SEVENTH FRAMEWORK PROGRAMME
FP7-ICT-2009-6

BlogForever
Grant agreement no.: 269963

BlogForever: D2.5 Weblog spam filtering report and associated methodology

Editor:	**Yunhyong Kim, Seamus Ross**
Revision:	**Yunhyong Kim, Seamus Ross, Vangelis Banos, Stella Kopidaki, Karen Stepanyan, Morten Rynning, Nikolaos Kasioumis**
Dissemination Level:	PU
Author(s):	Yunhyong Kim, Tracie Farrell
Due date of deliverable:	29 February 2012
Actual submission date:	29/02/12
Start date of project:	01 March 2011
Duration:	30 months
Lead Beneficiary name:	UG

Abstract: This report is written as a first attempt to define the BlogForever spam detection strategy. It comprises a survey of weblog spam technology and approaches to their detection. While the report was written to help identify possible approaches to spam detection as a component within the BlogForver software, the discussion has been extended to include observations related to the historical, social and practical value of spam, and proposals of other ways of dealing with spam within the repository without necessarily removing them. It contains a general overview of spam types, ready-made anti-spam APIs available for weblogs, possible methods that have been suggested for preventing the introduction of spam into a blog, and research related to spam focusing on those that appear in the weblog context, concluding in a proposal for a spam detection workflow that might form the basis for the spam detection component of the BlogForever software.

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

The **BlogForever** Consortium consists of:

Aristotle University of Thessaloniki (AUTH)	Greece
European Organization for Nuclear Research (CERN)	Switzerland
University of Glasgow (UG)	UK
The University of Warwick (UW)	UK
University of London (UL)	UK
Technische Universitat Berlin (TUB)	Germany
Cyberwatcher	Norway
SRDC Yazilim Arastrirma ve Gelistirme ve Danismanlik Ticaret Limited Sirketi (SRDC)	Turkey
Tero Ltd (Tero)	Greece
Mokono GMBH	Germany
Phaistos SA (Phaistos)	Greece
Altec Software Development S.A. (Altec)	Greece

History

Version	*Date*	*Modification reason*	*Modified by*
0.1	18/08/11	First draft	Yunhyong Kim
0.2	25/08/11	References inserted and structure in place	Yunhyong Kim
0.3	29/08/11	Section images inserted	Yunhyong Kim
0.5	-	Feedback received	From Vangelis Banos, Nikolaos Kasioumis, Stell Kopidaki
0.8	-	Contribution received	From Tracie Farrell
1	13/01/12	Revision	Yunhyong Kim
1.1	24/01/12	First level completion of revision	Yunhyong Kim
1.2	30/01/12	Revision for distribution to all of WP2	Yunhyong Kim
-	31/01/12	Received feedback from Vangelis Banos (AUTH) and Karen Stepanyan (UW)	-
1.5	31/01/12	Final revision for final draft for distribution within WP2	Yunhyong Kim
2	02/13/12	Corrections made after feedback received from AUTH (Vangelis Banos), Phaistos (Stella Kopidaki), CERN (Nikolaos Kasioumis), Cyberwatcher (Morten Rynning). Also change in title.	Yunhyong Kim
2.1	02/27/12	To reflect concerns expressed at the project meeting in Berlin (22-23 Feb 2012)	Yunhyong Kim

Table of Contents

TABLE OF CONTENTS 4
EXECUTIVE SUMMARY 5
1 INTRODUCTION 7
2 WEB SPAM TYPES 11
2.1 Content spam 13
2.2 Spam blogs (Splogs) 15
3 READY-MADE ANTI-SPAM TOOLS 16
4 PREVENTIVE METHODS FOR FILTERING SPAM 18
5 SPAM DETECTION APPROACHES 19
5.1 Stages of detection 20
5.2 Type of features 24
5.3 Filtering spam adaptively 27
5.4 Viability feature types mentioned in the literature 28
6 RECOMMENDED SPAM DETECTION STRATEGY 29
6.1 Spam detection strategy and the BlogForever spider prototype 32
6.2 Blogforever spam detection methodology 34
6.3 Other means of handling spam 41
7 CONCLUSIONS 42
8 REFERENCES 43
A. APPENDIX A. ANTI-SPAM API AND EXAMPLE USE CASES 47
B. APPENDIX B. BAYES AND SVM-LIGHT FOR PYTHON 55

Executive Summary

This report was written as a survey of web spam filtering methods that might be incorporated into the BlogForever blog archiving software as a component designed to remove material deemed unsuitable for inclusion into a blog archive. Spam comprises a significant proportion of the information on the web. For, example, the Spam Clock[1] claims that there are one million new spam pages created every hour. Given that 300 million websites were estimated to have been added in 2011[2], this is an impressive number should it be true. In the case of spam that arrive at weblogs, over 80% of comments submitted to blog spam filtering services (such as Akismet[3]) for assessment have been classified as spam.

The large volumes of spam form a formidable body of noise for a web archive targeting specified content, and, further, poses technical difficulties for the web spider in charge of discriminating between material relevant to the archive and that which is not. The discussion in this report highlights a range of approaches that have been used in different circumstances to detect spam on the web. In particular, we focus on those methods concerned with isolating blog related spam.

The promising features of different approaches have been aggregated in a multi-layered workflow as a proposal for the architecture for the BlogForever spam detection strategy. The first of this layer (Section 6.2.1) relies on matching strategies using the Unique Resource Locator (URL) and Internet Protocol (IP) address, blog identification, and on applying a range of ready-made anti-spam application platform interfaces (APIs). The second of these layers (Section 6.2.2) adopts a statistical machine learning strategy based on base classifiers built on simple content, link and temporal features. A third layer using implicit relevance feedback (Section 6.3.3) from end-user activity (number of visitations, downloads, clicks and queries) and reported spam (Section 6.3.2) to improve search ranking algorithms and feed suspected URLs and content back to the first two layers is also mentioned. It is also emphasised here that, ideally, it is recommended that the three layers of detection should be preceded by a directed spidering strategy (Section 6.3.1) designed to harvest a reduced number of blogs deemed truly relevant to the repository objectives which would immediately reduce the demand on the URL management capacity of the weblog spider and spam detector.

The work in this report was carried out to meet the practical demands of the BlogForever archive software for a spam removal strategy. However, some of the discussions in this report will show that spam is actually a valuable cultural artefact that tells a story about how human information technology and space has developed into what it is now. In fact, recently there have been discussions about spam culture and how attitudes towards different forms of spam differ according to the country in which you reside[4]. We build technology to avoid spam (if only by creating tools to find “truly relevant information”) and spam is created to outsmart these technologies. In effect, socially, historically, and technologically spam is a cultural heritage of our time. As such it is unclear whether we should be removing this part of our history for the immediate convenience of what we think is valuable with respect to our own cultural standards.

The introduction (Chapter 1) discusses the history and value of spam a bit further, followed by a discussion of web spam and blog spam types (Chapter 2). The overview of ready-made anti-spam APIs are presented in Chapter 3 and other methods recommended for the prevention of spam entering into blogs are mentioned in Chapter 4. A description of the research landscape in the area

1 A application created by blekko.com reportedly tracking spam. http://www.spamclock.com
2 http://royal.pingdom.com/2012/01/17/internet-2011-in-numbers/
3 http://www.akismet.com
4 http://blogs.computerworld.com/14830/spam_culture_part_1_china

of spam filtering is given in the next chapter (Chapter 5). The proposed spam detection strategy for BlogForever is presented in Chapter 6. The document concludes with a summary of the findings and few additional observations in Chapter 7. Where appropriate, we have also included in the Appendix, usage examples of the suggested anti-spam APIs (Appendix A) and statistical packages related to machine learning approaches that might serve as a reference in adaptive methodologies (Appendix B).

1 Introduction

Current archiving standards often place some emphasis on selection and appraisal, that is, selecting material to be included in the archive and "evaluating records to determine which are to be retained as archives, which are to be kept for specified periods and which are to be destroyed"[5]. Those archiving information from the web (e.g. the LiWA project[6]) have also become increasingly interested in selective archiving practices. This has led to proposals for web archiving practices that incorporate methods that might be able to detect and remove noise. One such perceived noise that has been singled out in the web archiving context is spam.

Although the association between spam and electronic communication has now become inseparable, spam, as "unsolicited bulk messages sent out indiscriminately", already existed in 1864[7] when a dentist used the telegram as a medium of advertisement to promote his new practice. In fact, unsolicited mail, advertising products, appealing for charities, promoting causes, campaigning for candidates, and, leading the reader away from the truth, arrive at our door everyday. The electronic communication systems that now allow users to generate and disseminate content easily (e.g. through email, websites, wikis, blogs, and social networks) have merely facilitated the invasion of unsolicited messages, irrelevant search engine results, and misleading data and/or information, into our *information space*, on a much larger scale.

Before the heavy use of web search engines, spam was considered to be mostly related to email messages. email spam is still a prolific form of spam: Message Labs monthly report, in May 2011, estimated 75.8% of emails globally to be spam[8]. However, the predominance of unsolicited, irrelevant content is now as visible within general web content as it is in email. For example, two of the leading anti-spam services for blogs, Akismet[9] and Mollom[10], report that, respectively, 83% and 90% of blog comments examined by them were predicted to be spam. In fact, in the context of web search, spam has come to mean more than unsolicited, irrelevant content to attract user responses. Gyöngyi & Garcia-Molina (2005) define web spam as comprising "any deliberate human action that is meant to improve a site's ranking without changing the site's true value". Whereas, on email, the aim is to fool and elicit a response from the recipients themselves, on the web, the intention, often, extends further to fooling the search engine to raise the rank of target pages (a.k.a. spamdexing[11]) and increase the number of visits.

This is done for a variety of reasons, for example, to promote pages that cause malware to be installed on your local computer when it is visited, to market products, services and affiliated pages, and to increase their revenue from advertisers that finance the site (e.g. see discussion in Egele et al. 2011). This leads to further polluted information generated by automated queries (submitted as part of the process to plagiarise highly ranked content), misleading tags (to promote or demote a page), click-frauds (to cause financial damage to advertisers without corresponding profit) and undeserving product reviews motivated by self interest (see discussions in Heymann 2007; Duskin & Feitelson 2009; Haddadi 2010; Catillo & Davison 2011).

It has been estimated by Kolari (2007) that 75% of pings received at ping servers are spings (that is, pings sent out by non-blogs or spam blogs created for the purpose of promoting affiliated websites). The same study predicts approximately 88% or more of the URLS received at the ping server to be

5 Ellis, J. (1993) (ed.). "Keeping Archives" 2nd edn (Melbourne: Australian Society of Archivists) p.461.
6 http://www.liwa-project.eu/
7 http://www.economist.com/node/10286400/
8 http://www.symanteccloud.com/mlireport/MLI_2011_05_May_FINAL-en.pdf
9 http://akismet.com/
10 http://mollom.com/
11 http://en.wikipedia.org/wiki/Spamdexing

non-blogs or spam blogs. Further, approximately 20-25% of blog search engine (e.g. Technorati [12]) results have been estimated to be spam blogs. The prevalence of spam has led Google to take action by offering the option to webpage managers (for example, blog owners and blog software providers) for including a “nofollow” attribute with respect to selected content, that Google will honour in calculating the page rank, to discourage spammers from submitting spam (e.g. see Marks & Celik 2011).

There have been several recent surveys on web spam which are listed in Figure1.1. The survey by Castillo & Davis (2011) on adversarial web search is the most comprehensive, covering an extensive array of topics related to web spam in general, while the surveys by Mishne (2007) and Kolari (2007) mostly limit their discussion to comment spam and spam blogs, respectively. To gain a fuller picture of the research area, this report should be considered in conjunction with these other surveys.

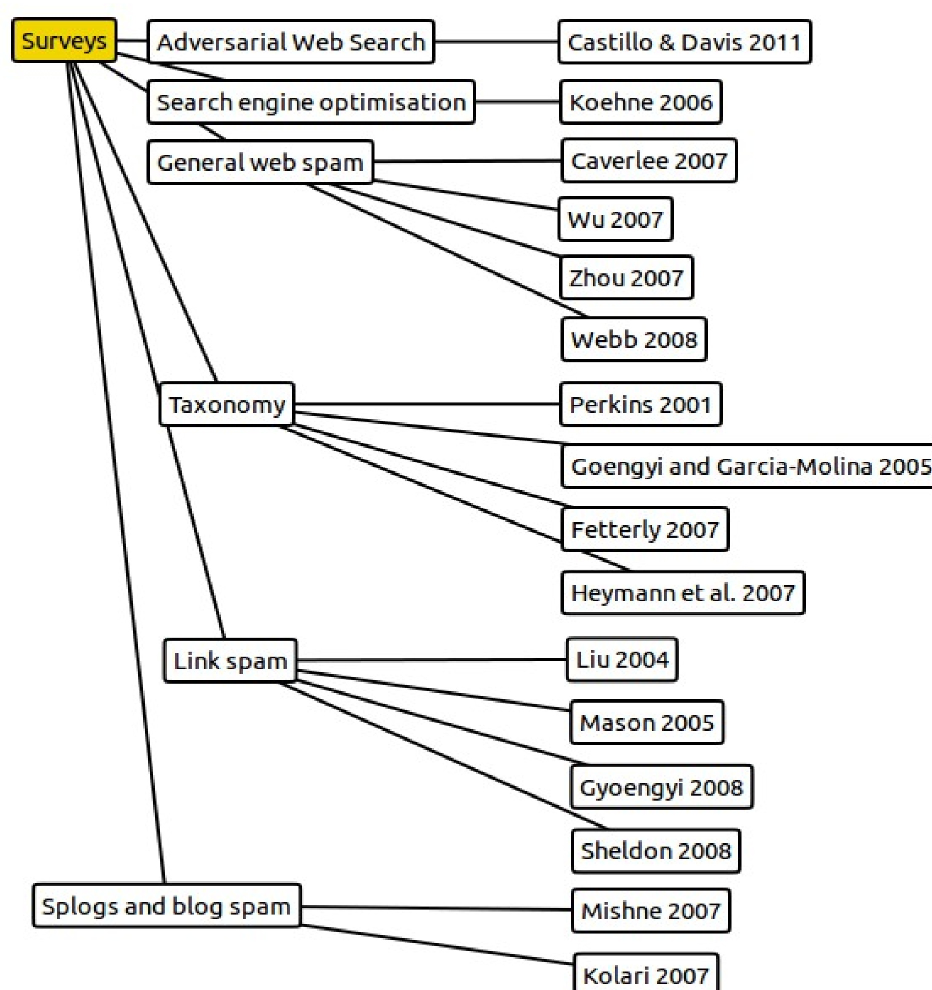

Figure 1.1 Mindmap snapshot of latest spam survey resources

Note that, while the above discussion clearly demonstrates that spam is a formidable obstacle in everyday search and exploration of information, from a web archival perspective, it is not clear that this should lead to the conclusion that we must remove the spam from archival holdings. This would depend on the archival objectives. For example, from a forensics or record keeping point of view, spam could hold the key to vital evidence in a criminal investigation (for example, some use

12 http://technorati.com/

spam to conduct fraudulent activity). Likewise, from a cultural and social studies point of view, spam is an essential part of our culture[13]. The examination of spam could provide clues to understanding their long term effects, on technological development and how we relate to information, because spamming techniques are developed to undermine search algorithms and search algorithms are, in turn, developed to avoid spam (this is why Castillo and Davison (2011) use the term "adversarial search"[14]). The history of changes in spam is the story of our technological achievements. Another more immediate reason to retain spam is the undeniable reality that they are the very examples and evidences of our failure to provide automated approaches to effective search systems that find "truly relevant information". They provide feedback to support the improvement of our information systems.

