Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 6091385, 18 pages
http://dx.doi.org/10.1155/2016/6091385

Research Article

WSF2: A Novel Framework for Filtering Web Spam

J. Fdez-Glez, D. Ruano-Ordás, R. Laza, J. R. Méndez, R. Pavón, and F. Fdez-Riverola

Higher Technical School of Computer Engineering, University of Vigo, Polytechnic Building, Campus Universitario As Lagoas s/n, 32004 Ourense, Spain

Correspondence should be addressed to F. Fdez-Riverola; riverola@uvigo.es

Received 11 June 2015; Revised 26 October 2015; Accepted 12 November 2015

Academic Editor: Wan Fokkink

Over the last years, research on web spam filtering has gained interest from both academia and industry. In this context, although there are a good number of successful antispam techniques available (i.e., content-based, link-based, and hiding), an adequate combination of different algorithms supported by an advanced web spam filtering platform would offer more promising results. To this end, we propose the WSF2 framework, a new platform particularly suitable for filtering spam content on web pages. Currently, our framework allows the easy combination of different filtering techniques including, but not limited to, regular expressions and well-known classifiers (i.e., Naïve Bayes, Support Vector Machines, and C5.0). Applying our WSF2 framework over the publicly available WEBSPAM-UK2007 corpus, we have been able to demonstrate that a simple combination of different techniques is able to improve the accuracy of single classifiers on web spam detection. As a result, we conclude that the proposed filtering platform is a powerful tool for boosting applied research in this area.

1. Introduction

During the last years, the exploitation of communication networks to indiscriminately distribute unsolicited bulk information (known as spam) has introduced important limitations that prevent taking full advantage of the latest communication technologies for increasing personal productivity. In fact, some of the well-known obstacles introduced by the spam activity in the web (i.e., web spam) are as follows: (i) users spend their valuable time manually viewing multiple searching results and discarding irrelevant entries, (ii) known search engines lose their utility and large corporations such as Google Inc. spoil one of their business areas, and (iii) WWW (World Wide Web) would not be useful as a reliable information source.

Furthermore, with the passage of time, different forms of spamming have also emerged (e.g., e-mail spam, forum spam, spam chat bots, SMS spam, and/or web spam), generating newer and more complicated situations. During June 2013, the US Food and Drug Administration (FDA) detected 1677 illegal online drug stores trying to sell illicit medicines and seized more than 41 million dollars of merchandise [1]. This outcome was executed through the recent effort named Operation Pangea (VI), which targeted websites supplying fake and illicit medicines in 99 different countries. Moreover, recent studies [2, 3] showed inner business details and revenue estimations that exceeded one million dollars per month. These research works also clarify the benefits of sending mass advertisements (spam) to e-mail users and public forums, and the essentialness of using search engine optimization (SEO) based spamming techniques to promote these websites [4] and ensure revenue. The increment in both tax evasion and public health costs represents the main risks of this illegal business mainly supported by spamming activities.

Web spam (also known as spamdexing or black hat SEO) comprises the usage of any kind of manipulative techniques to fraudulently promote web sites, attaining false high ranking scores in search engines. Thus, when users search for a specific subject matter, some results are completely unrelated to their interests. Due to the major implications of web spam, Google Inc. founded a web spam team led by Matt Cutts [5] to fight against spam page indexation.

At the same time, different individual techniques were introduced in a parallel effort with the goal of fighting web spam [6, 7]. Although most of the proposed filtering methods are very effective under some scenarios, none of them provide a completely successful approach. Specifically related to the business of web indexing—in which Google is a clear example—the cost of false negative (FN) errors is particularly noteworthy because they entail the loss of relevant entries in search results, while false positive (FP) errors do not usually represent an important issue to end users. Moreover, and apart from some recent works [8, 9], most of the existing models are built in a static way, without any consideration about the evolving dynamic nature of web spam. Keeping this situation in mind, we believe that current available techniques could be easily combined into a unique filter that could take advantage of the individual strengths of each technique while partially overcoming their limitations. This idea has already been successfully applied in the e-mail spam filtering domain using products such as SpamAssassin [10].

In this work, we introduce a novel Web Spam Filtering Framework (WSF2) that can be successfully used to combine machine learning (ML) techniques and other nonintelligent approaches (e.g., regular expressions, black lists) to improve the detection of spam web sites. The design of this platform has been widely influenced by SpamAssassin and other effective rule-based antispam filtering systems [11], being easily extended through the use of plug-ins. WSF2 was deliberately conceived to accomplish two main objectives: (i) being able to train/test the performance of different techniques in a scientific environment and (ii) working in an interconnected way with a web crawling system to prevent the indexation of spam websites. WSF2 is an open-project, licensed under the terms of GNU LGPL (Lesser General Public License), publicly available at http://sourceforge.net/projects/wsf2c/.

After establishing the motivation of the present work, the rest of the paper is organized as follows: Section 2 presents an overview of previous related work on web spam filtering. Section 3 describes in detail the proposed WSF2 framework, covering its main design principles and the filter definition process. In order to demonstrate the suitability of the developed platform, Section 4 compiles the output of different experimental benchmarks and discusses the main results. Finally, Section 5 summarizes the main conclusions and delineates new directions for further research.

