-only vs. spam-only test runs. This made for 450 filter runs.

Summarizing just the accuracy rates of all these runs produced the data in Table 1. Since the actual message types are known, accuracy rates could be calculated directly. The 5 sampled runs for each combination were averaged to reduce and normalize the analyzed data.

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

Table 1. Spam Corpus Testing Results 1

4.2 Contrived Log Entry Testing

For a simplified log entry test, a corpus of syslog data 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, and these lines were used to train the 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 2and 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 against the trained filter.

The following variables were controlled for independently:

1. Three different spam filtering tools

2. Numbers in log entries normalized to zeros or left as-is

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 "Now_is_theis_the_timethe_time_fortime_for_allfor_all_goodall_good_mengood_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.

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 1word chains and "Now_is_the_time_forNow_is_the_timeis_the_time_forNow_is_theis_the_timethe_time_for Now_isis_thethe_timetime_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.

Spam Assassin gives entries a floating 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 "Unsure" in addition to a 0 to 1-scale score.

The flow for this trivial log file testing stage is shown in Figure 4.

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.

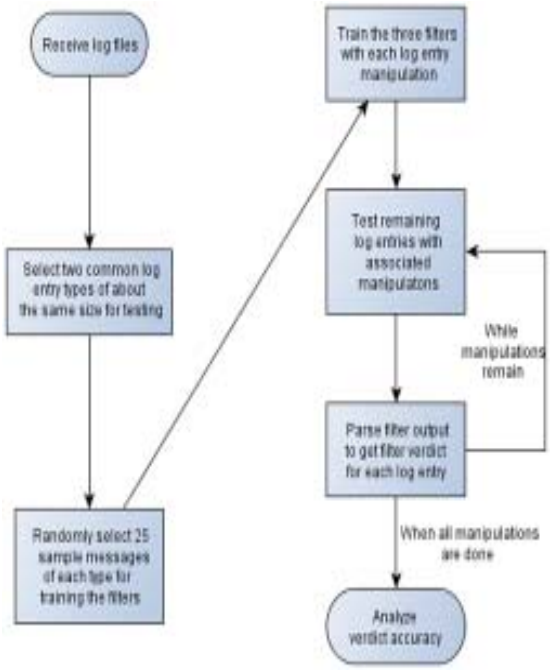

Figure 4. Trivial Log File Testing Flow

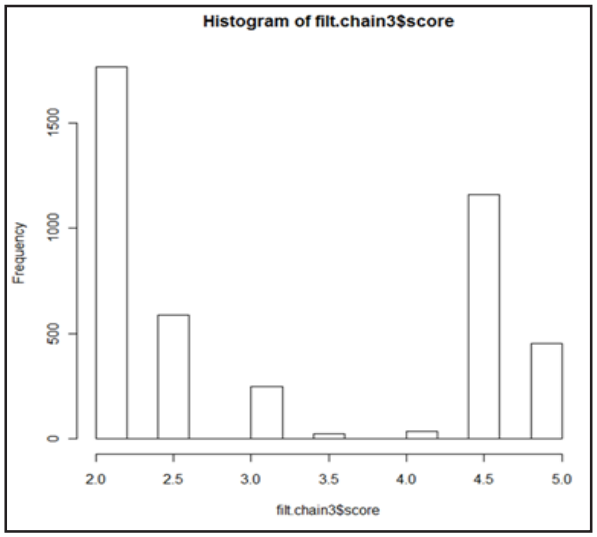

Figure 5. SpamAssassin 3-Chain Score Histogram

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 III shows the actual ham/spam split (i.e. the split of dhclient and kernel messages, per the syslog application field) and Table II 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 III; 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

above 0.5 as likely be spam and Bogofilter records with scores above 0.95 as likely to be spam. Note that although the range of the SpamAssassin scores is nearly fivetimes that of SpamBayes and Bogofilter, its standard deviation is not quite threetimes 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.

Type	Record Count
Ham	197791
Spam	160889

Table 3. Actual Ham/Spam Split

Detected Type	Record Count
Ham	139512
Spam	142796
Unsure	76399

Table 2. Detected Ham/Spam Split

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

Table 3. Filter Score Statistics by Filter

The actual types (as found in the syslog facility field) and discovered types (as detected by the filter) for the records were compared. Two analyses were done with those data: a simple table of successful match rates at given variable levels, and a logistic regression.