Regardless of the policies regarding their retention, their detection is undoubtedly valuable within the contexts mentioned above. As such, an efficient method for the detection of spam is likely to be an invaluable addition to a web archive. We propose to investigate here the spam detection methods available and implementable as part of the BlogForever weblog archive. The proposal made here is based on the following principles:

- consistent quality of service within the archive
- simplicity with respect to implementation
- adaptability of the method to new spam and technology
- compatibility with constraints within the repository

By *consistent quality of service*, it is intended that the spam detection method proposed here is designed with an effort to avoid sudden fluctuations of performance in the future. As such, if third party tools and APIs (details of which are closed to the archive) are employed there should be, in conjunction, a backup plan for the sudden unavailability of these tools or if further development of these tool become inactive. *Simplicity* not only lowers the effort of the repository with respect to implementation, maintenance and adaptation, but also reduces processing time, taking into consideration scalability (as the archive grows), and minimises the probability of crossing end-user's tolerance level, should any steps be carried out in real time. The *method's ability to adapt to emerging spamming techniques* is also essential: spam is designed to undermine ongoing developments in search technology, and, to sustain the archive's ability to deliver relevant content, any spam detection mechanisms should ideally maintain the initial performance level. All of this might still need to be governed by *practical constraints imposed by the resources available* (both technical and human) within the repository, and compatibility with other best practice recommendations of the archive.

In line with the BlogForever project Description of Work, we have focused on identifying anti-spam methods that might constitute an efficient approach to the detection of three types of spam specific to blogs[15]:

- splogs, i.e. blogs that exist to promote affiliated websites, by influencing users to visit a webpage or buy a product, as well as spamdexing to undeservedly improve the ranking of a page in a web search by plagiarising content, stuffing keywords or creating large number of links,
- blog comments that contain abusive content or are irrelevant to the original post, and,
- fraudulent pings from non-blogs and/or splogs to attract visitors by misrepresenting content as fresh.

13 http://blogs.computerworld.com/14830/spam_culture_part_1_china

14 A term borrowed from Game Theory: http://en.wikipedia.org/wiki/Talk%3ACombinatorial_game_theory#Adversarial_search

15 As outlined in Part B of the BlogForever Description of Work, Work Package 2 Deliverable 2.5 details.

In the next sections these spam types have been put into the context of general web spam (Chapter 2). This will be followed by a review of ready-made weblog anti-spam services for blogs that are already available (Chapter 3). For a complete discussion, a summary of methods for preventing the introduction of spam into blogs, recommended by blog service providers, have been included (Chapter 4). However, most of these methods involve direct interaction with users of the blog, a communication channel not usually available at the archive. Current research in web spam detection is presented in Chapter 5. This overview is weighted by an emphasis on those related to blogs. In Chapter 6, the discussion is reflected in a proposal for a weblog spam detection strategy that might be reasonably incorporated into the BlogForever archive framework. The feasibility of its implementation, however, still needs to be tested within the course of Work Package 4. In Chapter 7, we summarise our findings and make some final observations. We have also tried to include examples of codes in the appendix, where appropriate, for illustrative purposes.

2 Web spam Types

Email spam is intended to elicit a direct response from the recipients themselves (e.g. leading them to buy products, click on links, pass out information). While this is not outside the scope of web spam objectives, web spam is often also constructed with the aim to fool the web search engines (e.g. so that the search engine will rank the target pages higher in the returned list, rank other pages lower in the list, create misleading associations, and lead to undeserving costs and profits). This could be put into effect by generating links (e.g. link farms) that affect link-based ranking scores (e.g. Google's PageRank[16]), manipulating content (e.g. embedding spam in pages containing plagiarised content form highly ranked pages), cloaking parts of the page to supply independent content depending on the type of client requesting the page (e.g. whether it is a crawler or browser request), automated generation of user logs and data (e.g. a large number of the same site address in logs lead to a misrepresentation of site popularity, increased site visits, and raises the rank of the website), and using pings to mask content as being fresh and attract visitors. The taxonomy of different types of spam is summarised in Figure 2.1.

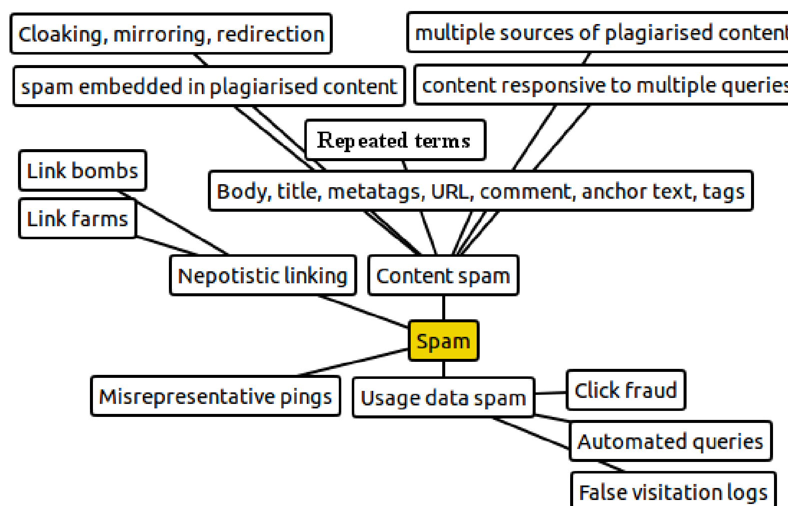

Figure 2.1 Mindmap snapshot of general web spam taxonomy

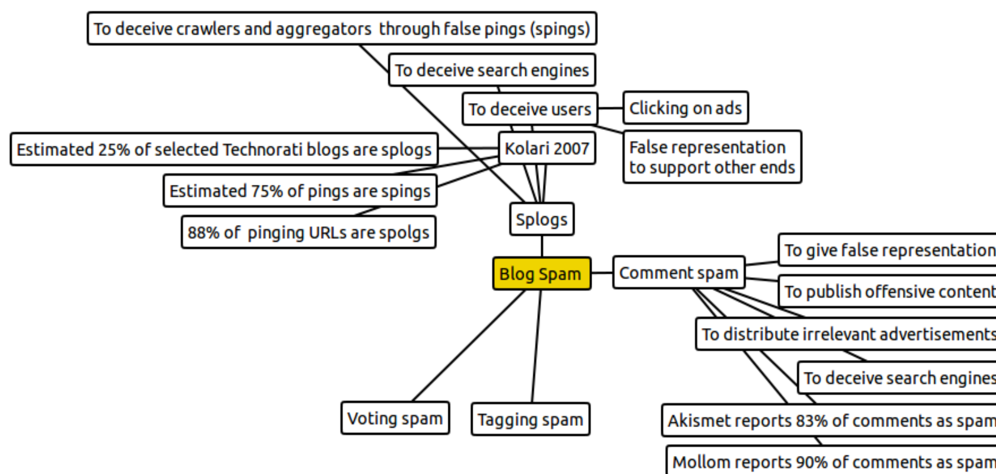

Figure 2.2 Mindmap snapshot of blog spam taxonomy

16 http://en.wikipedia.org/wiki/PageRank

In the context of blogs, content spam is largely dominated by comment spam, where spammers take advantage of the commenting system to market products, post links, and propagate misleading information.

In addition to comment spam, however, spam can take the form of splogs. These are websites that exist to promote affiliated sites by increasing user visits to the target site and raise the rank of target sites with respect to web services such as search engines. While splogs can also contain comment spam, these blogs are more frequently used to create link farms, a large number nepotistic links to manipulate search algorithms that associate importance of a website with its in-links, raising the rank of target websites. There are many splogs that use ping servers to simulate fresh content to attract visitors, sometimes even initiating the installation of malware (e.g. spyware) when it successfully attracts a visitor. Splogs are different from other web spam (Lin et al. 2008) regarding two main aspects:

1. Blogs are highly volatile and unlike the regular web where the content is relatively static, a blog continuously generates fresh content.
2. Hyperlinks are often interpreted as an endorsement of other pages. It is less likely that a web spam gets endorsements from normal sites. However, since spammers can create hyperlinks using comment links or trackbacks in normal blogs, all links cannot be treated as endorsements.

Because of these two significant differences, the splog problem is vastly different from that of traditional web spam.

There are other forms of spamming that are polluting voting system (e.g. the “like” button) statistics and automatically generated user data (e.g. automated generation of tags to obfuscate algorithms that use tags to improve relevance judgements). There is extensive research in all of these areas (e.g. see [9]): spamming techniques tend to employ automatic generation of blogs, comments, and user data, the statistical behaviour patterns of automatically generated information has been shown to be different from that of user generated information. For example, genuine usage data and automatically generated usage data exhibit distinct statistical patterns ([9]). The taxonomy of blog spam discussed in this report is summarised in Figure 2.2.

2.1 Content spam

There have been no shortage of proposals for web content spam detection: for example, people have suggested using topic detection (e.g. Blei et al. 2003; Bìrò 2009), host classification (Fetterly et al. 2004), cloaking detection (Chellapila & Chickering 2006), user query statistics (Ntoulas et al. 2006), syntactic models such as part-of-speech n-grams (Piskorski et al. 2008), term distance model (Attenberg & Suel 2008), and language model comparison (Mishne 2005) [see Figure 2.3].

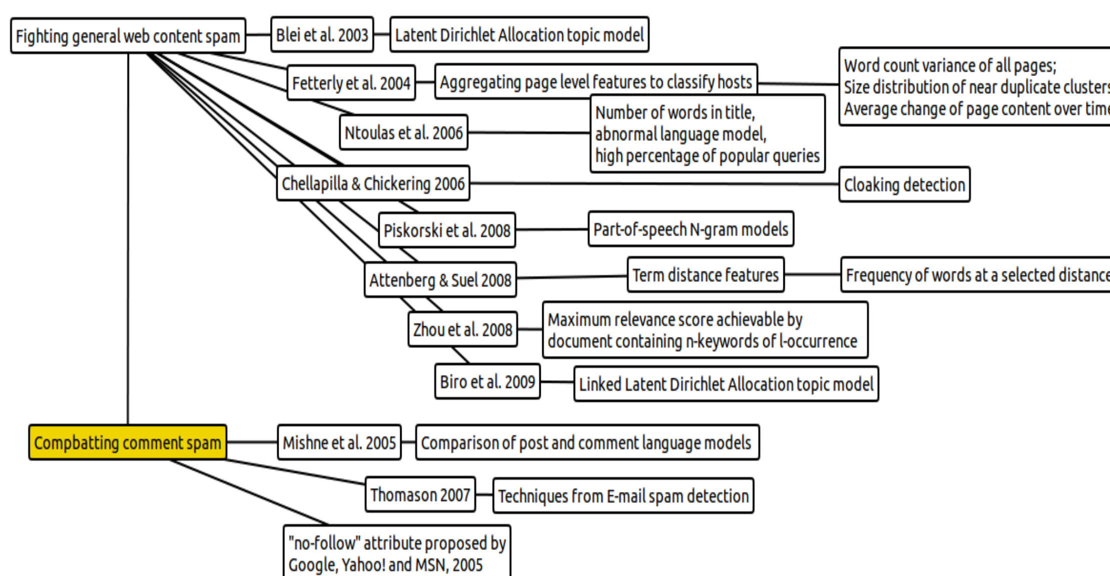

Figure 2.3 Mindmap snapshot of methods for combatting comment spam

There are characterisations of comment spam in terms of their content, e.g. comment-post similarity, frequency of nouns, redundancy of words, anchor text frequency and stop word ratio (Bhattarai, Rus, Dasgupta 2009[17]), which have been used to propose supervised and semi-supervised classification engines. The difficulty of spam is that as soon as an approach to eradicate spam is devised, better spamming techniques will be developed in parallel.

Nevertheless most of the identified features have been incorporated into spam filtering methods and/or services (e.g. Akismet[18], Mollom[19], Defensio[20], and TypePad Anti-Spam[21]). While the inference engines are sometimes published, the rules that are used are often hidden to prevent the emergence of new spamming techniques that target these rules. However, the services are often available for private use as an API that detects spam on the basis of submitted content.

The detection methods developed often work at different stages of the repository workflow: e.g. submission, indexing, and ranking (more detail in Chapter 5). While ready made solutions are convenient, they are generally limited to content spam (i.e. does not address splogs and spings – see Chapter 3). They are also only externally adaptable, i.e. the repository has no control over the

17 http://issrl.cs.memphis.edu/files/papers/blog-spam_IEEE-SSCI-09.pdf
18 http://akismet.com/
19 http://mollom.com/
20 http://www.defensio.com/
21 http://antispam.typepad.com/

source, and, therefore, cannot easily change it internally to incorporate new methods for spam detection that become available).

2.2 Spam blogs (Splogs)

In Section 2.1, we visited the area of content spam. In this section we turn our attention to Spam blogs, a. k. a. Splogs, that are created solely for the purpose of promoting affiliated sites. Lin et al (2008) observed temporal behaviour that distinguish such splogs, e.g. they noted that links in splogs vary little over time, they are characterised by very narrow or very broad topics with respect to their content, and they tend to be updated regularly at very precise times. Sato et al. (2008) also noticed temporal features such as life span of keyword in splogs which was found to be very long lived or very short lived.

Urvoy et al. (2008) found that web pages could be grouped according to stylistic similarity based on HTML templates, which, in turn, can be used to infer whether or not a page is a spam by association. Some have also tried to detect nepotistic links by measuring the similarity between source and target document (Davison 2000; Benczúr et al. 2006; Qi et al. 2007; Martinez-Romo & Araujo 2009). This is comparable to Mishne 2007 who compared the language model of post to that of comment to detect comment spam.

Kolari (2007) developed a meta-ping system to tag triples (name, url, time-stamp) with a score of *legitimacy*. He presented a comparative study of several previously identified features for spam filtering (e.g. word grams, ratios). However, his contribution to spam filtering was in proposing a multilevel approach to be applied to several stages (e.g. before fetching the page and after fetching the page), and proposing the use of ensemble base learners to implement an adaptive spam filtering technique in an adversarial search context.

An overview of the research landscape described above is captured as a mindmap in Figure 2.4.