2. Related Work on Web Spam Filtering

In this section, we present a brief overview about existing techniques and initiatives especially devoted to the fight against web spam. As previously commented, the development of new methods for web spam filtering has gained importance for the software industry over the last several years. Despite the fact that there are strong similarities with spam e-mail, specific research in this domain has attracted a good number of scientists leading to the development of novel approaches for fighting web spam.

Although several taxonomies of web spam filtering methods have been proposed in literature [7, 12–14], these approaches can be roughly categorized into three main groups: (i) content-based techniques, (ii) link-based approaches, and (iii) hiding methods, of which content- and link-based are the most common approaches for web spam detection.

To begin, content-based web spam techniques analyse content features in web pages (e.g., popular terms, topics, keywords, or anchor text) to identify illegitimate changes which try to improve their ranking and increase their likelihood of being returned as a "normal" result of a given user search. Several techniques and different works have focused on this area. Among the earliest papers, Fetterly and colleagues [15, 16] statistically analysed content properties of spam pages, while Ntoulas et al. [17] used machine learning methods to detect spam content. More recently, Erdélyi and colleagues [18] presented a comprehensive study about how various content features and machine learning models can contribute to the quality of a web spam detection algorithm. As a result, successful classifiers were built using boosting, Bagging, and oversampling techniques in addition to feature selection [19–21].

Link spam is based on adding inappropriate and misleading association between web pages. It incorporates extraneous pages or creates a network of pages that are densely connected to each other in order to manipulate the built-in search engine ranking algorithm. In this context, the work of Davison [22] was the first to cope with the problem of link spam. Since then, different approaches have focused on link spam, analysing several ways to detect it [7].

The appropriate combination of link-based techniques and content-based methods can also be successfully applied to this problem. In fact, Geng and colleagues [23] introduced the first proposal using both content- and link-based features to detect web spam pages. In the same line, the work of Becchetti and colleagues [24] combined link- and content-based features using C4.5 to detect web spam. Complementarily, Silva and colleagues [25] also considered different methods of classification involving decision tree, SVN, KNN, LogitBoost, Bagging, and AdaBoost in their analyses. Other related approaches were also introduced [26, 27].

Additionally, hiding techniques are based on concealing the original high quality page from the user. Generally, this method consists of cloaking [28–31] and redirection [32, 33].

Summarizing the state of the art previously introduced, it can be concluded that research on web spam detection has evolved from simple content-based methods to more complex approaches using sophisticated link mining and user behaviour mining techniques.

Regarding the combination of different filtering spam techniques for web classification, only the use of large collections of different classifiers has been successfully applied [18], to the best of our knowledge. However, there is no configurable framework able to integrate diverse sets of existing techniques. Nowadays, there are providers of sophisticated enterprise-level security solutions such as WebTitan (http://www.webtitan.com/) or zVelo (http://zvelo.com/) that through their services (WebTitan Cloud and zVeloDB + zVeloCat, resp.) offer professional web filtering solutions to the industry area. However, these implementations are not suitable for research environments in which there is a lack of an appropriate framework supporting advanced functionalities.

3. WSF2: The Proposed Framework

As the central contribution of this work, we present our WSF2 software architecture and operational process in detail, together with its integration into a web crawler environment. The WSF2 design was straightforwardly inspired from our previously developed Wirebrush4SPAM platform [11], obtaining a novel framework able to provide flexible support for web page filtering using new available antispam techniques inside a completely readapted filtering process. Initially, Section 3.1 presents a comprehensive description of the WSF2 filtering process. Then, Section 3.2 describes the architecture of WSF2 and introduces the concept of spam filters, exemplifying how to develop them using the WSF2 filtering platform. Finally, Section 3.3 demonstrates the ease with which WSF2 can be integrated into both a real-time domain (e.g., business) and scientific environments.

3.1. Main Design Principles. WSF2 implements a framework and middleware for the development and execution of user-defined web spam filters. To support this functionality, WSF2 works as a daemon (wsf2d) listening on a specific TCP port in order to carry out a complete filtering cycle for each received web page. The diagram in Figure 1 describes all the stages involved in the filtering process and their associations with the classes implementing the main system architecture.

As we can observe in Figure 1(a), the main operation of our WSF2 framework is divided into five different stages: (i) filtering platform initialization, (ii) web domain analyser and information retrieval, (iii) spam filtering rules execution, (iv) spam decision system, and (v) learning after report.

The start-up phase (represented as stage 0 in Figure 1(a)) is instantly executed whenever the WSF2 framework is invoked (described in the wsf2d class of Figure 1(b)), handling the initialization of the filtering platform. During this process, all the rules comprising the WSF2 filter are loaded into the *ruleset* data structure represented in Figure 1(b) and sorted by a rule-scheduling algorithm. This rule planning module is implemented into the *prescheduler_t* data type and, as outlined in Ruano-Ordás and colleagues [34], it is responsible for elaborating an optional arrangement of the execution of the filtering rules in order to improve WSF2 classification throughput. Moreover, with the aim of reducing the filtering time, all available parsers, spam filtering techniques, and event-handlers are loaded into memory within this stage. When this phase is completed, the WSF2 core module (*wsf2d*) is able to start receiving and filtering web domains. Each time WSF2 receives new web domain, a four-stage filtering process is started (represented in Figure 1(a) by a circular operation denoted by rounded arrows).