Table IV shows the table analysis. For this, the percentage of correctly identified records for each 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% recognition. 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 as 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 V, stacked chains are not significant, while filter tool, chain length and normalized numbers are very significant predictors of the correctness of the filter. This suggests 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 IV, shows that filter type and chain length, in particular, 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%

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 4. Test Results Sorted by Correctness

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

Table 5. Logistic Regression Results

effective in correctly matching the test records to the trained record types, a 13.278% improvement in differentiation over untreated data for this log set.

SpamBayes does best with data that have only had numbers normalized, the manipulation making 4.112% improvement.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.

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 logs from a production application.

The lack of differentiation from chain stacking is likely 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.

4.3 Outage Log Entry Testing and Record Comparison

Finding a corpus of log files for outage log entry testing was difficult, reflecting current industry attitudes towards logs.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, two sets of logs were obtained from the Family History Department (FHD) of the LDS Church which runs the FamilySearch.org web site. This department runs applications which reside on hundreds of servers, many of which log 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, which was reflected in syslog.The entries 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 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.The application 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?"

4.4 Spring Outage Syslog Analysis

The relatedness of the Springoutages was already determined by looking at the logs, so these were used for the first question.

These outages occurred on two systems: three on server app2 and one on server 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 to Mar. 31, and the app1 logs ran from Mar. 13, to Mar. 31.

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 chainlength 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° longitude). 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, 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 until after the work had been done, but not considered a problem for the balance of the filter.

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 visually lost.To assist the graph viewer, the regression line and loess smooth and smoothed conditional spread were also fitted by the graph function chosen (scatterplot from the R "car" package[28]); 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.

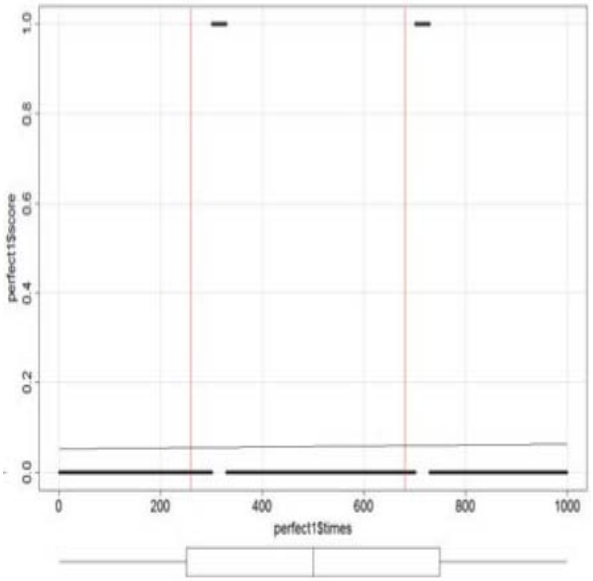

Figure 6. Simulated graph of a perfect filter's scores

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 6, where all data points are either 0 (ham) or 1 (spam), and the distinction is very clear.

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 7. In this example, no clustering is evident at all, which is a characteristic of 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 discriminate is of no use. The second factor becomes obvious as one looks at the major difference between Figure 6 and Figure 7; if scores are all over the gamut, "spam" score patterns would 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 (0-3 representing noisy through clear score patterns).And the implementation difficulty of such a filter might be represented by 2 points (0-2 being difficult through simple to implement and interpret).This scale will be referred to as the Filter Effectiveness Scale (or FES) going forward.

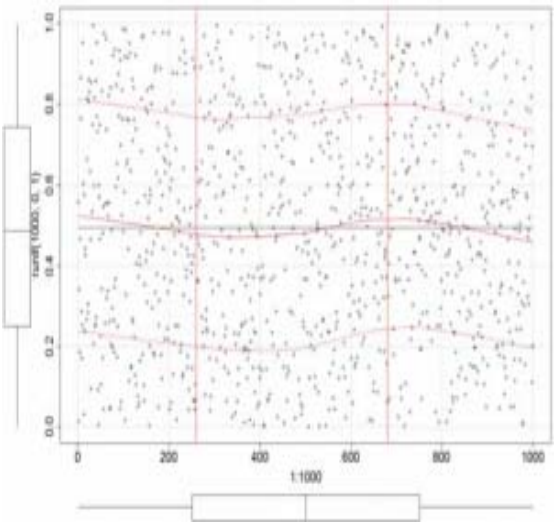

Figure 7. Simulated graph of a perfectly ineffective filter's scores

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 6, and the completely ineffective filter that would generate Figure 7, would receive FES scores of 10 and 0, respectively.