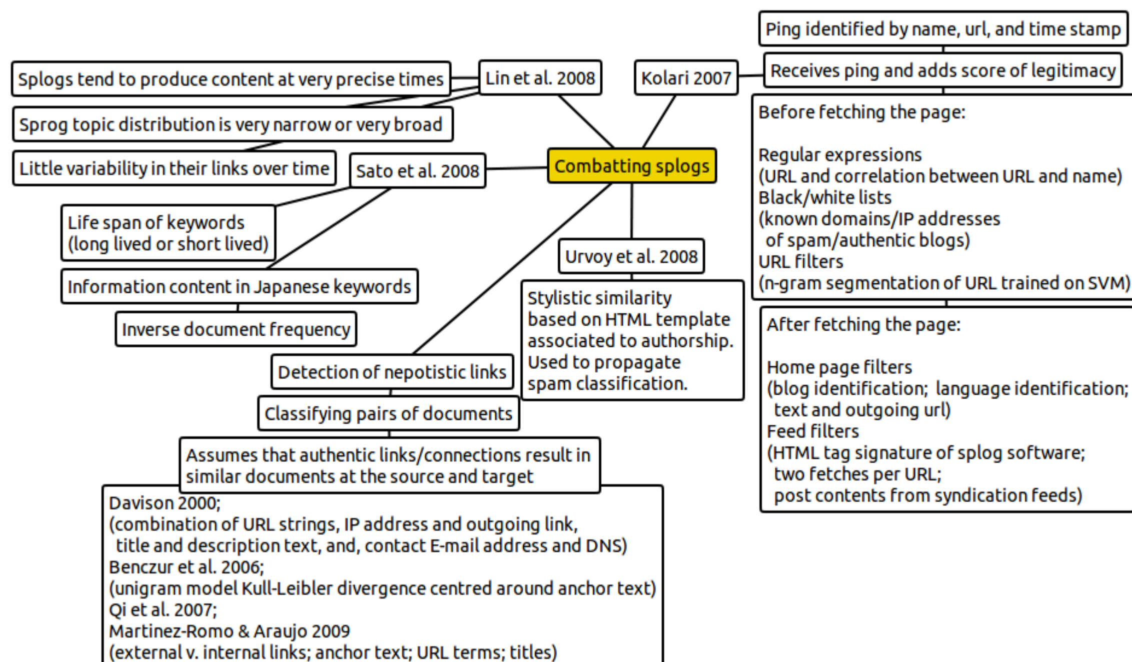

Figure 2.4 Mindmap snapshot of methods for combatting splogs

3 Ready-made anti-spam tools

Currently, there are several widely adopted spam fighting services. Best known anti-spam services for blog platforms are:

Bad Behaviour (http://bad-behavior.ioerror.us/)
Spam Karma (http://unknowngenius.com/blog/wordpress/spam-karma/)
Akismet (http://akismet.com/)
Defensio (http://www.defensio.com/)
Mollom (http://mollom.com/)
TypePad Anti-Spam (TAPAS, http://antispam.typepad.com/)

The first two, while it is still widely used within many circles, have now ceased further development. Here, we will consider the four latter applications more carefully. There isn't much information regarding the inner workings of any of these applications, with good reason, as spammers would take advantage of the information to try to get through. TypePad Anti-Spam (TAPAS) is the only one of these four that provides open access to their inference engine (rules are kept hidden). While some of these remove suspect comments even before moderation by the blog owner (e.g. Mollom), others offer the owner the opportunity to provide feedback, exposing false positives to the user. In the case of Defensio, the user is also provided with a *spaminess* (the estimated likelihood that a message is spam) score, which could be informative.

Key characteristics of these tools are summarised in Table 3.1.

Table 3.1 Comparison of Anti-Spam Plugin Software

	Akismet	Defensio	Mollom	TAPAS
Company	Automatic	Websense	Mollom	Six Apart
When in doubt challenge with CAPTCHA?	No	No	Yes	No
Own API?	Yes	Yes	Yes	Uses Akismet API
Open source engine?	No	No	No	Yes
Free for personal use?	Yes	Yes	Yes (limited volume and features)	Yes
Free for commercial use?	No ($5/month for problogger earning more than $500/month; $50/month for enterprise)	Yes (limited traffic)	Yes (limited volume and features)	Yes

The APIs discussed here along with a few other APIs and libraries are proposed as a first stage spam detection step within the BlogForever spam detection strategy (see discussion in Section 6.2.1). There are some concerns in relying solely on these APIs. We will discuss these concerns further at the beginning of Chapter 6 which describes the recommended spam detection strategy.

In addition to the above services, SplogSpot (http://splogspot.com/) provides a splog search engine, which may, unlike these other services, be able to go beyond comment spam, and help determine site level legitimacy. SplogSpot provides an API that allows access to the database of splogs.

4 Preventive methods for filtering spam

Even before the new comments or blogs are accepted into the blogosphere several preventive mechanisms can be put into place to discourage their publication. Among these are:

- Turing test
- Throttling user actions (allow limited number or time period for comment submission)
- Regulating comments to old posts
- Software update
- Authentication
- Obfuscating comment script
- Add new required fields for comments
- Use of spam word and black/grey/white list databases

The assumption that motivates the use of Turing tests (e.g. Captcha[22] and ReCaptcha[23]) is the notion that spam is mostly created automatically, i.e. not by a human being. The same notion motivates throttling repeated actions (e.g. putting a limit on absolute number of action and rate of repeated actions), discontinuing comments to old posts (i.e. limited time for actions), updating the blog or commenting system software so that automated scripts are forced to deal with changing circumstances and different security protocols, and, authenticating login, obfuscating comment script, and having a variable set of required fields for submitted comments.

These methods, while somewhat successful in reducing the number of spam, have been criticised for discouraging users with legitimate posts. Captchas have been criticised for discouraging interaction from the visually and/or aurally impaired persons. Likewise other forms of preventive methods (e.g. authentication) require users to share information, which many are reluctant to do. Adding required fields also discourages communication by increasing the labour of submitting information.

A more immediate concern, however, is whether these approaches are applicable within the context of a crawler of an archive outside of the context of real-time blogging. For example, at the stage of crawling the post and/or blog the archive may not have access to the user to ask them to complete a Turing test, and even if such information were to be available, it is unclear that it is appropriate for the archive to ask the user for information, since there was no explicit request from that user that the information they submitted be included in an archive. Throttling, regulating, and authenticating actions are also primarily options available for blog owners implement as a process at the time of submission. Software update, obfuscating scripts, as well as, adding required fields are only applicable within the context of the blog platform through which the user submitted the information.

The only reasonable preventive methods from the list above is the use of databases compiling spam words, and black/grey/white lists of IP addresses and URLs. As mentioned in Chapter 3, the use of such database carries some preservation, sustainability and adaptability risks, especially when the list is managed by a third party. Note that black lists refer to identified splogs, white lists refer to legitimate blogs, while grey lists refer to temporarily rejected posts/blogs set aside for verification.

In Chapter 5 we will introduce some more sophisticated filtering methods that have been employed at the time of crawling for blogs (Section 5.1). These can be considered preventive methods as well, while being more likely to be applicable to a blog archive crawler.

22 http://en.wikipedia.org/wiki/CAPTCHA
23 http://en.wikipedia.org/wiki/ReCAPTCHA

5 Spam detection approaches

Spam detection approaches can be examined from several perspectives. On one hand, there are approaches that are tailored to handle specific spam types (Chapter 2). On the other hand, spam detection can be categorised by the timing of that the detection takes place, or by the type of features that the detection algorithm incorporates.

In this section we look at the latter two perspectives. In particular, we give a brief summary of methods that might be employed at the time of crawling (Section 5.1.1), at the time of indexing (Section 5.1.2), and the time of ranking the results of searching a collection (Section 5.1.3). Then, in Section 5.2, we will discuss different types of features, divided into spatial (i.e. features associated with the webpages at a fixed point in time) and temporal properties (i.e. changes that occur over time). The spatial features are further broken down in terms of the cost of their extraction (i.e. how much of the content has to be extracted and processed to obtain the information).

In this section, we also address the question of adaptability, as spam detection that is effective only now incurs additional cost in the long term, while an evolving and/or adaptive framework would reduce the cost in the long term. We feel that it is mandatory for the prototype spam filtering approach within an archival context to be adaptive to new spamming techniques. This is crucial for efficient and effective managing, quality maintenance, and preservation of the web archive material.

5.1 Stages of detection

5.1.1 At the time of crawling

The earlier stages of the framework described by Kolari (2007) (Section 2.2) are spam detection method using features available at the time of crawling and harvesting pages. There is no serious indexing of the content used at this point. Features used will tend to be local features found with the target blog. While ready made anti-spam APIs (Chapter 3) can be used at the time of crawling, the methods they employ could be based on information they have obtained from indexed content stored elsewhere.

Ma et al. (2009) have proposed further filtering methods based on an analysis of IP addresses, and geographic information, and Webb et al. (2008) have used information found within the HTTP response, e.g. IP address to website ratio and identification of software being used, to single out suspect sources. While the question of quality is distinct from *spamicity* (likelihood that a message is spam – similar to spaminess discussed in Chapter 3), Castillo (2004) observes that prioritising high quality sites can be helpful. Some have also remarked that the avoidance of crawler traps can aid spam filtering [Lee et al. (2009)].

Within the context of email spam, P2P collaborative filtering has also been suggested[24], however, the extent of this sort of implementation and their effectiveness within the community of weblog providers and users is unclear. A more common method of filtering would be a direct use of a IP blacklist such as that found at Blog Spam Blacklist (http://blogspambl.com/) or databases of spam pages such as that available at SplogSpot (http://splogspot.com/). A continuous survey of other lists of this kind would be desirable to update the database on a regular basis.

Other methods may include identifying URL patterns (e.g. see Section 5.2.1). The mindmap in Firgure 5.1 summarises the main methods.

24 http://dspam.nuclearelephant.com/

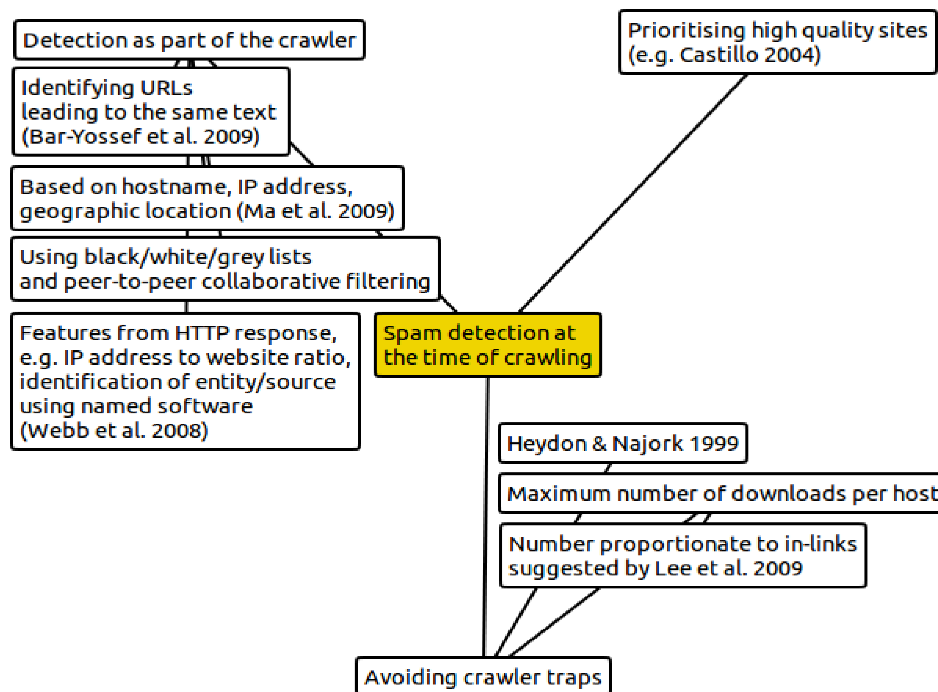

Figure 5.1 Mindmap snapshot of methods for detecting spams at the time of crawling

In addition to the above, Graham (2002, 2003) suggests a simple Bayesian method for spam filtering based on estimating the probability that a message containing a selected single word is spam. Although his method does require some training data (in this sense, it could be considered to be a detection method applicable only at the time of indexing – see Section 5.1.2) it may be handled incrementally with only a small seed set obtained from elsewhere. The strategy was described in the context of email spam but has potential to be modified for blog posts. One of his observations relevant to web spam is that vocabulary of a message should include all header information including formatting tags as they contribute to spam identification.

5.1.2 At the time of indexing

Detection of spam at the time of indexing is the most prolific approach to spam detection. While it is more expensive than detection methods that might be employed at the time of crawling, the amount of information that becomes available upon accessing and comparing home pages makes the detection more robust and effective. We have grouped features that have been indexed for spam detection into four groups: content based features (e.g. language models, words, popular queries, redundancy, style, anchor text, structural elements such as titles and headings, and URL characteristics), usage data based features (e.g. number of visits, query history, and browsing history), link based features (e.g. number of out-links and in-links, neighbourhood size, linking pattern, graph structure, and degrees), and temporal features (e.g. change over time and rate of growth). The mindmap in Figure 5.2 summarises these key features.

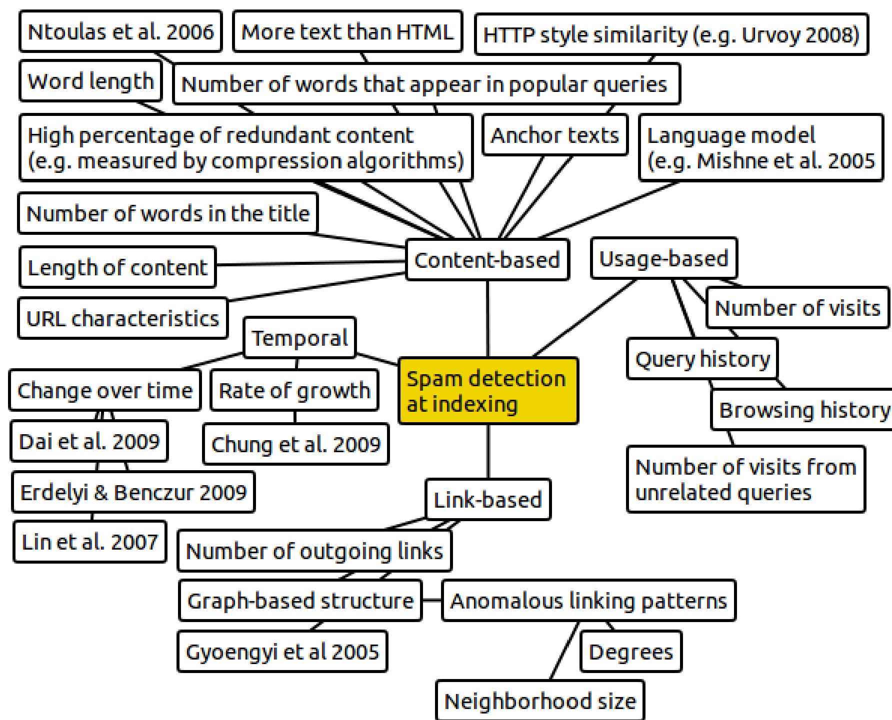

Figure 5.2 Mindmap snapshot of methods for detecting spams at the time of indexing

Ntoulas et al. (2006) concentrated on content based features, noting the percentage of redundant content and average word lengths distinguishing features of spam, and the fact that popular queries appear frequently in spam. Other indicative features draw on anchor text characteristics and words in the title and headings as well and URL. Mishne et al. (2005) used statistical language modelling to compare the probability of both post and comment to be from the same language model, to weed out comment spam. Urvoy (2008) demonstrated some success in clustering pages according to HTML based stylistic similarity thereby propagating the spamicity of a seed authoritative set of pages.

While the content based approaches are the most straightforward and obvious, link based approaches that not only look at in-link out-link statistics (shown not to be too effective against spam) but also more global graph structures and patterns as well as density (e.g. Gyöngyi et al. 2005), temporal features to detect the content change over time (Dai et al. 2009; Erdélyi et al. 2009; Erdélyi 2011), and user data (e.g. click and query history) analysis have been shown to be effective in spam detection. On the other hand, it should be noted that, this type of feature set can only be gathered over a long period of time and after full indexing procedures.