During next stage, *wsf2d* executes all the previously loaded parsers over the web domain content for gathering the data needed by the selected filtering techniques. To perform this task, each parser must be implemented using the *parser_t* data type. As it can be observed in Figure 1(b), all the parsers provided by the filtering platform (such as *web_header*, *web_body*, or *web_features*) are defined as inheritance relationship from the abstract class *parser_t*. When this stage is actually accomplished and all the information is successfully extracted, the WSF2 platform automatically evolves to the following phase.

Stage 2 is responsible for executing the antispam techniques (implemented by a *function_t* data type) belonging to the filtering rules over the information extracted by the parsers. As it can be seen from Figure 1(b), in order to facilitate the development, implementation, and the automatic deployment of the antispam techniques, *function_t* module is implemented as a generic template able to adapt to the specification of each filtering function.

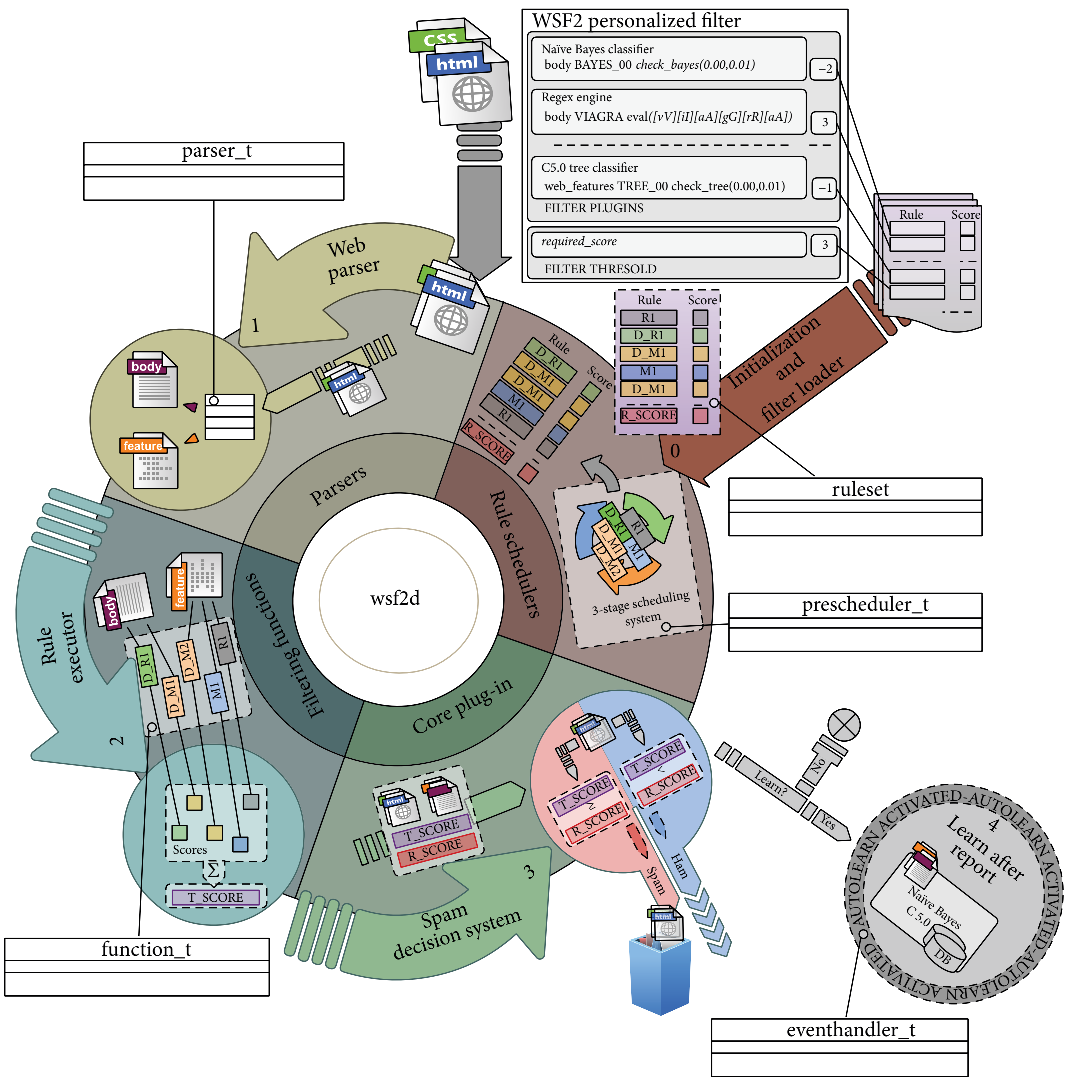
To accomplish the filtering task, the WSF2 framework implements a T_SCORE attribute (see stage 2 on Figure 1(a)) used to compute the global score achieved by the platform during the filtering process. Whenever an executed rule achieves a positive evaluation (i.e., its associated filtering technique matches the web content), the system automatically adds the rule score to the T_SCORE attribute.

With respect to the next stage, the filtering platform is in charge of generating a definite classification (spam or ham) depending on the final value of the T_SCORE attribute. To make the corresponding decision, the filtering platform compares the current value of T_SCORE attribute with the value defined by the *required_score* parameter (denoted as R_SCORE in Figure 1(a)). As we can observe from this stage, if the T_SCORE value is less than R_SCORE, the WSF2 platform automatically classifies the web domain as legitimate (ham); otherwise, it is classified as spam.