4.5 Scored Spring Log Entries

The 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 easily used in scatterplots and the like, and is less likely to cause parsing problems for the R[28]scripts used for analysis.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.

Known outage start times were marked as vertical red lines on the graphs for reference; blue lines represent the outage end-times.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. Because these are approximate, they are used only to guide the eyes to the graph area where issues occurred.

The graphs from SpamBayes were typical of the three filters, so its graphs are shown here. (All graphs can be seen at www.russelhavens.org). Figure 8 shows log entry filter scores vs. Time for the app1 log and Figure 9 shows the same for the app2 log.

Between the two logs, all four outages have associated vertically-spread datapoints which are very plain in the SpamBayes and Bogofilter graphs; the SpamAssassin graphs showed similarly plain spread data points once the scores were jittered vertically. Jittering proved necessary for SpamAssassin, as the results were so tightly clustered that visually they appeared absent, except for the marginal whisker plots.

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 for those messages, 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 higher 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 plainly correlated with the four outages noted by the vertical red lines, and show several other, earlier, undocumented outages on app2 during the time frame of the graphs.

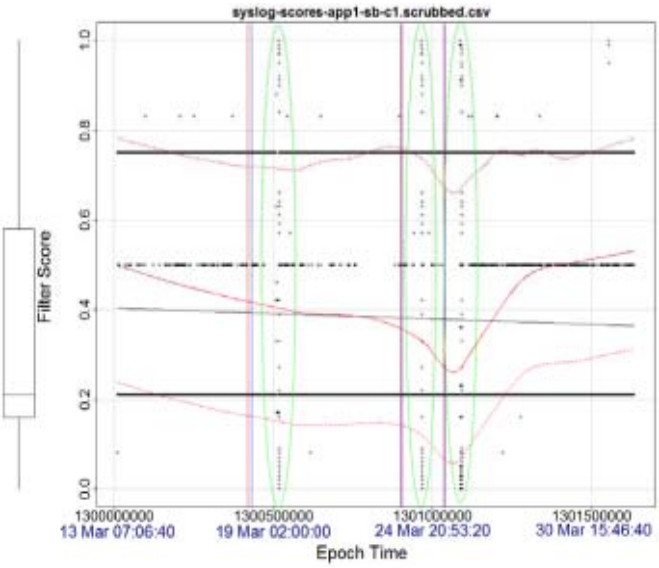

Figure 8. SpamBayes- app1, chain length=1, no normalization

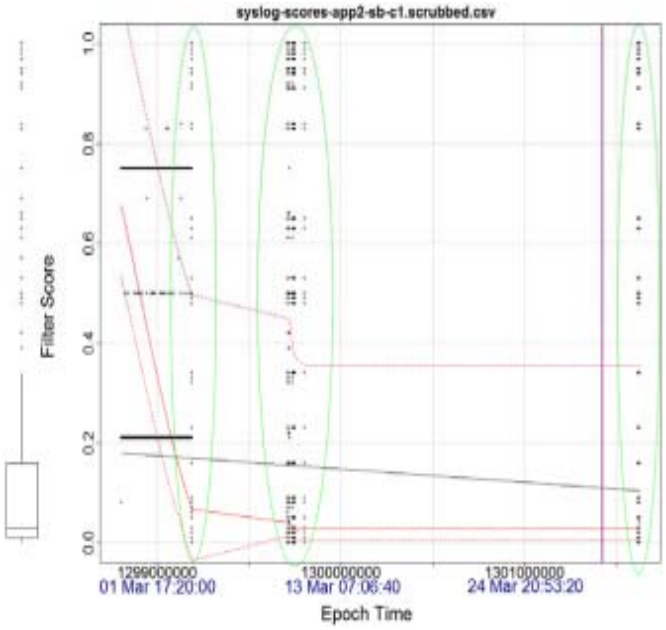

Figure 9. SpamBayes - app2, chain-length1, no normalization

Following the scoring criteria listed above, both SpamBayes and Bogofilter would score 5 for ability to differentiate "spam" records from "ham" records. Once the graph was jittered, SpamAssassin would score the same. All three filters would have a score of 2 for noise,because they do have some scattered score noise, at least more than the ideal graph, Figure 6. All three filters score a 2 for ease of implementation, as they were all about the same difficulty for integration, and all three filters are widely available, free tools, with simple command-line interfaces. Thus all three of these filters would receive an FES score of 9.