5.1.3 At the time of ranking

Spam detection and demotion at the time of ranking the results of a search issued by an end-user can be enormously challenging: by the time we arrive at this stage, all traditional detection algorithms have been applied, and therefore, those spam content that persist to be in the collection are likely to be those that are highly ranked with respect to content and links, i.e. those that have *already passed all the tests.* However, at the stage of ranking, we are provided with additional information from the query itself, i.e. spam demotion becomes a query specific task. To this effect some have tried to use query term counts, e.g. with respect to different parts of the document

(Figure 5.3), to improve the ranking of authentic documents. Nevertheless, the effectiveness of spam demotion is limited.

It is deemed that spam demotion at the time of ranking can become more effective if temporal analysis, behaviour analysis and usage data be combined with ranking methods: for example, blog life cycle characteristics, out-link characteristics of the blog, and/or the usage statistics and click data analysis correlated to relevance judgements collected as the data is accessed through the web archive could be used to improve or adapt the ranking in the long term.

Learning to rank in the general web retrieval context is already a difficult problem, so we can expect the difficulty to be comparable in the context of weblogs. On the other hand, weblogs change over time, contain dynamic content and are associated to life-cycles. This gives us the potential to produce better ranking for weblogs.

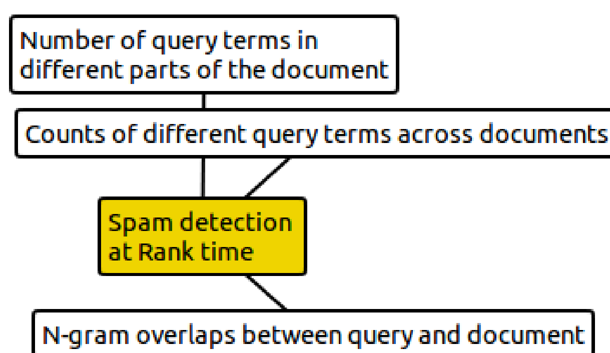

Figure 5.3 Mindmap snapshot of methods for detecting spam at the time of ranking

5.2 Type of features

5.2.1 Spatial characteristics

URL pattern information

Methods using these type of features are based on the observation that:

- spammers tend to stuff the URL with combination, mutations, and permutations of context rich keywords to benefit from the search engine ranking that rewards these URLs
- spammers tend to use hyphens and long length URLs
- domains that are cheap to acquire, such as ".info" domains, tend to be populated with a higher proportion of spam sites than expensive sites such as ".edu".

Classification based on URLs feature 3,4, and 5 n-grams (after tokenising) along with URL lengths and tokenisation techniques that capture usage of special symbols have been suggested by several researchers (e.g. Salvetti & Nicolov 2006; Kolari 2007). This approach is highly desirable because it prevents the spam from entering the archive in the first place (optimising storage). It also carries low cost (i.e. no page fetch required).

Home page content information

Methods here are based on the observation that:

- spammers tend to repeat the same links and keywords
- spam sites have short life span and grow very quickly
- spammers employ a high percentage of nouns and only a few pronouns characteristic of expression of opinions
- coherence of spam content may be lower that authentic content, that is the content would exhibit deviation from a general n-gram language model for n>1
- HTML templates created by automated blog creation software will be repetitive

Classification based on the home page use out-links, anchor text, words, word grams, character grams, HTML tag (as a style template, see, for example, Urvoy et al. 2008), and archive dates to determine blog age and life cycle. To emphasise the repetitive terms and high proportion of nouns and named entities in spam, derived features using compression ratio and entity ratio have been suggested but these do not actually perform as well as the raw features (Kolari 2007). Given that derived features also incur increased processing and index storage overhead, it is unclear whether using these derived features in a large scale blog archive would be beneficial.

Feed-based information

The feature set can be strengthened by using the life cycle of blog characterised by the structured RSS feed (e.g. number of posts and age of blog). There has also been evidence (e.g. Section III.D of Kolari 2007) that the characterisation of HTML template based on RSS feeds enables the classifier to learn spam software detection at a faster rate than when using the home page content.

Link based relational information

The approaches using link based features are inspired by the Google PageRank[25] which is based on the intuition that pages that are cited often are more likely to be "important". This intuition,

25 http://en.wikipedia.org/wiki/PageRank

however, also inspires the spammer to create link farms (see Chung, Toyoda, & Kitsuregawa 2009) that exist solely to promote affiliated web sites. Perhaps this is why Kolari (2007) finds that link based features such as number of out-links (the pages to which the target page points), the number of in-links (the pages that point to the target page), and the number of co-citations (pages to which other pages point at the same time as pointing to the target page), processed on their own do not out perform bag-of-words approach. On the other hand, there has been evidence that, link based features combined with usage data such as query statistics (e.g. see Castillo et al. 2008), can be effective. Also, link based network structure on a global level to determine a trust measure (Gyöngyi 2008) could be effective.

5.2.2 Temporal features and user data

Most of the features discussed in Section 5.2.1 have been described in a fashion that renders them time independent. These kinds of features can be quite limited in its ability to fight spam which is adversarial in nature and perhaps can be best detected by characterising the content change observed across time (e.g. Dai et al. 2009; Lin et al. 2007; Lin et al. 2008). In addition to content change, Erdélyi et al. (2011) showed that changes in linkage across neighbours, in-link growth and death also characterise spam. They measure link structural change surrounding a node using Jaccard similarity coefficient[26].

While these researchers have shown that temporal analysis could be effective, these still seem to fall behind direct content analysis. Some have also suggested monitoring changes of sites that occur as search results of a selected set of popular queries (e.g. Zhu et al. 2011). The latter highlights three observations:

- a small set of popular queries are popular for a long time
- there are a few blogs that are featured frequently in top search results of popular blogs
- some of the top search results of popular queries do not attract noticeable increment of in-link count

They suggest that weblogs that attract few in-link increments while appearing in the top search results are most likely to be splogs (i.e. it is ranked highly but people do not link to it at a noticeable rate). More generally, they propose that a concentrated effort to examine blogs that responds frequently to popular queries could benefit spam filtering approaches. Their approach may be especially suitable for implementation at the time of ranking (Section 5.1.3).

It has already been observed that splogs propagate, grow, and change in a way distinct from the space of authentic weblogs (e.g. see Fetterly et al. 2004; Bhattarai et al. 2009; Erdélyi et al. 2011). The keywords, repetitiveness, network structure, size, and density with respect to splogs change at a different rate from that observed with respect to authentic blogs. It seems reasonable then to exploit these temporal characteristics. Some of the temporal characteristics depends on usage data. For example Liu et al. (2008) present the use of user page visitation behaviour to classify spam. Their approach is based on the following observations:

- Users visit spam pages as a result of search results more often than as a result of recommendation by friends or links from legitimate web sites.
- Spam pages are rarely recorded as source pages because links within spam pages are rarely clicked.
- Navigation time within spam websites is expected to be short.

These observations lead to the construction of spam detectors that employ relative counts of types of visits, types of clicks, and number of pages within sites visited.

26 http://en.wikipedia.org/wiki/Jaccard_index

These statistics, however, can also be affected by automated user data generation (e.g. see Buehrer et al. 2008; Duskin and Feitelson 2009). The trick would be to combine different source of information to learn from each other, such as an ensemble classifier implemented at several stages of acquiring information (see the stages of information acquisition, Section 5.1 and adaptive filtering approaches, Section 5.3).

5.3 Filtering spam adaptively

Kolari (2007) suggested an adaptive filtering method that used an ensemble of classifiers each trained on local features (i.e. no global link relation based features) URL n-gram, words, word-n-grams, charactergrams, tags, out-links, and anchor text. An ensemble classification was performed using these base classifiers on unlabelled instances to improve the effectiveness of each classifier.

His results, however, did not consider how his adaptive methods would fare in comparison to adaptive methods that reflect the changes that occur to the content and network over time. Nor did he consider usage based features (such as visitations, query history) and global link based features (such as patterns of out-links and in-links). To maximise the benefit of temporal change in detecting spam, it is suggested that this could be a direction to explore. Adapting the spam filter based on usage data, which the web archive is in a position to collect, say, by capturing passive user interaction could be highly cost effective. This data may include query statistics, session data, and or click data. Active user feedback such as voting could also be considered.

5.4 Viability feature types mentioned in the literature

As a illustration of the viability of different feature types discussed in the literature, in Table 5.1, selected feature types and their pros and cons have been summarised.

Features	**Pros**	**Cons**
URL analyser/template	Low process cost	Limited information – not very adaptive to change
IP/Post frequency	Could be difficult for spammers to manipulate	Must have history of updates and could become quite involved – e.g. where is the threshold for the frequency and how will it adapt to changes in the spam landscape?
Blacklist	Straightforward methodology and thrid party support available	Could lead to exploding blacklists.
RSS/content match	Indicates some level of agreement that the content is what the RSS feed says it is.	This requires that RSS feed is already available. Strictly speaking this is not spam filtering.
Full content analysis	Could be useful for removing duplicates. Difficult for spam to completely confound.	Could be process intensive.
User feedback	High precision	Labour intensive. Low recall because too many items for humans to examine.

6 Recommended spam detection strategy

According to the BlogForever Description of Work, BlogForever will "develop robust digital preservation, management and dissemination facilities for weblogs"[27]. The ability of this project to reach these goals during the scope of this project, both effectively and efficiently, requires that we consider what types of collections we can reasonably expect and to anticipate our needs for spam filtering in relation to those expected use-cases.

Our current expectations of possible use-cases, according to the preliminary analysis of qualitative interviews conducted with potential future users (e.g. reported as part of BlogForever deliverable D4.1 "User requirements and platform specifications"), involve blog collections of two main types. The first type would be the collection of distinct or expert blogs, targeted by an administrator for specific preservation purposes and according to specific criteria. These types of collections are what we expect of potential users such as libraries, universities or organizations that provide Blog hosting, for example. The second type consists of those B2C consumer blogs, that would be potentially collected in huge volume and without prior identification.

The BlogForever Description of Work targets two types of spam for detection[28]:

1. Blogs that are created for the sole purpose of raising the rank of affiliated target websites by linking to and supporting these websites (**spam blogs** or **splogs**)
2. Comments that are submitted to the blogs for the sole purpose of disseminating content irrelevant to the original post, including links to raise the rank of affiliated sites, and/or relaying abusive content (**comment spam**)

For expert blog collections, the blogs to be preserved are expected to be chosen by an administrator, entailing less potential for the inclusion of spam blogs within the chosen body of blogs. For this reason, spam detection strategy with respect to expert blog collections will need to place more weight on comment spam. There are several ready-made, anti-spam tools with learning capabilities available that deal with comment spam (examples are discussed in Chapter 3). These learning capabilities are triggered when errors of the anti-spam tool are reported back by blog content moderators. These APIs will form the first stage of the BlogForever spam detection strategy. For the second type of collection (B2C blogs), the current weblog spider prototype architecture (Section 6.1) suggests that ping servers will be used more extensively, making the detection of spam blogs more of a problem. While it might be possible to customise existing anti-spam tools for the detection of spam blogs (more on this in Section 6.2.1), the feasibility of this approach is yet unclear. For both types of blogs (expert blogs and consumer blogs), it is recommended that intelligent blog harvesting strategies be developed to alleviate overload of URLs arriving at the spam detection component rather than trying to compromise the quality of the detection method (Section 6.3.1)

As a point of observation, even in the case of the expert blog collection, the assumption that the administrator will be able to list blogs relevant to the collection may become questionable in the future. Blogs are created at a alarming rate: for example, the number of tumblr blogs is now over 41 million[29] and the number of WordPress blogs has been reported to be 70 million[30]. Last year, Technorati was reported to be tracking over 100 million blogs[31](note that Technorati no longer indexes non-English blogs). In the plethora of emerging blogs, archivists of the future will not be

27 Part A1: Project summary , BlogForever project Description of Work
28 covering three spam types listed on Page 28, Part B, BlogForever project Description of Work
29 http://www.tumblr.com
30 http://royal.pingdom.com/2012/01/17/internet-2011-in-numbers/
31 http://www.infotoday.com/linkup/lud021510-stern.shtml

able to hand-pick blogs for archiving, nor will they be able to rely completely on popularity rankings (which is almost impossible to surpass as a late-comer in the game even with high quality content, unless the new blogger resorts to spamming techniques themselves), to find relevant blogs. For example, in order to harvest and archive blogs addressing events of social significance across the blogosphere (not an unlikely mandate on an archivist) such as those discussed by Chen (2010), a blog archive administrator might need to automatically discover these blogs. In such a scenario, spam blog detection (not just comment spam detection) to support selection of relevant blogs will become necessary. In fact, in the future, it may be that the spam detection strategy will become an essential component of selection and appraisal procedures carried out by the archivist.

Consequently, it is important to ensure that the spam detection mechanism set up by the BlogForever archive should not falter in the continuation of its service and in the maintenance of the quality of its service. As such, the design of a spam detection strategy for a robust weblog archive should not depend solely on closed third-party tools. A third-party service could be discontinued at any time and development could cease at any time (i.e. no longer evolving in response to new emerging knowledge about spam). While the continuation and evolution of these tools to meet the demands of the user community is likely, and modifications might be possible on a need-to-do basis, there is no guarantee. To ignore the ramifications of such a risk goes against good archiving, repository management, and digital preservation practices.

In response, in addition to the first stage of spam detection carried out by third-party APIs (Section 6.2.1), a second stage detection based on statistical methods is proposed as a backup strategy (Section 6.2.2). The second stage detection will also be carried out at the time of web crawling, i.e. the spam detection will be in full cooperation with the BlogForever weblog spider. In fact, the design of the spam detection framework has been configured to meet the general recommendations presented in the BlogForver deliverable D2.4 BlogForever weblog spider prototype (see discussion in Section 6.1).

The basic step for a spam detection module takes an input candidate and determines the likelihood of it being spam or not (Figure 6.1). Once the input is flagged as spam, it can be ignored, kept within the archive, reported to a spam database, or collected in a separate location as part of an internal spam database of the archive. The course of action would depend on the goals, policies, objectives, mandates, and legal requirements imposed on the archive.

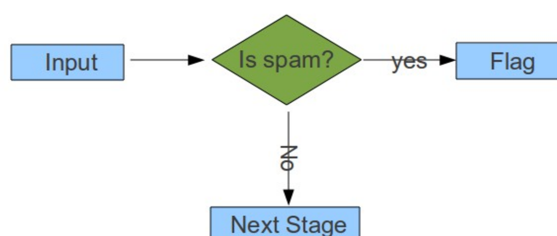

Figure 6.1 Basic step in the spam detection module

While the retention of spam will pose a considerable burden on a web archive (in light of the organisation's data storage capacity), as illustrated in the introduction to this report, spam has distinctive historical, research, and commercial value. To repeat, spam tells a story about how the

adversarial development of technology versus spam has evolved over the years. Thus, we recommend that all spam or some selected portion of spam should be preserved as part of future web archiving initiatives. Regardless of the policy on storing (or not storing) spam, however, the URL of the flagged spam should be communicated to the Source Database (see Section 6.1) specified in the BlogForever deliverable D2.4 Weblog Spider Prototype and Associated Methodology to be used for URL blacklist analysis.