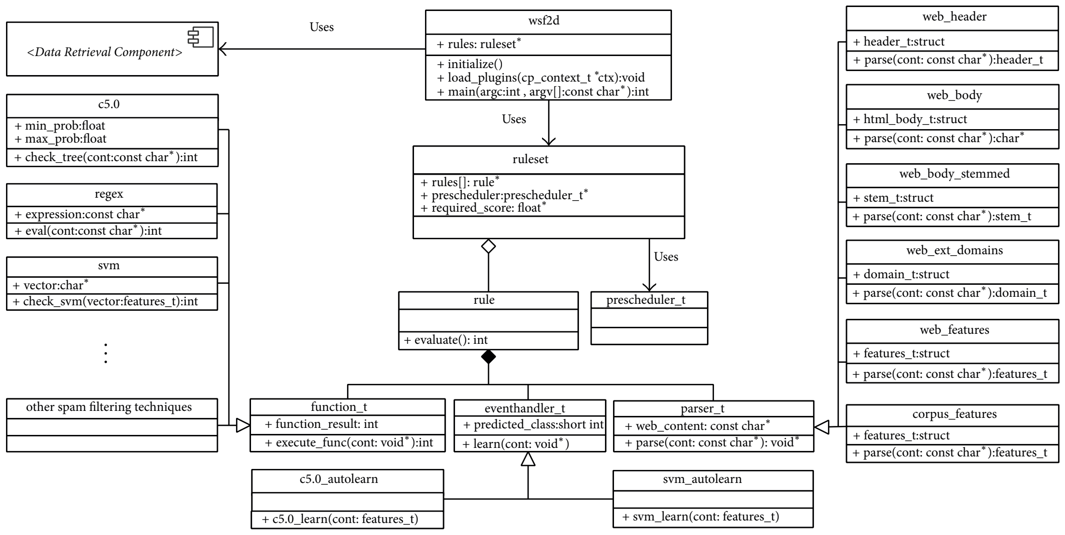
Additionally, a learning phase (stage 4 in Figure 1(a)) can be conditionally executed depending on the user preferences (defined in *the wsf2d_config.ini* file). If the autolearning module is activated, WSF2 will acquire new knowledge from current classification data. The learning process of each specific spam filtering technique should be implemented inside an *eventhandler_t* data type. As indicated in Figure 1(b), WSF2 actually provides learning methods only for C5.0 and SVM techniques. However, the flexibility provided by having inheritance relationships through the *eventhander_t* class enables an easy way to implement new learning tasks for the filtering techniques.

If the learning module is activated, the WSF2 framework traverses and executes (in separate threads) all the existing event-handlers. This operation mode (described in Pérez-Díaz and colleagues [11] as learning after report) allows the execution of learning tasks in background, avoiding the disruption of the filtering process.

3.2. WSF2 Filter Development. As previously commented, WSF2 acts as an antispam middleware and framework, enabling the automatic classification of web content guided by the development and execution of user-defined spam filtering rules. In line with Wirebrush4SPAM or SpamAssassin, a filter in WSF2 is composed by a set of scored rules together with a global threshold called *required_score* (denoted as R_SCORE



(a) WSF2 daemon (wsf2d) filtering process



<Data Retrieval Component>
Uses
wsf2d
+ rules: ruleset*
+ initialize()
+ load_plugins(cp_context_t *ctx):void
+ main(argc:int , argv[]:const char*):int
Uses
ruleset
+ rules[]: rule*
+ prescheduler:prescheduler_t*
+ required_score: float*
Uses
rule
+ evaluate(): int
prescheduler_t
c5.0
+ min_prob:float
+ max_prob:float
+ check_tree(cont:const char*):int
regex
+ expression:const char*
+ eval(cont:const char*):int
svm
+ vector:char*
+ check_svm(vector:features_t):int
⋮
other spam filtering techniques
function_t
+ function_result: int
+ execute_func(cont: void*):int
eventhandler_t
+ predicted_class:short int
+ learn(cont: void*)
parser_t
+ web_content: const char*
+ parse(cont: const char*): void*
c5.0_autolearn
+ c5.0_learn(cont: features_t)
svm_autolearn
+ svm_learn(cont: features_t)
web_header
+ header_t:struct
+ parse(cont: const char*):header_t
web_body
+ html_body_t:struct
+ parse(cont: const char*):char*
web_body_stemmed
+ stem_t:struct
+ parse(cont: const char*):stem_t
web_ext_domains
+ domain_t:struct
+ parse(cont: const char*):domain_t
web_features
+ features_t:struct
+ parse(cont: const char*):features_t
corpus_features
+ features_t:struct
+ parse(cont: const char*):features_t

(b) Main classes comprising the WSF2 platform

FIGURE 1: WSF2 framework.

Table 1: Description of inheritance relationships and classes in the WSF2 framework.