4.6 Fall Outage Application 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 huge, so they were trimmed to the hour of the error for two logs and to the hour prior to the outage in question as well as the hour of the outage for the other two logs (as the outages were early in those hours).This was intended to bracket the problem time frames sufficiently. There were 78,779 log lines for the hour of outage of the 12th , 167,449 lines for the hour of and before the outage on the 16th, 79,000lines for the hour of outage on the 18th, and 18,654 lines for the hour of and prior to the outage on the 21st.

The detected outage for the 12th started at 18:48, and had 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 being sensitive to imbalanced training sets, the original 33 lines were removed from the log file and 33 lines were randomly sampled 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. SpamAssassin graphs were jittered to increase the visibility of patterns, as with the Spring data.

After training each filter with these 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 so that a simple numeric scatterplot could be used. Actual dates for the reference epoch time values were added to the graphs to give a better idea of the timeline on the x-axis.

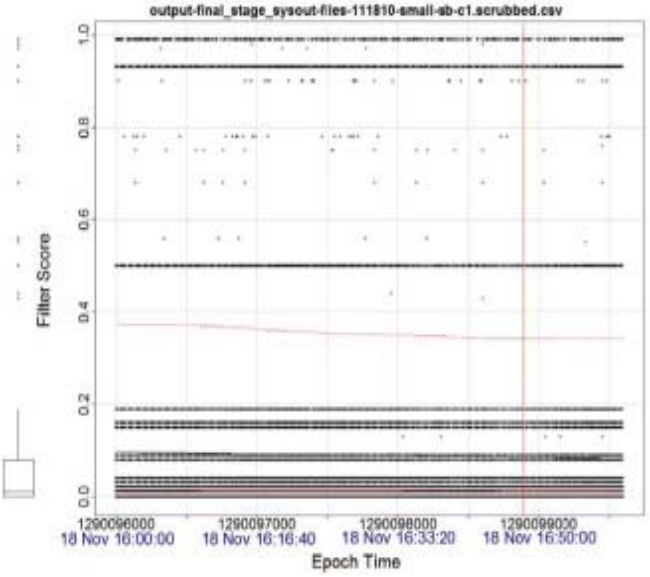

Figure 10. SpamBayes, 11/18

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 of integration).

As with the spring outages, it is interesting to note the differences in the score graphs for the various tools. SpamAssassin's score set showed 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 and Bogofilter, showed 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/Bogofilter pattern, one explanation could be that they take message proximity into account when scoring log entries and that it uses more differentiating factors so that it gets more possible scores.

Using the proposed 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. So, "Is it possible to find a widely available, advanced filtering and data clustering technology that is useful for log analysis?" Yes it is.

5. Conclusion

With the high cost of computer outages, approachable, effective log filtering tools come into a high demand.In this research, three spam filters utilizing Bayesian content filters (SpamAssassin, SpamBayes and Bogofilter) were tested for effectiveness, first with a known spam corpus, then with outage log entries, first with a differentiation test, and then with anoutage log entry test. The effectiveness of these filters was scored on a proposed Filter Effectiveness Scale (FES) for comparison with each other.

Outage-related filter score patterns were clearly related to known outages, even revealing previously unreported outages. As a corollary, unrelated outages were similarly visible in a second, larger set of logs, though the greater number of records for the Fall data significantly increased the noise level of output score graphs.

This sort of statistical text analysis and filtering of log entries is not widely used, and yetshows 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 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.

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.

The premise of using a simple Bayesian content filter for 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 tiny 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.

Even simple statistical text analysis and data mining tools could be a great boon in intelligently filtering log entries, making them easier to analyze. Like its real world counterpart, mining of logs can be much more effective with the right tools. These tools are smaller and more efficient than their real-world counterparts, but they require investment nonetheless. The gold to be found is less tangible but just as real, being made of critical time and knowledge.