Finally, in Section 6.3, we have proposed some end-user mechanisms that might be integrated into the User Interface design and repository Data Management framework that could further support the spam handling strategy of the BlogForver archive.

6.1 Spam detection strategy and the BlogForever spider prototype

The final spam detection strategy must be designed to fit in with other components of the BlogForver archive implementation. Further, the spam detection component should involve low cost in implementation, processing, and maintenance. To support low cost, the spam detection will be limited to take place during or right after the web crawling or spidering process, so as it decreases involvement with the more complex architecture of the repository. The BlogForever spider will be prototyped, designed and implemented as part of BlogForever deliverables D2.4, D4.2, and D4.3, respectively. Already, in the deliverable D2.4, it was proposed that the spam detection takes place in two places: using a URL blacklist stored in the Source Database ("Source DB" in Figure 6.2), coming from previous instances of spam detection, and using the fetched content (see "Spam filter" component in Figure 6.2) in relation to either the Host Analyzer or the Worker stage of the spidering process. By adhering to the architecture suggested in D2.4, we can ensure compatibility between the spam detection strategy and the general BlogForever archive implementation.

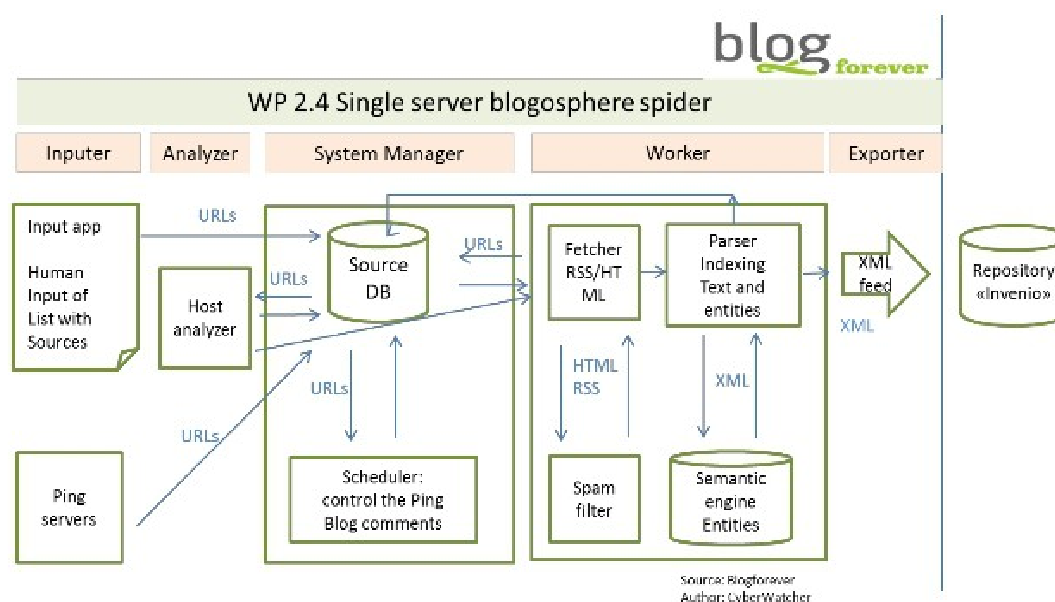

Figure 6.2 BlogForever Spider Prototype Architecture (Section 4.2.4, BlogForever deliverable D2.4 "Weblog spider prototype"). Spam detection is part of the "Worker" stage of the prototype.

The discussion in the D2.4 deliverable suggests (Section 4.2.4, D2.4), as first line of defence against spam, the elimination of URLs that:

- do not match acceptable blog formats (based on an ID3 decision tree algorithm),
- have splog-like characteristics (e.g. high volume of associated updates and pings),
- do not lead to the identification of valid RSS feeds for blogs and blog posts,
- lead to mismatch of RSS and HTML content using Levenshtein distance.

Further suggestions made by deliverable D2.4[32] includes the use of heuristic scoring functions based on in-link statistics and spam scores of posts, mentioned by Wu (2007), as well as, a list of derived features, mentioned by Kolari (2007). While the steps to examine URL characteristics are reasonable precautions within any web spidering frameworks that aim to retrieve *authentic blogs*, the features proposed in Section 3.5 of the deliverable pose a couple of immediately observable problems. The distributional characteristics proposed by Wu (2007) is based on data and even usage data (e.g. query logs) collected over a considerable time period. At the initial period of launching the archive this kind of data is limited. Also, characteristics such as in-link increase rate can be manipulated by a clever spammer, say, for instance, more easily than linguistic elements such as indent and coherence.

Also, the derived features described by Kolari (2007) take time to extract, and, further, it should be noted that Kolari (2007) observes, in his experimentations, that the performance of spam detection based on these derived features do not lead to as good a performance as that obtained from raw features (e.g. bag of words, n-grams, anchor text). For example, he reports that derived features at its best achieve an accuracy rate of 0.75, while classification on the bag of words method achieves 0.9 accuracy. Consequently, we recommend that the initial approaches be limited to the use of raw features.

Graham (2003) who also uses statistical methods on words (he does not distinguish header information from content) reports that 99.75% of his email spam was caught by his Bayes spam filtering method[33] and he measured a false positive rate of 0.06%. Admittedly, his method was applied on email, and it is questionable whether the same methodology would apply to blog posts, but his philosophy seems sound: he contends that strategies based on heuristics can eventually be out smarted by a clever spammer, and that strategies based on deep language processing techniques are more robust because the message of the spam has to be explicitly or implicitly (as the target of a link, for example) embedded in the post.

Based on the finding of Kolari (2007) and Graham (2003), it is recommended that any implementation of the BlogForever spam detection strategy going beyond the integration of ready made API (e.g. those mentioned in Chapter 3), use raw features such as words in the content rather than derived or heuristic features.

32 Section 3.5, D2.4
33 http://paulgraham.com/better.html

6.2 Blogforever spam detection methodology

In this section, we have described two core phases intended to form the basis of the BlogForever spam filtering strategy. The initial stages of the first phase described in Section 6.2.1 (based on looking up blacklisted IP/URLs and/or URL pattern rules) is largely independent of the content of the target page, post or comment. However, it might be commented here that URL lists or pattern based methods are not reliable in the long term, as this is something that spammers change frequently for the very reason that it is the first line of defence used in any spam detection methodology. On the other hand the volume of URLs arriving at the ping server (the source of harvest suggested suggested within BlogForever deliverable D2.4) makes it difficult to examine the target content of each URL. There seem to be only two immediate solutions available for dealing with this dilemma:

1. The creation and maintenance of multiple spam filtering agents on several servers assigned with limited number of URLs.
2. The design of an intelligent spidering strategy for harvest in parallel to the update strategy suggested in D2.4. that moves away from the breadth-first harvest at the ping server to depth-first harvest sourced through identified non-spam blogs (see Section 6.3.1). The depth-first harvest increases the likelihood that material relevant to material already in the collection is being collected, and also encourages the collection to approximate the single network of linked pages (i.e. the statistics used in link-based ranking becomes more accurate).

The first of these may be only a temporary solution as the number of URLs at the ping server may grow at an accelerated rate in the future. While the possibility of extensive investigations in the direction of the second solution may be limited within the BlogForever project, it is recommended that future archiving initiatives give it serious consideration.

The latter part of the first phase (blog identification and the use of anti-spam APIs) and the second phase (section 6.2.2) of the spam detection (adaptive ensemble classifiers) relies on some form of blog content. It should be pointed out, however, that the term content here does not necessarily refer to comment, post, or page content associated to the URL. Even with the information received at the ping server only, access to title, url and links to RSS is available. These features may be used in the first instance as content submitted to APIs and/or to apply the other approaches outlined in Section 6.2.2 at an early stage to eliminate obvious instances of spam. For this reason, the strategy has made no distinction between splog detection and content spam detection: the philosophy behind the simplified strategy suggested here, at least within the limits of present resources, is to cast the detection of different types of spam instances as a question of considering content and links and changes thereof at different levels of granularity, and at different time periods in the repository life cycle.

Finally, it should be noted that none of the strategies outlined here is intended for blog platform providers and is intended for use within retrospective weblog repositories created after blogs and their content have been published within the blogosphere.

6.2.1 IP and URL Blacklist/Whitelist lookup, blog detection, and third-party APIs

This stage of the spam detection is designed to reduce the spam detection process overhead by carrying out simple IP and URL matching against blacklisted and whitelisted IP addresses and URLs, leaving more sophisticated methods for later to be applied on a smaller set of data. All three

steps are expected to be carried out during the Host Analyzer stage of the weblog spider (see Figure 6.2).

Blacklist/Whitelist Lookup

As mentioned in Section 6.1, the BlogForever weblog spider prototype already incorporates part of the first and second steps (URL lookup and blog detection) of this detection stage. However, we propose here three improvements of the already implemented steps:

1. In addition to URLs, IP addresses, if available, could be checked against a blacklist database. It has been observed (e.g. by Kolari 2007) that the same IP addresses use different URLs to avoid being caught as spam.
2. The dnspython toolkit[34] could be used to look up URLs associated to a selected IP address mapped to a rejected URL. Associated URL can be added to the Source Database suspected URL list.
3. In addition to storing internally detected suspect URLS in the Source Database, it might be worth exploring external APIs, for example, that from SplogSpot (http://splogspot.com/) to check for blacklisted IP addresses and URL. This is to circumvent the fact that, at the beginning, there are very few suspect addresses recorded in the Source Database.

Blog detection/identification

The weblog spider also already incorporates simple blog detection techniques using URL format analysis, RSS check (URLs without RSS links are rejected), and RSS validation against content. In conjunction with query logs (see discussion of usage data in Section 6.3), a detection method based on the number of popular query words in the URL has been shown to be an indicator of spam blogs (Kolari 2007).

While an additional backup strategy based on weblog structure (drawing on the elements of blogs that have been identified as part of the work presented in the BlogForever deliverable D2.2, “Report on Weblog Data Model”), and Kolari’s support vector machine detection method using a combination of binary features coming from words in the content and page URL (this combination was reported to have best performance resulting in precision 0.985 and recall of 0.966) would be useful, the increase in overhead may make it not worth implementation. It is proposed that, if resources are available, a test be conducted to evaluate the accuracy of this strategy against the projected increase in overhead.

Third-party API

In Chapter 3, we discussed a range of ready-made third-party anti-spam APIs. There is a wrapper in many languages for most of these APIs. Python API, wrappers, or code examples for most of these APIs have been provided in Appendix A. The focus on python is due to the awareness that the BlogForever repository software Invenio is based on python and also that python is recognised an ideal language for efficient string processing[35]. We have not included BadBehaviour and SpamKarma in the discussion here. Although there is still a strong support from the community for these anti-spam services, they have been known to be described process intensive[36].

34 http://www.dnspython.org
35 http://stackoverflow.com/questions/635155/best-language-for-string-manipulation
36 http://wordpress.org/support/topic/bad-behavior-and-admin-ajaxphp-crawl-backend

The methods available to each of these APIs are slightly different. Akismet python API[37] is undoubtedly the simplest including four methods for verifying a submitted key, checking a comment for spam, and reporting back spam and ham, respectively. A sample code is presented in Appendix A.1.2. The same API can be used for TypePad antispam (Appendix A.4) with the appropriate key.

Defensio also comes with a python library[38] (see Appendix A.3). Unlike Akismet, Defensio provides a score of spaminess. The latter functionality might be useful in other ways as well. For example, if the spaminess level is low then we might allow its inclusion into the repository but make use of this as a feature in tagging or ranking the blog post at the time of displaying it to the end-user of the archive. Further functionalities of the Defensio API is available as a pdf document at: http://download.defensio.com/docs/api/defensio_api_2.0.pdf

In addition to APIs mentioned in Chapter 3, there are also services like BlogSpam[39] (Appendix A.5). Comments can be submitted through a server proxy API using XML-RPC standards. Unlike other offerings listed here, with BlogSpam, you can run your own service by downloading the code[40]. However, there may be little support for the python and/or other languages in this case.

There is also the Trac project SpamFilter[41], which operates as a bridge across these different API to allow customised selection of any of these API. It also provides a gateway to specialised spam filters such as StopForumSpam[42] and LinkSleeve[43], as well as, access to a range of internal spam filtering strategies including tools for implementing a trained Bayes spam filter. Python codes using the Trac SpamFilter and each of these API are at the SpamFilter website. As an example, however, the sample code for the implementation of Akismet within this context has been included in Appendix A.6.

One widely used external anti-spam service that is not included in the Trac SpamFiltering framework is Mollom (see Chapter 3). An independent python wrapper for the Mollom API is available at http://www.itkovian.net/base/python-wrapper-mollom/

Ideally, we would like to propose that several APIs be used in parallel. We could not find a reference to such a use case at the time of writing this report but perhaps three or more spam filter APIs can be operated as a committee of spam filters. This can be applied to incoming posts and comments. The output can be also tagged with Defensio's spaminess index for later use (see discussion above). If a post is identified as spam then the website is suspect and can be greylisted. Greylisted websites can be assumed to be spam blogs or further examined by retrieving further posts and comments from the site for examination.

A committee of spam filters can be either implemented through Trac SpamFilter (this has not been tested for feasibility yet), or independently scripted. The effort in creating a committee should be minimal.

Additionally, the use of blekko[44] should be investigated. blekko claims to be a spam free search engine for the web. They use slashtags (indicated topic/genre) curated by experts to eliminate spam

37 http://www.voidspace.org.uk/python/akismet_python.html
38 http://www.defensio.com/downloads/python/
39 http://www.blogspam.net
40 http://blogspamnetapi.codeplex.com/releases/view/62472
41 http://trac.edgewall.org/wiki/SpamFilter
42 http://stopforumspam.com/
43 http://linksleeve.org/
44 http://www.blekko.com

websites. They claim that an API is available that will allow external applications to use the slashtags. However, how easy it is to use the API is yet unknown, and whether it can be applied to the blog context is questionable. For example, the content they index may not include many blogs.

6.2.2 Ensemble adaptation framework

The steps in Section 6.2.1 will reduce the scope of sites that need serious analysis. The next stage of spam filtering is designed to support simplicity while enabling adaptability. An automatically adapting spam detection strategy may involve some cost at the beginning of its implementation but will have an advantage in the long run by reducing the necessity of manual intervention for the maintenance, improvement, and retraining of the spam detector.

Many researchers engineer features (see derived features – e.g. compression ratio and entity ratios described by Kolari 2007; and link structural coefficients used by Erdélyi et al. 2011) or select the best ones from a big pool of features. This approach generally creates a fast learning curve at the beginning, and is perfect for finding local optima, but does not necessarily lead to a global optimum. In fact, Kolari (2007) shows that the use of poor base classifiers to build an ensemble adaptive framework is effective in achieving relatively high performance levels at a later stage of the learning process (reaching precision and recall of approximately 92% and 94% across all classifiers – some of these classifiers started off at less than 84% precision and 71% recall).

He also presents evidence that supervised classifiers using Support Vector Machine algorithms on raw content features (e.g. words) performs better than those based on derived features (e.g. ratio between words belonging to different part of speech). He also shows that simple link features (e.g. number of in-links; number of out-links; number of sites co-cited with the target site) are not as effective as word features. While some researchers (e.g. Wu 2007) have shown that in-link growth rate is a good indicator of spaminess, these are features that can be captured over a considerable period of time (see also discussion at end of Section 6.1). Also, note that the more derived the feature is, the more expensive the process will be to capture it, and difficult to efficiently integrate into the rest of crawling, and indexing system of the repository.