Parent class	Inherited class	Method signature	Method description
parser_t	web_header	get_params(): header_t	Gathers relevant information from the response header of the retrieved web site (e.g., status of the HTTP response or web-content encoding).
	web_body	get_content(): const char*	Extracts the content of the body of each web page (ignoring the HTML tags).
	web_body_stemmed	get_stem_words(): stem_t	Reduces the words to their root form, returning a list of stem terms together with their occurrence count inside a web page.
	web_ext_domains	get_domains(): domain_t	Returns information related to those domains linked from a given web site.
	web_features	get_features(): features_t	Builds a vector that contains a set of features extracted from the header and the content of the retrieved web site.
	corpus_features	get_features(): features_t	Retrieves all the preprocessed features of a corpus to an internal format use by our framework.
function_t	c5.0	check_tree(cont: const char*): int	Performs C4.5 tree over the content of the web domain.
	regex	eval(cont: const char*): int	Verifies if a specific regular expression matches the web content.
	svm	check_svm(vector: features_t): int	Executes SVM algorithm over the features extracted from the web domain.
eventhandler_t	c5.0_autolearn	c5.0_learn(cont: features_t)	Executes the learning tasks for C5.0 tree.
	svm_autolearn	svm_learn(cont: features_t)	Performs the learning method for SVM algorithm.

```
(00) parser_t RULENAME call_to: function_t
(01) describe RULENAME rule_description
(02) score RULENAME rule_score
```

Algorithm 1: Example definition of a common rule in the WSF2 framework.

in Figure 1(a)). In this regard, Algorithm 1 shows the typical structure of a rule in our WSF2 framework.

As it can be seen in Algorithm 1, each rule is defined by four keywords: (i) *parser_t* denotes the type of the web parser needed by the filtering plug-in, (ii) *call_to: function_t* represents the rule triggering condition (usually a call to a Boolean function that implements an antispam filtering technique), (iii) *describe* is optionally being used to introduce a brief description about the behaviour of the rule, and (iv) *score* determines the value to be added to the total score attribute (T_SCORE) if the filtering technique associated with the rule matches the target web domain.

It is important to outline that the interconnection between the rule definition shown in Algorithm 1 and the WSF2 main architecture described in Figure 1(b) provides a great versatility in the development and deployment of new filtering techniques due to the inheritance relationships between classes that allow the modelling of each functionality offered by WSF2 (i.e., parsers, filtering functions, and event-handlers) as separate plug-ins, making it possible to dynamically interact and manage each plug-in as an independent entity. To accomplish this goal, *parser_t* and *function_t* implement a method able to associate the parser (using the *parse()* function implemented inside *parser_t* class) and the filtering technique (using *execute_func()* method allocated inside *function_t* class) specified in the definition of the rule.

Moreover, in our WSF2 framework, learning algorithms are related to antispam filtering techniques (instead of the rules). To this end, whenever a rule is loaded, WSF2 automatically checks if its associated filtering technique provides a learning method. In this case, WSF2 performs the following operations: (i) associating the *eventhandler_t* class with the learning method by a function pointer between the *learn()* function (implemented in *eventhandler_t* class) and the filtering technique and (ii) loading in memory the *eventhandler_t* data structure for subsequent execution.

In order to facilitate the understanding of the inheritance relationships between those classes shown in Figure 1(b) and Algorithm 1, Table 1 presents a summary highlighting the most relevant aspects of each one, together with a brief description of their methods.

Taking into account (i) all the data types and classes involved in the WSF2 rule execution system shown in Algorithm 1, (ii) the purpose of the keywords previously presented in Figure 1, and (iii) the summary of classes presented in Table 1, we can conclude that the WSF2 filtering rule behaviour is straightforward.

In order to clarify the association between filtering rules and the inherited classes included in our platform, we present in Algorithm 2 an example of a dummy filter coded for the WSF2 framework.

As we can see from Algorithm 2, the example filter is composed of five different rules together with the unique

```
(00) web_features SVM check_svm();
(01) describe SVM Classifies a web page as spam using Support Vector Machine classifier
(02) score SVM 3
(03)
(04) web_features TREE_95 check_tree(0.95, 0.99);
(05) describe TREE_99 C5.0 between 0.99 and 1.00
(06) score TREE_99 1.5
(07)
(08) web_features TREE_99 check_tree(0.99, 1.00);
(09) describe TREE_99 C5.0 between 0.99 and 1.00
(10) score TREE_99 3
(11)
(12) web_body HAS_VIAGRA_ON_WEB_BODY eval("[vV][iI?1!][aA][gG][rR][aA]")
(13) describe HAS_VIAGRA_ON_WEB_BODY Check if the web page contains references to viagra on body
(14) score HAS_VIAGRA_ON_WEB_BODY 2
(15)
(16) meta HAS_HIGH_SPAM_RATE (SVM & (TREE_95 || TREE_99))
(17) describe HAS_HIGH_SPAM_RATE Has high probability of being spam
(18) score HAS_HIGH_SPAM_RATE +
(19)
(20) required_score 5
```

Algorithm 2: Dummy filter definition for the WSF2 platform.

required_score field. The first rule (SVM) is applied to the most relevant features extracted from the web content by using the *web_features* parser. Then, there are defined two additional rules that cope with the execution of the C5.0 algorithm over the same features used by the SVM classifier. Each C5.0 rule is in charge of verifying if the execution of C5.0 algorithm is contained inside a specific probability interval (defined by the user as function parameters in lines (04) and (08)). Following that, the definition of the fourth rule (HAS_VIAGRA_ON_WEB_BODY) involves the execution of regular expressions applied to the web page content. As we can observe from line (12), this rule is triggered every time the web page contains the word "viagra." Finally, in line (16), a special type of rule (META) is introduced. This kind of rule is used to properly combine the results of other types of rules using Boolean expressions and mathematical operators. In the example, the rule HAS_HIGH_SPAM_RATE (lines (16) to (18)) incorporates a Boolean expression integrating previously commented SVM, TREE_95, and TREE_99 rules. Following the proposed scheme, if the Boolean expression is true, the score associated with the META rule is added to the total score of the web page.