References

[1] Lonvick, C. (2010). RFC 3164. n.d. http://www.ietf.org/rfc/rfc3164.txt (accessed March 13).
[2] Hughs, D. (1996). Using visualization in system and network administration, *In: Proceedings of 10th Systems Administration Conference* (LISA'96) 59--66.
[3] Vaarandi, R. (2003). A Data Clustering Algorithm for Mining Patterns from Event Logs, *In:* Proceedings of the 2003 IEEE Workshop on IP Operations and Management,119-126.
[4] Vaarandi, R. (2002). Sec - A Lightweight Event Correlation Tool. IEEE Workshop on IP Operations and Management. 111-115.
[5] Stearly, J. (2004). Towards Informatic Analysis of Syslogs, *In:* Proceedings of IEEE International Conference on Cluster Computing.
[6] Makanju, A.A.O., Zincir-Heywood, A.N., Milios, E.E. (2009). Clustering event logs using iterative partitioning, *In:* Proceedings of the 15th ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, 1255-1264.
[7] Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M. (2008). Mining console logs for large-scale system problem detection, *In:* Proceedings of the Third conference on Tackling computer systems problems with machine learning techniques, USENIX Association.
[8] Aharon, M., Barash, G., Cohen, I., Mordechai, E. (2009). One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs, *In:* Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases.

[9] Takada, T., Koike, H. (2002). Tudumi: Information Visualization System for Monitoring and Auditiong Computer Logs. *In:*Proceedings of the 6th International Conference on Information Visualization.
[10] Hochheiser, H., Schneiderman, B. (2003). 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*.
[11] Microsoft Corp. Download details: Log parser 2.2. n.d. http://www.microsoft.com/downloads/en/details.aspx?familyid=890cd06b-abf8-4c25-91b2-f8d975cf8c07&displaylang=en (accessed 4 17, 2010).
[12] Apache Foundation. Apache Chainsaw.n.d. http://logging.apache.org/chainsaw/index.html (accessed April 17, 2010).
[13] Thebert, S. Octopussy [home]. n.d. http://www.8pussy.org/dokuwiki/doku.php (accessed April 17, 2010).
[14] Splunk Inc. Splunk | IT Search for Log Management, Operations Security and Compliance. Splunk.n.d. http://www.splunk.com/ (accessed March 13, 2010).
[15] XpoLog Ltd. XpoLog log management and log analysis platform. n.d. http://xpolog.com (accessed April 17, 2010).
[16] Novell, Inc. Log Management software | Sentinel Log Manager. n.d. http://www.novell.com/products/sentinel-log-manager/ (accessed April 17, 2010).
[17] Liquidlabs. Logscape.n.d. http://www.liquidlabs-cloud.com/products/logscape.html (accessed April 17, 2010).
[18] Bambauer, D., Palfrey, J., Abrams, D. (2005). A Comparative Analysis of Spam Laws: The Quest for Model Law. ITU WSIS Thematic Meeting on Cybersecurity. Geneva: International Telecommunication Union.
[19] Ragan, S. Spam levels now at 90 percent says Symantec - junk mail arriving like clockwork - Security. The Tech Herald.n.d. 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).
[20] 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.
[21] Graham, P. (2004). Hackers & Painters: Big Ideas from the Computer Age. Beijing: O'Reilly.
[22] Zdziarski, J. A. (2005). Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification. San Francisco, CA, USA: No Starch Press.
[23] Apache Foundation. SpamAssassin - Welcome to SpamAssassin. n.d. http://spamassassin.apache.org/ (accessed June 25, 2011).
[24] SpamBayes. SpamBayes: Bayesian anti-spam classifier written in Python. SpamBayes. n.d. http://spambayes.sourceforge.net/ (accessed February 13, 2010).
[25] Raymond, E. S. Bogofilter home page. n.d. http://bogofilter.sourceforge.net (accessed 4 10, 2010).
[26] Meyer, T. A., Whateley, B. (2004). Spambayes: Effective open-source, Bayesian based, email classification system. Proceedings of the First Conference on Email and Anti-Spam (CEAS), 98.
[27] Apache Foundation. SpamAssassin public corpus readme. SpamAssassin. n.d. http://spamassassin.apache.org/publiccorpus/ (accessed February 13, 2019).
[28] Fox, J. and Weisberg, S. (2011). An {R} Companion to Applied Regression. Thousand Oaks: Sage.
[29] R Development Core Team. R: A Language and Environment for Statistical Computing. n.d. http://www.R-project.org (accessed November 20, 2010).