Here we suggest an ensemble frame work with poor base classifiers distributed across those trained on content features (e.g. base classifiers presented by Kolari 2007), link structural features (e.g. in-link counts, out-link counts, network node degrees; see also Erdélyi et al. 2011), temporal features (e.g. changes of content and link features). These can then be used independently as the basis for a classifier using a variation of Naïve Bayes (see Appendix B.1 for a simple implementation of, for example, Graham's Bayes spam filter[45]) and/or Support Vector Machine[46] (see how it is used in Appendix B.2) to form an ensemble of classifiers that can learn from each other, without manual input. In the first instance, the ensemble classifier labels a new candidate for spam detection using the averaged probability across all classifiers. The new labelled instance is used to re-trained and hopefully improve each individual classifier (see Figure 6.3).

Initially we suggest only three sets of features (resulting in three classifiers):

- tokenized content (this is the entire HTML for the content retrieved before parsing the HTML),
- link structure features including number of in-links, number of out-links, number of co-cited sites (if possible),

45 http://www.paulgraham.com/spam.html
46 http://tfinley.net/software/svmpython1

- temporal features that track changes in the tokenized content, link numbers and co-citation numbers since the last update as a ratio with respect to time.

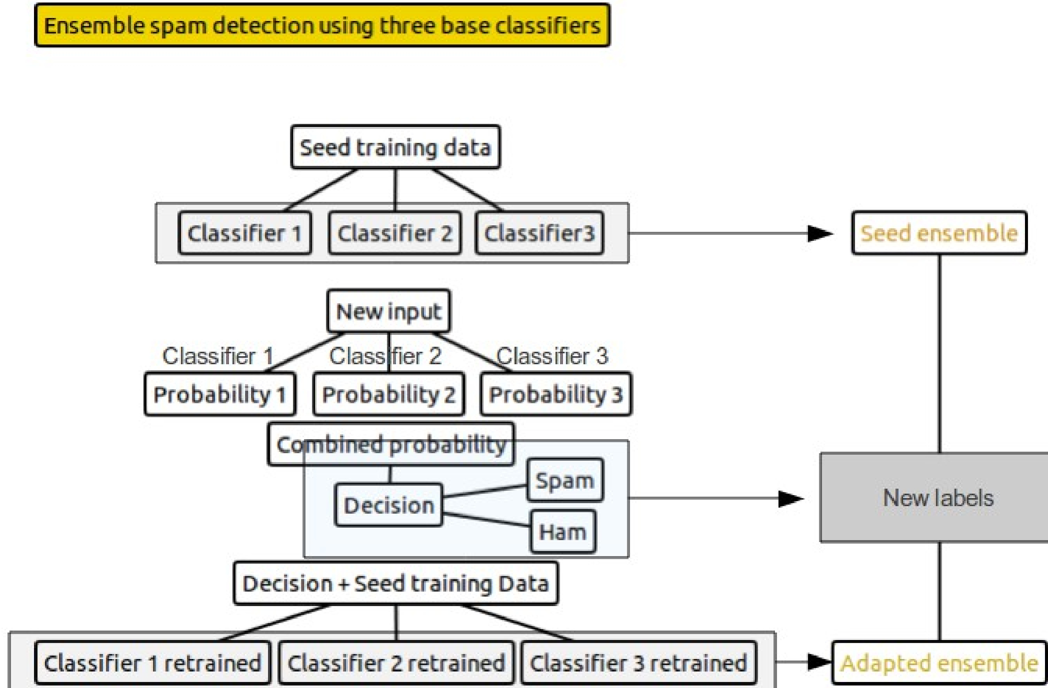

Figure 6.3 The three stages of an adaptive ensemble classifier workflow

Not all of the features will be available in the first instance. To compensate we could first work with a single content based classifier, followed by new classifiers based on link features and temporal features (and, even later, classifiers based on features derived from end-user activity) added to the ensemble at a later date. In fact, the ensemble spam detection strategy can be introduced at a later date based on a small samples of posts and comments labelled as spam or non-spam as a result of the spam detection carried out by the committee of ready-made anti-spam APIs (Section 6.2.1).

Because the features described above are very basic features of a webpage, the implementation of the framework will not be incur extra labour with respect to data extraction (that is, these features need to be extracted as a matter of course during indexing and can be partially communicated back to the spam detector). Support vector machines can, however, be quite process intensive (which is why a lighter version of the algorithm has been suggested – Appendix B.2) so depending on the volume of classification, it might be better to select a simplified Bayes classification approach (such as that presented in Appendix B.1).

Note that the features for the individual classifiers are all features that need to be extracted anyway as important aspects of the blog as part of the data extraction framework. Later, if resources permit it, an advanced module could be developed to build other classifiers into the ensemble including one based on user data features (visitation statistics and browsing history, such as that presented by Zhu et al. 2011; query log statistics such as that presented by Castillo et al. 2008) to augment the ensemble.

6.2.3 Requirements for the implementation of the spam detection strategy

In Sections 6.1, 6.2.1, 6.2.2, we described the steps for an efficient spam detection strategy to be employed within the BlogForever weblog archive. We summarise the workflow in three phases (Figure 6.4). The first two phases (Phase 1, described in Section 6.2.1, and, Phase 2, described in Section 6.2.2) form the core components of the detection strategy. In this section we discuss the requirements of these two phases. The details of Phase 3 is discussed in Section 6.3.

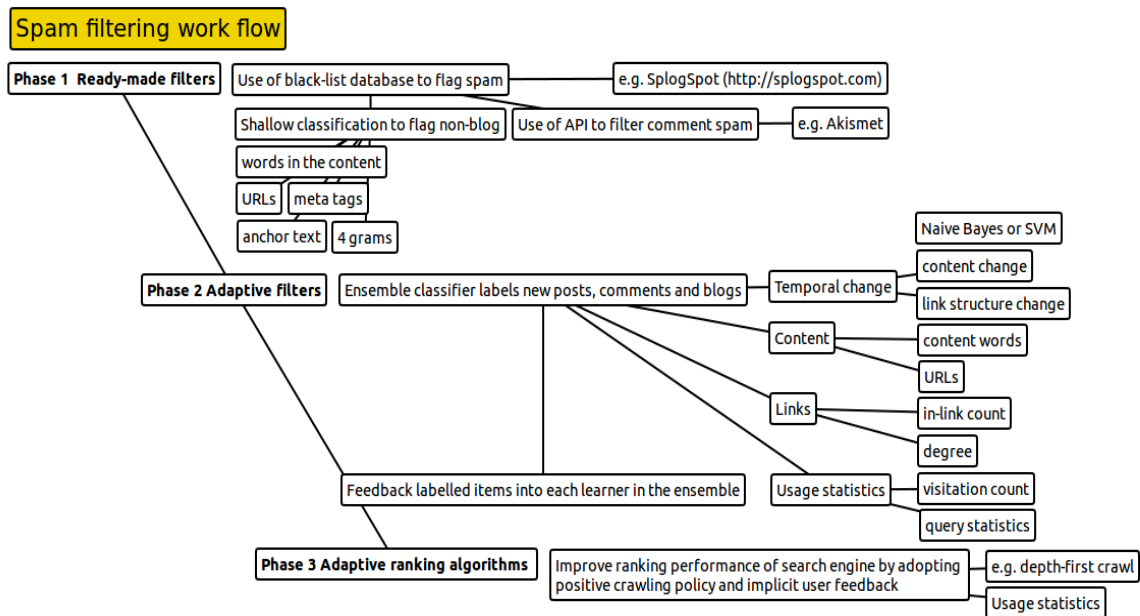

Figure 6.4 Spam detection workflow in three phases

Training data for the ensemble classifier

Since it is unlikely that the spam detection component will have access to explicit user feedback, the Phase 2 component of the spam detection strategy will lack the training data to initiate the base classifiers. Although, this data could be obtained externally (say, for example, through SplogSpot or Technorati), a better strategy for the archive is to feed a portion of the labelled result of Phase 1, initially, as training data for Phase 2. This would make the data more domain specific (that is, relevant to the collection objectives – both for BlogForever, and for future archives), and the classifiers are likely to perform better when trained on representative datasets.

Considerations with respect to indexing

The training of the ensemble classifier does not necessarily depend on the full indexing of the collection, as you might do within the repository. The initial statistics that drives the classification can be based on the statistics of all the spam considered together as one document and all the non-spam (a.k.a ham) considered together as one document. Also, the document vectors can be reduced in dimension on the basis of the top 20 most distinguishing words in the document.

Not all features will be available from the beginning

While there is always some content associated to a blog candidate for inclusion into the repository, links (incoming and outgoing) and changes of content and links may not be available at the outset. This can be handled by initially only introducing the content-based classifier augmented by a link-based classifier and a classifier using temporal features (changes in content and links) when information on these features become available.

Defining what we mean by content

In extracting content to be submitted to the APIs and ensemble classifier, while we suggest tokenizing of HTML and URL content, it is not assumed that we will be removing structural and header information in the form of HTNL or XML from the page or message before submitting it to the relevant classifiers. For example, Graham (2003) found that, in the case of email, the inclusion of header information as part of the content improved classification. It is suggested that initially we do not rip content from HTML and evaluate efficacy before taking this step. If we do not need to parse html, time and processing intensity can be reduced.

Mapping the spam detection Phase 1 and 2 onto the weblog spider

Both Phases 1 and 2 can be implemented at several stages of the weblog spider (Section 6.1). As mentioned earlier, while the amount of information available at different stages of the crawl (updates from the ping, updates from the RSS feed, and update of the target page content) will differ, it should also be noted that there will always *some* content available.

Step 1: Weblog spider Host Analyzer

At this stage of the weblog spider we will only have information received from the ping server. Nevertheless, even at this stage, we have access to the title, url, rss link (while not all updates received at the ping server is accompanied by rss links, it has already been suggested that only URLs with RSS feeds be considered for inclusion into the repository. Hence, it seems reasonable to discard updates at the ping server that come without RSS links). This content satisfies the minimum requirements for information to be submitted to the APIs and ensemble classifiers.

Step 2: Weblog spider Worker

The RSS feed for the URLs that have passed the title, URL, and RSS link check in Step 1, will be retrieved at this stage. Once the RSS feed is available, we will have content from posts and comments available as well as url, title and RSS link. This in turn can be submitted to the suggested committee of APIs and the ensemble classifier.

Step 3: after the full content has been fetched by the weblog spider

For URLs surviving the classification at Step 1 and 2, we arrive at Step 3 which is based on the submission of full webpage content to the APIs and ensemble classifier.

In the next section we will discuss some approaches that can be implemented within the wider content of the repository to support spam detection, and, more generally, selection and appraisal. For example, user feedback (e.g. spam reports from users of the archive) could be collected to inform the Source Database of the weblog spider and the API servers to improve Phase 1 (by updating black lists and utilising the API adaptive functionalities) and Phase 2 (by updating the training data). Also, it might be possible to improve the weblog spidering strategy with respect to aggregating repository content beyond that received from a general ping server to avoid large volumes of URL at earlier stages of the detection.

6.3 Other means of handling spam

In addition to the two proposed phases described in Sections 6.2.1, 6.2.2, and 6.2.3, in this section, we list a few related areas of research regarding repository design that we believe to be relevant to spam detection. Apart from the direct spam filtering method that have been proposed so far in this report, another way to fight spam cost-effectively is to make improvements to the information retrieval strategy. That is, by ranking truly relevant material higher, we can make spam effectively invisible without actively removing them. In relation to this, we point to four different research areas for improving retrieval.

6.3.1 What we crawl and have in the collection affects retrieval performance

It has been noted that crawling policy on retrieval performance (Fetterly et al. 2009) affects the performance of the retrieval algorithm. That is, different collections respond differently to the algorithms we use. For example, web spidering approaches could follow the “breadth-first” or “depth-first” approach. Some have observed that prioritising depth could improve the usability of the collection. While this is in the context of general web, analogous strategies may exist with respect to blogs. Hypothetically, for example, while comments can be submitted from outside the blog author’s circle, it may be less likely that posts are submitted by suspect contributors. Hence, if the blog has already been identified as authentic information found within the posts might be a source for finding more authentic blogs by association. It is recommended that such strategies be investigated.

6.3.2 Option to report spam: a way to improve the user interface?

The ready-made APIs discussed in this report rely on user reports of erroneously labelled spam and ham (non-spam) to learn to classify spam more accurately. In the way we are using the APIs, it is difficult to report back spam to the API servers, because explicit human moderation of the data labelled as ham and spam is not configured into the design (as it is in the context of active blog plugins). It could result in continued poor performance of the APIs on blogs that meet the requirements of the archive. If we use the blogs labelled by the APIs for Phase 2, this error may be propagated. Incorporating a mechanism to allow users of the archive to report spam within the user interface could, therefore, prove useful. Recorded spam reports can be also used to improve the search results displayed within the user interface.

6.3.3 Using click, query and usage data regarding blogs to improve search

User implicit feedback models (e.g. Joachims and Radlinski 2007) information retrieval performance has been reported in recent years. For example, some have observed that, while clicks do not indicate relevance in itself, clicks may indicate that the clicked item is more relevant than those preceding the item in the list. Also, some of the strategies that have been suggested for spam filtering in recent years depend on the use of user data features (visitation statistics and browsing history, such as that presented by Zhu et al. 2011; query log statistics such as that presented by Castillo et al. 2008). It is, therefore, recommended that user data (visitation statistics, download frequency, queries and click data) be recorded and managed as part of the repository design, if possible. Usage data of selected blogs may become especially useful for improvements in ranking and spam detection. Some of the data here could be used for IP analysis (see discussion of dnspython toolkit in Section 6.2.1) to populate and update whitelists with respect to the Source Database.

7 Conclusions

In this report, we have presented a survey of web spam filtering methods relevant to blog spam detection. We have looked at different types of spam that have infiltrated the Blogosphere (Chapter 2), discussed the pros and cons of different ready-made APIs and tools (Chapter 3 and Section 6.2.1), presented an overview of the research landscape in the area of spam detection (Chapter 5), and proposed what we view to be a feasible two-phase spam detection strategy for implementation as a component in the BlogForever weblog archive (Sections 6.2.1, 6.2.2 and 6.2.3): consisting of,

1. **Phase 1:** spam detection carried out by a committee of APIs
2. **Phase 2:** spam detection carried out by an ensemble of adaptive base classifiers built on three independent sets of features.

The proposed strategy has been designed to conform to the architecture of the weblog spider described in Deliverable D2.4 of the BlogForever project. The strategy described, however, is subject to modifications dependent on further discussions and research results within the BlogForver project, especially with respect to future finding related to repository and weblog spider design (Work Package 4 of the BlogForever project).

We have also discussed some factors involving the design of the repository that might affect the handling of spam (Section 6.3), as points of consideration in the next steps of the BlogForever repository design. Where appropriate, we have also suggested improvements that might be introduced in the future, should the resources of the BlogForever project, or organisations adopting the platform, allow investigations in that direction. Spam detection is a complex problem and the proposed strategy for its detection is a naïve approach at best. The main objective of the BlogForever project is not in the development of spam detectors. As such resources for its development are deemed to be limited.