Additionally, an important aspect to keep in mind when defining a filter is that the WSF2 platform allows the characterization of rules with both numerical and definitive scores. A definitive score is used to abort the execution of the filtering process at any time, carrying out the classification of the web page depending on the symbol associated with the score value (i.e., "+" for spam and "–" for ham). In the example shown in Algorithm 2, and taking into account the use of definitive scores (line (18)), we can conclude that if the HASH_HIGH_SPAM_RATE rule is triggered, the whole filtering process will be automatically aborted, classifying the web page as spam.

3.3. Integrating the WSF2 Framework into a Standard Web Crawler Architecture.

Our WSF2 platform has been entirely coded in ANSI/C language which guarantees an adequate filtering speed and a satisfactory throughput. Moreover, with the aim of providing a versatile platform able to be easily adapted to both academic (research) and enterprise environments, WSF2 implements two different interfaces: a storage communication interface (SCI) and a crawler communication interface (CCI). SCI is designed to enable the possibility of loading web contents from a previously compiled corpus, avoiding the need of executing WSF2 inside a crawler. CCI complementarily allows a straightforward management of the communication between the crawler and our WSF2 system for a real-time web filtering operation.

In this context, Figure 2 introduces a detailed class diagram showing the interconnection of both interfaces and their role inside our WSF2 platform. As Figure 2 shows, WSF2 also implements the WCM (WSF2 *Communication Manager*) module that is in charge of correctly handling the connection between CCI and SCI interfaces. In particular, this module is responsible for translating the information provided by SCI and CCI interfaces into an input stream ready to be processed by the filtering platform. To perform this task, the WCM module implements two methods: *get_from_crawler* that obtains and parses the information provided by CCI and *get_from_storage* that accesses web contents from a defined corpus.

Additionally, as we can observe from Figure 2, the CCI interface provides two methods: *receive_web_content*, which is responsible for obtaining all the entries from the crawler, and *send_web_content*, which is in charge of giving legitimate web content back to the crawler for normal operation. SCI implements three complementary methods to enable the offline operation of the WSF2 platform: *load_stored_files*,

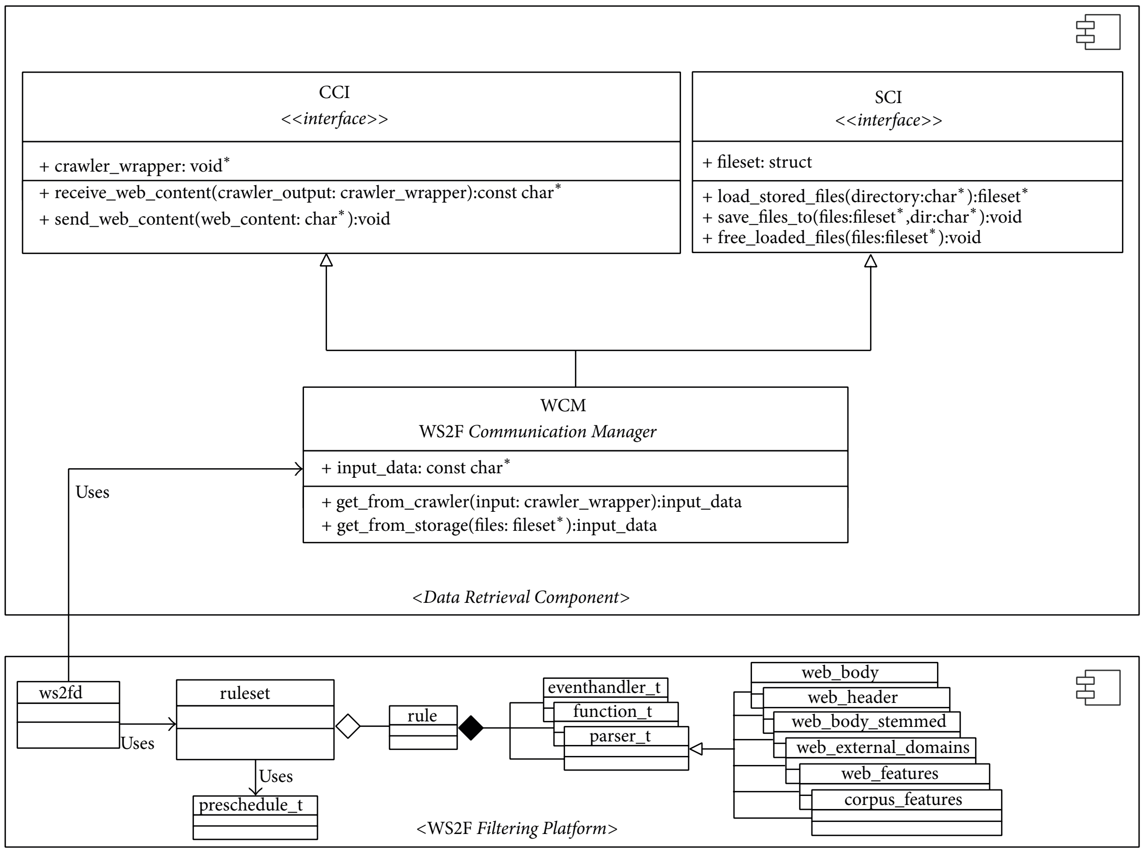

CCI
<<interface>>
+ crawler_wrapper: void*
+ receive_web_content(crawler_output: crawler_wrapper):const char*
+ send_web_content(web_content: char*):void
SCI
<<interface>>
+ fileset: struct
+ load_stored_files(directory:char*):fileset*
+ save_files_to(files:fileset*,dir:char*):void
+ free_loaded_files(files:fileset*):void
WCM
WS2F Communication Manager
+ input_data: const char*
+ get_from_crawler(input: crawler_wrapper):input_data
+ get_from_storage(files: fileset*):input_data
Uses
<Data Retrieval Component>
ws2fd
Uses
ruleset
Uses
preschedule_t
rule
eventhandler_t
function_t
parser_t
web_body
web_header
web_body_stemmed
web_external_domains
web_features
corpus_features
<WS2F Filtering Platform>