As a concluding comment, we would like to add that there was one final component missing in the strategy described: this must consist of a vigilant continuation of investigations into new technologies that might become available. Spam is adversarial, that is, the better we make our technologies for discovery, research, interpretation, and search, the cleverer the spam will become. It is very unlikely that the solution developed now will last into the future.

8 References

[1] J. Attenberg and T. Suel, "Cleaning search results using term distance features," in *Proceedings of the Fourth International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 21–24, New York, NY, USA: ACM, 2008.

[2] A. A. Benczúr, I. B ́ o, K. Csalog ́ny, and M. Uher, "Detecting nepotistic links by language model disagreement," in *Proceedings of the 15th International Conference on World Wide Web (WWW)*, pp. 939–940, ACM Press, 2006.

[3] A. Bhattarai, V. Rus, D. Dasgupta, "Characterizing comment spam in the blogosphere through content analysis", In *Distribution*. IEEE Press, pp. 37-44. 2009. Available at: http://dx.doi.org/10.1109/CICYBS.2009.4925088.

[4] I. Bíró, D. Sikl ́si, J. Szab ́, and A. Bencz ́r, "Linked latent dirichlet allocation in Web spam filtering," in *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 37–40, ACM Press, 2009.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[6] G. Buerer, J. W. Stokes, K. Chellapilla, "A large-scale study of automated web search traffic", *Proceedings of the 4th international workshop on Adversarial information retrieval on the web (AIRWeb '08)*, ACM, pp 1-8, ISBN 978-1-60558-159-0, New York, NY, USA. 2008. http://doi.acm.org/10.1145/1451983.1451985

[7] C. Castillo, "Effective Web Crawling," *PhD thesis*, University of Chile, 2004.

[8] C. Castillo, C. Corsi, D. Donato, P. Ferragina, and A. Gionis, "Query-log mining for detecting spam" In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web (AIRWeb '08)*, Carlos Castillo, Kumar Chellapilla, and Dennis Fetterly (Eds.). ACM, New York, NY, USA, pp17-20, 2008. DOI=10.1145/1451983.1451987 http://doi.acm.org/10.1145/1451983.1451987

[9] C. Castillo and B. D. Davison, "Adversarial Web Search", *Found. Trends Inf. Retr.* 4, 5 (May 2011), pp377-486, 2011. DOI=10.1561/1500000021 http://dx.doi.org/10.1561/1500000021

[10] J. Caverlee, "Tamper-Resilient Methods for Web-Based Open Systems," *PhD thesis*, College of Computing, Georgia Institute of Technology, August 2007.

[11] K. Chellapilla and D. M. Chickering, "Improving cloaking detection using search query popularity and monetizability," in *Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 17–24, August 2006.

[12] X. Chen, "Blog Archiving Issues: A Look at Blogs on major Events and Popular Blogs.", Internet Reference Services Quarterly, 15, pp21-33, 2010. DOI: 10.1080/10875300903529571.

[13] Y.-J. Chung, M. Toyoda, and M. Kitsuregawa, "A study of link farm distribution and evolution using a time series of Web snapshots," in *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 9–16, ACM Press, 2009.

[14] N. Dai, B. Davison, and X. Qi, "Looking into the past to better classify Web spam," in *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 1–8, ACM Press, 2009.

[15] B. D. Davison, "Recognizing nepotistic links on the Web," in *Artificial Intelligence for Web Search*, pp. 23–28, AAAI Press, July 2000.

[16] O. Duskin and D. G. Feitelson, “Distinguishing humans from robots in web search logs: preliminary results using query rates and intervals”, in *Proceedings of the 2009 workshop on Web Search Click Data (WSCD '09)*, 978-1-60558-434-8}, pp. 15-19, ACM, Barcelona, Spain, 2009. http://doi.acm.org/10.1145/1507509.1507512

[17] M. Egele, C. Kolbitsch, C. Platzer, “Removing web spam links from search engine results”, *J. Comput. Virol*, vol. 7, no. 1, pp. 51-62, Springer, February 2011. http://dx.doi.org/10.1007/s11416-009-0132-6

[18] M. Erdélyi, A. A. Benczúr, J. Masanes, and D. Siklósi, “Web spam filtering in internet archives,” in *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 17–20, ACM Press, 2009.

[19] M. Erdélyi, and A. A. Benczúr, "Temporal Analysis for Web Spam Detection: An Overview ", In the *Proceedings of the 1st Intl. Temporal Web Analytics Workshop (TWAW 2011)*, Hyderabad, India, March 28, 2011, pp.17-24.

[20] M. Erdélyi, A. Garzó, and A. A. Benczúr, "Web spam classification: a few features worth more", In the *Proceedings of the Joint WICOW/AIRWeb Workshop on Web Quality (WebQuality 2011)*, Hyderabad, India, March 28, 2011, ACM Press 2011.

[21] D. Fetterly, M. Manasse, and M. Najork, “Spam, damn spam, and statistics: Using statistical analysis to locate spam Web pages,” in *Proceedings of the Seventh Workshop on the Web and databases (WebDB)*, pp. 1–6, June 2004.

[22] D. Fetterly, “Adversarial Information Retrieval: the manipulation of Web content,” *ACM Computing Reviews*, July 2007.

[23] D. Fetterly, N. Craswell., and V. Vinay, "The impact of Crawl Policy on Web Search Effectiveness", *Proceeding of SIGIR'09*, Boston MA, USA, pp580-587, 2009. http://doi.acm.org/10.1145/1571941.1572041

[24] P. Graham, “A Plan for Spam”, article online, published August 2002. http://www.paulgraham.com/spam.html (accessed 23 january 2012).

[25] P. Graham, “Better Bayesian Filtering”, article online, published january 2003. http://www.paulgraham.com/better.html (accessed 23 january 2012).

[26] Z. I. Gyöngyi and H. Garcia-Molina, “Web spam taxonomy,” in *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 39–47, May 2005.

[27] Z. I. Gyöngyi, “Applications of Web link analysis,” *PhD thesis*, Stanford University, Adviser: Hector Garcia-Molina, 2008.

[28] H. Haddadi, “Fighting online click-fraud using bluff ads”, *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, ISSN 0146-4833, pp. 21-25, April 2010. http://doi.acm.org/10.1145/1764873.1764877

[29] A. Heydon and M. Najork, “Mercator: A scalable, extensible web crawler,” *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999.

[30] P. Heymann, G. Koutrika, and H. Garcia-Molina, “Fighting spam on social Web sites: A survey of approaches and future challenges,” *IEEE Internet Computing*, vol. 11, no. 6, pp. 36–45, 2007.

[31] T. Joachims and F. Radlinski, “Search Engines that Learn from Implicit Feedback”, *IEEE Computer,* Volume 40 (8), pp34 – 40, 2007. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4292009

[32] J. Köhne, “Optimizing a large dynamically generated Website for search engine crawling and ranking,” *Master's thesis*, Technical University of Delft, 2006.

[33] P. Kolari, “Detecting Spam Blogs: An Adaptive Online Approach,” *PhD thesis*, Department of Computer Science and Electrical Engineering, University of Maryland-Baltimore County, 2007.

[34] H.-T. Lee, D. Leonard, X. Wang, and D. Loguinov, “Irlbot: Scaling to 6 billion pages and beyond,” *ACM Transactions on the Web*, vol. 3, no. 3, pp. 1–34, 2009.

[35] Y.-R. Lin, H. Sundaram, Y. Chi, J. Tatemura, and L. B. Tseng, “Splog detection using self-similarity analysis on blog temporal dynamics,” in *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 1–8, New York, NY, USA: ACM Press, 2007.

[36] Y.-R. Lin, H. Sundaram, Y. Chi, J. Tatemura, and L. B. Tseng, “Detecting splogs via temporal dynamics using self-similarity analysis,” *ACM Transations on the Web*, vol. 2, no. 1, pp. 1–35, 2008.

[37] T. Liu, “Analyzing the importance of group structure in the Google Page- Rank algorithm,” *Master's thesis*, Rensselaer Polytechnic Institute, November 2004.

[38] Y. Liu, R. Cen, M. Zhang, S. Ma, and L. Ru, “Identifying web spam with user behavior analysis”, In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web (AIRWeb '08)*, Carlos Castillo, Kumar Chellapilla, and Dennis Fetterly (Eds.). ACM, New York, NY, USA, pp9-16, 2008. DOI=10.1145/1451983.1451986 http://doi.acm.org/10.1145/1451983.1451986

[39] J. Ma, K. L. Saul, S. Savage, and M. G. Voelker, “Beyond blacklists: Learning to detect malicious web sites from suspicious urls,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1245–1254, New York, NY, USA: ACM, 2009.

[40] K. Marks and T. Celik, “Microformats: The rel=nofollow attribute,” Technical Report, Technorati, 2005. Online at http://microformats.org/wiki/rel-nofollow. Last accessed 25 August 2011.

[41] J. Martineau, A. Java, P. Kolari, T. Finin, A. Joshi, and J. Mayfield, BlogVox: learning sentiment classifiers. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2* (AAAI'07), Anthony Cohn (Ed.), Vol. 2. AAAI Press 1888-1889, 2007.

[42] J. Martinez-Romo and L. Araujo, “Web spam identification through language model analysis,” in *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 21–28, ACM Press, 2009.

[43] K. Mason, “Detecting Colluders in PageRank: Finding Slow Mixing States in a Markov Chain,” PhD thesis, Department of Engineering Economic Systems and Operations Research, Stanford University, September 2005.

[44] G. Mishne, D. Carmel, and R. Lempel, “Blocking blog spam with language model disagreement,” in Proceedings of the *1st International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, May 2005.

[45] A. G. Mishne, “Applied Text Analytics for Blogs,” *PhD thesis*, University of Amsterdam, April 2007.

[46] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly, “Detecting spam Web pages through

content analysis," in *Proceedings of the 15th International Conference on World Wide Web (WWW)*, pp. 83–92, May 2006.

[47] A. Perkins, "The classification of search engine spam," Available online at http://www.silverdisc.co.uk/articles/spam-classification/, September 2001.

[48] J. Piskorski, M. Sydow, and D. Weiss, "Exploring linguistic features for Web spam detection: A preliminary study," in *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 25–28, New York, NY, USA: ACM, 2008.

[49] X. Qi, L. Nie, and D. B. Davison, "Measuring similarity to detect qualified links," in *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb),* pp. 49–56, May 2007.

[50] F. Salvetti and N. Nicolov. "Weblog Classification for Fast Splog Filtering: A URL Language Model Segmentation Approach". In P*roceedings of the Human Language Technology Conference of the NAACL*, Companion Volume: Short Papers, pages 137–140, New York City, USA, June 2006.

[51] Y. Sato, T. Utsuro, Y. Murakami, T. Fukuhara, H. Nakagawa, Y. Kawada, and N. Kando, "Analysing features of Japanese splogs and characteristics of keywords," in *Proceedings of the Fourth International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pp. 33–40, New York, NY, USA: ACM, 2008.

[52] D. Sheldon, "Manipulation of PageRank and Collective Hidden Markov Models," *PhD thesis*, Cornell University, 2009.

[53] A. Thomason, "Blog spam: A review," in *Proceedings of Conference on Email and Anti-Spam (CEAS)*, August 2007.

[54] T. Urvoy, E. Chauveau, P. Filoche, and T. Lavergne, "Tracking Web spam with HTML style similarities," *ACM Transactions on the Web*, vol. 2, no. 1, 2008.

[55] S. Webb, "Automatic Identification and Removal of Low Quality Online Information," *PhD thesis*, College of Computing, Georgia Institute of Technology, December 2008.

[56] B. Wu, "Finding and Fighting Search Engine Spam," *PhD thesis*, Department of Computer Science and Engineering, Lehigh University, March 2007.

[57] B. Zhou, "Mining page farms and its application in link spam detection," *Master's thesis*, Simon Fraser University, 2007.

[58] B. Zhou, J. Pei, and Z. Tang, "A spamicity approach to Web spam detection," in *Proceedings of the SIAM International Conference on Data Mining (SDM)*, April 2008.

[59] L. Zhu, A. Sun, and B. Choi, "Detecting spam blogs from blog search results", *Information Processing and Management* 47 pp246–262 , 2011.

Appendix A - Anti-spam API and example use cases

A.1 Akismet

A.1.1 Akismet python API

Located at http://www.voidspace.org.uk/python/akismet_python.html

A.1.2 Example use case for Akismet python API

```
api = Akismet(agent='Test Script')
# if apikey.txt is in place,
# the key will automatically be set
# or you can call ``api.setAPIKey()``
#
if api.key is None:
  print "No 'apikey.txt' file."
elif not api.verify_key():
  print "The API key is invalid."
else:
  # data should be a dictionary of values
  # They can all be filled in with defaults
  # from a CGI environment
  if api.comment_check(comment, data):
     print 'This comment is spam.'
  else:
     print 'This comment is ham.'
```

A.2 Mollom

A.2.1 Python wrapper for Mollom API

Located at https://github.com/itkovian/PyMollom

A.2.2 Example methods available for Mollom python API

- getServerList
- checkContent
- sendFeedback
- getImageCaptcha
- getAudioCaptcha
- checkCaptcha
- getStatistics
- verifyKey

A.3 Defensio

A.3.1 Python package for Defensio API

Located at http://www.defensio.com/downloads/python/

A.3.2 Example unit test code for Defensio python package[47]

From:https://github.com/defensio/defensio-python/blob/master/test/defensio_test.py

```
import unittest
import sys
sys.path.append('.')
from defensio import *

class TestDefensio(unittest.TestCase):

  def is_python3(self):
    return sys.version_info[0] == 3

  def setUp(self):
    # Set this to an actual key before running tests
    self.api_key = 'retrainer_key'
    self.client = Defensio(self.api_key)

  def testGenerateUrls(self):
    self.assertEqual("/2.0/users/" + self.api_key + ".json", self.client._generate_url_path())
          self.assertEqual("/2.0/users/" + self.api_key + "/action1.json"%locals(),
self.client._generate_url_path('action1'))
          self.assertEqual("/2.0/users/" + self.api_key + "/action1/id1.json"%locals(),
self.client._generate_url_path('action1', 'id1'))

  def testGetUser(self):
    status, result = self.client.get_user()
    self.assertEqual(200, status)
    self.assertEqual(dict, type(result['defensio-result']))
    result_body = result['defensio-result']
    self.assertEqual('success', result_body['status'])
    self.assertEqual('', result_body['message'])
    self.assertEqual('2.0', result_body['api-version'])

    if self.is_python3():
      self.assertEqual(str, type(result_body['owner-url']))
    else:
      self.assertEqual(unicode, type(result_body['owner-url']))

    self.assertTrue(len(result_body['owner-url']) > 0 )

  def testPostDocumentWhenFail(self):
    doc = {'content': 'Hi Hola Salut'}
```