FIGURE 2: Class diagram showing the implementation of CCI and SCI interfaces and their role inside WSF2 platform.

which retrieves and allocates all the web contents from a user-defined corpus path into a fileset structure; *save_files_to*, which stores the content of the fileset structure into a WARC (http://www.iso.org/iso/catalogue_detail.htm?csnumber=44717) (Web ARChive) file; and *free_loaded_files*, which cleans all the allocated resources from memory.

In order to complement operational details concerning our WSF2 platform, Figure 3 shows how the platform can be integrated into both research (offline filtering mode) and enterprise (real-time filtering) environments.

In case of a real-time web filtering deployment, the CCI interface enables WSF2 to be smoothly integrated inside the internal workflow of any web crawler. As pointed out in some works [35–37], crawler systems (also called web spiders or web robots) systematically browse the WWW with the goal of indexing existing web pages. Usually, web searches make use of web robots to both update their own content and perform the indexation of third-party web documents. Additionally, web crawlers can make a copy of visited pages with the goal of delaying their processing by a search engine. Regarding this situation, crawlers are usually composed of different modules [37] (i.e., downloader, DNS resolvers, and crawler application) allowing components being instantiated more than once (operating in parallel). Therefore, making local copies of visited pages is a mandatory feature and enables web crawlers to operate faster.

As we can see at the top of Figure 3, the web crawling process starts from a set of initial URLs pending to be processed (also called seeds). For each URL (i.e., web page), the crawler parser uses the extractors to perform (i) the identification and storage of text and metadata and (ii) the URL retrieval. After the content of each URL is collected, the crawler checks whether this location was previously processed in order to prevent adding multiple instances of the same hyperlink to the queue of pending URLs (frontier in Figure 3). The frontier should allow the prioritization of certain URLs (e.g., those referring to continually updated web sites) because the large amount of URLs available through Internet impedes the crawler from indexing the whole content within a given time period. Therefore, each new prioritized URL added to the frontier will wait its turn to be downloaded and inspected recursively by the crawler operational process.

Moreover, as we can observe from Figure 3, the CCI component runs autonomously from the web crawler, therefore circumventing the redesign of its architecture. This fact has the added advantage of both avoiding a negative impact on the system performance and preventing modifications in the execution scheme of the web crawler. To accomplish its