47 The code is from: https://github.com/defensio/defensio-python/blob/master/test/defensio_test.py

```
    status, result = self.client.post_document(doc)
    self.assertEqual(200, status)
    self.assertEqual(dict, type(result['defensio-result']))
    result_body = result['defensio-result']
    self.assertEqual('fail', result_body['status'])
    self.assertEqual('The following fields are missing but required: platform, type', result_body['message'])
    self.assertEqual('2.0', str(result_body['api-version']))

  def testPostDocumentWhenSuccessThenPutThenGet(self):
    doc = {'content': 'Hi Hola Salut', 'type' : 'comment', 'platform' : 'python-test'}
    status, result = self.client.post_document(doc)
    self.assertEqual(200, status)
    self.assertEqual(dict, type(result['defensio-result']))

    result_body = result['defensio-result']
    self.assertEqual('success', result_body['status'])
    self.assertEqual('', result_body['message'])
    self.assertEqual('2.0', str(result_body['api-version']))
    self.assertAlmostEqual(0.05, result_body['spaminess'])
    self.assertEqual('legitimate', result_body['classification'])
    self.assert_(result_body['profanity-match'] == False or result_body['profanity-match'] == None)
    self.assertTrue(result_body['allow'])

    if self.is_python3():
      self.assertEqual(str, type( result_body['signature'] ))
    else:
      self.assertEqual(unicode, type( result_body['signature'] ))

    signature = result_body['signature']

    status, put_result = self.client.put_document(signature, {'allow' : 'false'})
    self.assertEqual(200, status)
    put_result_body = put_result['defensio-result']
    self.assertEqual('success', put_result_body['status'])

    status, get_result = self.client.get_document(signature)
    self.assertFalse(get_result['defensio-result']['allow'])

  def testProfanityFilter(self):
    doc = {'bad' : 'some fucking cursing here', 'good' : 'Hey... how is it going?'}
    status, res = self.client.post_profanity_filter(doc)
    self.failIfEqual(403, status, "Seems like the profanity filter is not enabled for key: " + self.api_key + " ")
    self.assertEqual(200, status)
    self.assertEqual('Hey... how is it going?', res['defensio-result']['filtered']['good'])
    self.assertEqual('some ****ing cursing here', res['defensio-result']['filtered']['bad'])

  def testBasicStats(self):
    status, res = self.client.get_basic_stats()
    self.assertEqual(200, status)
    result_body = res['defensio-result']
```

```
  self.assertEqual('success', result_body['status'])
   self.assertEqual(set(['status', 'false-positives', 'false-negatives', 'unwanted', 'legitimate', 'learning', 'api-version', 'learning-status', 'message', 'accuracy']), set(result_body.keys()))

 def testExtendedStats(self):
  data = {'from' : '2010-01-01', 'to' : '2010-01-04'}
  status, res = self.client.get_extended_stats(data)
  self.assertEqual(200, status)
  result_body = res['defensio-result']
  self.assertEqual('success', res['defensio-result']['status'])

 def testHandlePostDocumentAsyncCallback(self):
  if self.is_python3():
   handle_post_document_async_callback( b'{"defensio-result": {"status": "success"}}' )
  else:
   handle_post_document_async_callback( '{"defensio-result": {"status": "success"}}' )

if __name__ == '__main__':
 unittest.main()
```

A.4 TypePad Anti-Spam

The python package for TypePad Anti-Spam is provided through the Akismet API, and, therefore, there is no separate implementation. See Appendix A.1.

A.5 BlogSpam

The BlogSpam server is available through an XML-RPC based API. More information is available at http://blogspam.net/api/

A sample python code for accessing this API is provided below:

```
#! /usr/bin/python

from xmlrpclib import ServerProxy, Error

if __name__=='__main__':
        server=ServerProxy('http://test.blogspam.net:8888/')

        comment_details={
                        'ip':'1.2.3.4',
                        'email':'pvsnpnutter@nutters.com',
                        'name':'nutcase',
                        'comment':'some comment'
                        }
        try:
                print server.testComment(comment_details)
        except Error, v:
                print v
```

A.6 Akismet within Trac SpamFilter[48]

From:
http://trac.edgewall.org/browser/plugins/0.12/spam-filter-captcha/tracspamfilter/filters/akismet.py

```
# -*- coding: utf-8 -*-
#
# Copyright (C) 2005-2006 Edgewall Software
# Copyright (C) 2005-2006 Matthew Good <trac@matt-good.net>
# Copyright (C) 2006 Christopher Lenz <cmlenz@gmx.de>
# All rights reserved.
#
# This software is licensed as described in the file COPYING, which
# you should have received as part of this distribution. The terms
# are also available at http://trac.edgewall.com/license.html.
#
# This software consists of voluntary contributions made by many
# individuals. For the exact contribution history, see the revision
# history and logs, available at http://projects.edgewall.com/trac/.
#
# Author: Matthew Good <trac@matt-good.net>
#         Christopher Lenz <cmlenz@gmx.de>

from email.Utils import parseaddr
from urllib import urlencode
import urllib2
from pkg_resources import get_distribution

from trac import __version__ as TRAC_VERSION
from trac.config import IntOption, Option
from trac.core import *
from trac.mimeview.api import is_binary
from tracspamfilter.api import IFilterStrategy


class AkismetFilterStrategy(Component):
    """Spam filter using the Akismet service (http://akismet.com/).

    Based on the `akismet` Python module written by Michael Ford:
      http://www.voidspace.org.uk/python/modules.shtml#akismet
    """
    implements(IFilterStrategy)

                     noheaders     =     ['HTTP_COOKIE',     'HTTP_HOST',
'HTTP_REFERER','HTTP_USER_AGENT',
                   'HTTP_AUTHORIZATION']

    karma_points = IntOption('spam-filter', 'akismet_karma', '5',
        """By how many points an Akismet reject impacts the overall karma
of
        a submission.""")

    api_key = Option('spam-filter', 'akismet_api_key', '',
        """Wordpress key required to use the Akismet API.""")
```

48 From http://trac.edgewall.org/browser/plugins/0.12/spam-filter-captcha/tracspamfilter/filters/akismet.py

```
               api_url    =    Option('spam-filter',    'akismet_api_url',
'rest.akismet.com/1.1/',
        """URL of the Akismet service.""")

    user_agent = 'Trac/%s | SpamFilter/%s'  % (
        TRAC_VERSION, get_distribution('TracSpamFilter').version
    )

    def __init__(self):
        self.verified_key = None

    # IFilterStrategy implementation

    def is_external(self):
        return True

    def test(self, req, author, content, ip):
        if not self._check_preconditions(req, author, content):
            return

        try:
                    url  =  'http://%s.%scomment-check'  %  (self.api_key,
self.api_url)
            self.log.debug('Checking content with Akismet service at %s',
url)
            resp = self._post(url, req, author, content, ip)
            if resp.strip().lower() != 'false':
                self.log.debug('Akismet says content is spam')
                 return -abs(self.karma_points), 'Akismet says content is
spam'

        except urllib2.URLError, e:
            self.log.warn('Akismet request failed (%s)', e)

    def train(self, req, author, content, ip, spam=True):
        if not self._check_preconditions(req, author, content):
            return

        try:
            which = spam and 'spam' or 'ham'
             url = 'http://%s.%ssubmit-%s' % (self.api_key, self.api_url,
which)
                self.log.debug('Submitting %s to Akismet service at %s',
which, url)
            self._post(url, req, author, content, ip)

        except urllib2.URLError, e:
            self.log.warn('Akismet request failed (%s)', e)

    # Internal methods

    def _check_preconditions(self, req, author, content):
        if self.karma_points == 0:
            return False
```

```
        if not self.api_key:
            self.log.warning('Akismet API key is missing')
            return False

        if is_binary(content):
              self.log.warning('Content is binary, Akismet content check
skipped')
            return False

        try:
            if not self.verify_key(req):
                self.log.warning('Akismet API key is invalid')
                return False
            return True
        except urllib2.URLError, e:
            self.log.warn('Akismet request failed (%s)', e)

    def verify_key(self, req, api_url=None, api_key=None):
        if api_url is None:
            api_url = self.api_url
        if api_key is None:
            api_key = self.api_key

        if api_key != self.verified_key:
            self.log.debug('Verifying Akismet API key')
            params = {'blog': req.base_url, 'key': api_key}
            req = urllib2.Request('http://%sverify-key' % api_url,
                                  urlencode(params),
                                  {'User-Agent' : self.user_agent})
            resp = urllib2.urlopen(req).read()
            if resp.strip().lower() == 'valid':
                self.log.debug('Akismet API key is valid')
                self.verified = True
                self.verified_key = api_key

        return self.verified_key is not None

    def _post(self, url, req, author, content, ip):
        # Split up author into name and email, if possible
        author = author.encode('utf-8')
        author_name, author_email = parseaddr(author)
        if not author_name and not author_email:
            author_name = author
        elif not author_name and author_email.find("@") < 1:
            author_name = author
            author_email = None

        params = {'blog': req.base_url, 'user_ip': ip,
                  'user_agent': req.get_header('User-Agent'),
                  'referrer': req.get_header('Referer') or 'unknown',
                  'comment_author': author_name,
                  'comment_type': 'trac',
                  'comment_content': content.encode('utf-8')}
        if author_email:
            params['comment_author_email'] = author_email
        for k, v in req.environ.items():
```

```
            if k.startswith('HTTP_') and not k in self.noheaders:
                params[k] = v
        urlreq = urllib2.Request(url, urlencode(params),
                              {'User-Agent' : self.user_agent})

        #self.log.warn('AkismetPOST2 %s URL %s', urlencode(params), url)
        resp = urllib2.urlopen(urlreq)
        return resp.read()
```

Appendix B - Bayes and SVM-light for python

B.1 Example code: Bayes classifier training with python[49]

```
###script to train some data Paul Graham style###
import datetime
print datetime.datetime.now()

from nltk.corpus import movie_reviews
import random
from nltk import FreqDist
from operator import itemgetter
import pickle

pos_input = open('posTrainData', 'rb')
neg_input = open('negTrainData','rb')
#min_sample_size = 50 #for each category

pos_ids = pickle.load(pos_input)
neg_ids = pickle.load(neg_input)

pos_rev_num = len(pos_ids)
neg_rev_num = len(neg_ids)
total_rev_num = pos_rev_num + neg_rev_num
min_sample_size = 200 #change this to suit your purpose

if (pos_rev_num < min_sample_size) or (neg_rev_num < min_sample_size):
        print "your training sample is not large enough. come back when you have more!"

else:
        #select training samples
        train_pos_ids = random.sample(pos_ids,min_sample_size)
        train_neg_ids = random.sample(neg_ids,min_sample_size)
        #segregate test set
        #test_pos_ids = [i for i in pos_ids if not(i in train_pos_ids)]
        #test_neg_ids = [i for i in neg_ids if not(i in train_neg_ids)]

        #get words from reviews for training
        pos_rev_words = movie_reviews.words(fileids = train_pos_ids)
        neg_rev_words = movie_reviews.words(fileids = train_neg_ids)

        #vocab for both
        vocabulary = list(set(list(pos_rev_words) + list(neg_rev_words)))

        #get word frequency distribution
        pos_fdist = FreqDist(pos_rev_words)
        neg_fdist = FreqDist(neg_rev_words)
```

49 This example is Yunhyong Kim's own implementation of Paul Graham's approach to spam classification found at http://www.paulgraham.com/spam.html Because there was no spam data easily available the movie review corpus from the NLTK toolkit was used, pretending that negative reviews are spam.

```
        #create probaility table
        hasht={}
        for word in vocabulary:
                pn =0
                nn =0
                if word in pos_fdist.keys():
                        pn = pos_fdist[word]
                if word in neg_fdist.keys():
                        nn = neg_fdist[word]
                if nn + pn >5:
                        nquant = min(1,float(nn)/float(min_sample_size))
                        pquant = min(1,float(pn)/float(min_sample_size))
                        prob = max(0.1, min(0.99,float(nquant)/float(pquant+nquant)))
                        hasht[word] =prob

output1 = open('TrainResult', 'wb')
pickle.dump(hasht, output1)
output1.close()

#output2 = open('s250testData', 'wb')
#test = test_pos_ids + test_neg_ids
#pickle.dump(test, output2)
#output2.close()

print datetime.datetime.now()
```

B.2 Using SVM Light in Python[50]

B.2.1 A multi-class learner implementation example: multiclassify.py

```
"""A module for SVM^python for multiclass learning."""

# The svmlight package lets us use some useful portions of the C
code.
import svmlight

# These parameters are set to their default values so this
declaration
# is technically unnecessary.
svmpython_parameters = {'index_from_one':True}

def read_struct_examples(filename, sparm):
    # This reads example files of the type read by SVM^multiclass.
    examples = []
    sparm.num_features = sparm.num_classes = 0
    # Open the file and read each example.
    for line in file(filename):
        # Get rid of comments.
        if line.find('#'): line = line[:line.find('#')]
```

50 These scripts are from http://tfinley.net/software/svmpython1/

```
        tokens = line.split()
        # If the line is empty, who cares?
        if not tokens: continue
        # Get the target.
        target = int(tokens[0])
        sparm.num_classes = max(target, sparm.num_classes)
        # Get the features.
        tokens = [tuple(t.split(':')) for t in tokens[1:]]
        features = [(int(k),float(v)) for k,v in tokens]
        if features:
                       sparm.num_features  =  max(features[-1][0],
sparm.num_features)
        # Add the example to the list
        examples.append((features, target))
    # Print out some very useful statistics.
    print len(examples),'examples read with',sparm.num_features,
    print 'features and',sparm.num_classes,'classes'
    return examples

def loss(y, ybar, sparm):
    # We use zero-one loss.
    if y==ybar: return 0
    return 1

def init_struct_model(sample, sm, sparm):
    # In the corresponding C code, the counting of features and
    # classes was done in the model initialization, not here.
    sm.size_psi = sparm.num_features * sparm.num_classes
    print 'size_psi set to',sm.size_psi

def classify_struct_example(x, sm, sparm):
    # I am a very bad man.  There is no class 0, of course.
    return find_most_violated_constraint(x, 0, sm, sparm)

def find_most_violated_constraint(x, y, sm, sparm):
    # Get all the wrong classes.
    classes = [c+1 for c in range(sparm.num_classes) if c+1 is not
y]
    # Get the psi vectors for each example in each class.
    vectors = [(psi(x,c,sm,sparm),c) for c in classes]
    # Get the predictions for each psi vector.
    predictions = [(svmlight.classify_example(sm, p),c) for p,c in
vectors]
    # Return the class associated with the maximum prediction!
    return max(predictions)[1]

def psi(x, y, sm, sparm):
     # Just increment the feature index to the appropriate stack
position.
    return svmlight.create_svector([(f+(y-1)*sparm.num_features,v)
                                    for f,v in x])
```

```
# The default action of printing out all the losses or labels is
# irritating for the 300 training examples and 2200 testing
examples
# in the sample task.
def print_struct_learning_stats(sample, sm, cset, alpha, sparm):
     predictions = [classify_struct_example(x,sm,sparm) for x,y in
sample]
           losses = [loss(y,ybar,sparm) for (x,y),ybar in
zip(sample,predictions)]
    print 'Average loss:',float(sum(losses))/len(losses)

def print_struct_testing_stats(sample, sm, sparm, teststats): pass
```

B.2.2 How to train and test the classifier

Once you've written a Python module in the file multiclassify.py based on svmstruct.py and you want to use SVM^{python} with this module, you would use the following command line commands to learn a model and classify with a model respectively.

```
./svm_python_learn --m multiclassify [options] <train>
<model>
./svm_python_classify --m  multiclassify [options] <test>
<model> <output>
```

Note that SVM^{python} accepts the same arguments as SVM^{struct} plus this extra --m option. If the --m option is omitted it is equivalent to including the command line arguments --m svmstruct. Note that though we put this command line option first, the --m option may occur anywhere in the option list.