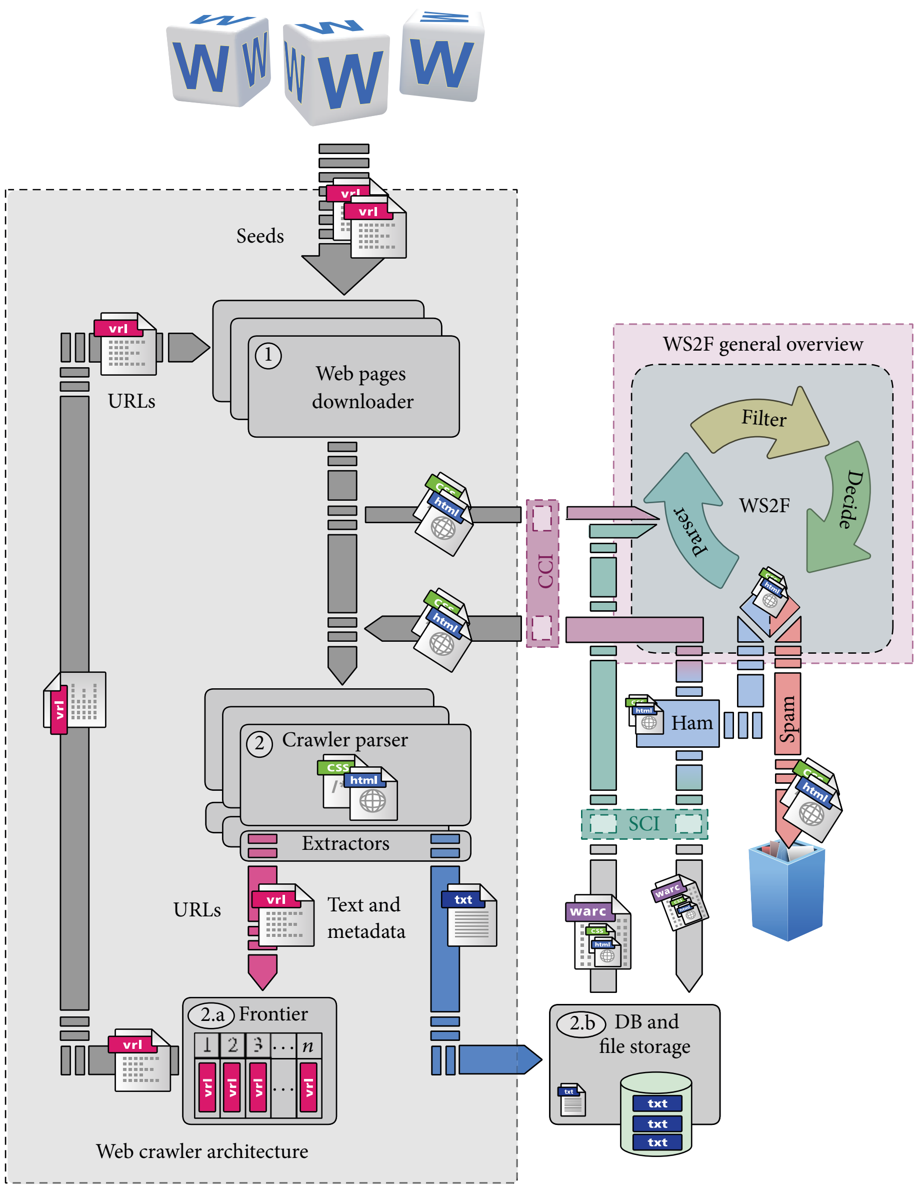


FIGURE 3: General web crawler architecture and WSF2 runtime representation.

purpose, the CCI component is embedded between the first and second stages of the web crawler workflow. Hence, the CCI component transforms the output obtained from the crawler downloader (see stage 1 in Figure 3) to a ready-to-be-processed WSF2 input data (also called forward translation). When the filtering process finishes with the location finally classified as ham (legitimate), the CCI component transfers the WSF2 output to a data structure of the web crawler (reverse translation).

Finally, as shown in Figure 3, the SCI component enables the execution of the WSF2 platform under a research (more academic) domain. In this context, the SCI component is responsible for establishing a connection between a web corpus (usually stored using a special structure called WARC) and the WSF2 data input. The operational process of a SCI component is similar to the behaviour of CCI. Under this scenario, SCI accomplishes a special preparsing operation responsible for traversing each file structure in order to identify and retrieve (i) file metadata values and (ii) all the web content information necessary by the filtering platform to carry out the classification process.

4. Case Study

With the goal of demonstrating the suitability of the proposed framework for filtering web spam content, we have designed and executed a set of experiments involving a publicly available corpus and different classification techniques. Considering the web spam domain from a machine learning perspective, we analysed the behaviour of two well-known state-of-the-art algorithms for web spam classification (i.e., SVM and C5.0) comparing their performance first as individual classifiers and then as hybridized classifiers (using regular expressions) in our WSF2 framework. These classifiers were selected because of their effectiveness and relative efficiency as evidenced in previous research works [17, 23, 38, 39]. Although regular expressions used as an individual technique achieve poor results in spam filtering, their proper combination with other machine learning approaches improves the accuracy of definitive antispam classification. This occurrence, combined with the fact that regular expressions are commonly used in the business environment, makes our WSF2 platform a resource of great value for both academia and industry.

The selected corpus, together with the data preprocessing carried out, is introduced in Section 4.1. Section 4.2 describes core aspects related to the experimental protocol followed during the tests. Section 4.3 presents and discusses the results obtained from the experiments. Finally, Section 4.4 describes in detail current limitations of our WSF2 framework.

4.1. Corpus Selection and Data Preprocessing. The experience gained over the years by the web spam community put in evidence the need for a reference collection that could both guarantee the reproducibility of results and assure the correct comparison of novel approaches. A reference collection specifically designed for web spam research (WEBSPAM-UK2006) was first introduced in Castillo and colleagues [40]. Later, an enhanced version was labelled by a group of volunteers, building the publicly available WEBSPAM-UK2007 version of the corpus. Finally, during 2011, Wahsheh and colleagues [41] compiled an updated version of the *Webb Spam Corpus 2006* by only taking into consideration active links.

In early 2010, the ECML/PKDD Discovery Challenge on Web Quality also created the DC2010 dataset [8]. Additionally, in Webb and colleagues [42], a novel method for automatically obtaining web content pages was presented, resulting in the generation of the Webb Spam Corpus 2006, the first public dataset of this kind. Additionally, during 2011, this corpus was updated by deleting all unavailable web pages [43]. Table 2 summarizes the main characteristics of these accessible datasets.

As we can observe from Table 2, the main drawback of Webb Spam Corpus 2006 and 2011 lies in the lack of a collection of ham domains needed to perform the experimental protocol explained in next subsection. Additionally, the high unbalanced ratio between ham and spam characterizing the *DC2010* corpus (with only 3.2% spam pages) could provide unreal statistical outcomes after performing the experiments. Therefore, the set of *WebSPAM-UK* corpora are the best standard alternatives to use in our work.

In detail, although at first sight it might appear that *WebSPAM-UK2011* is the best option for being the most up-to-date dataset, the lack of a significant number of web pages (only 3,766) turns it into an unfeasible alternative. Therefore, we finally selected the *WebSPAM-UK2007* corpus mainly due to its extensive use by the scientific community in most of the web spam research works [9, 18, 27, 38, 40] together with its completeness in terms of (i) the number of web pages compiled (up to 114,529 hosts, of which 6,479 are labelled using three different categories: spam, ham, and undecided) and (ii) the inclusion of raw HTML for web pages, which allows for preserving their original format. Additionally, these HTML pages are stored in WARC format, so a given domain is composed of several WARC files. Coinciding with the Web Spam Challenge 2008 [44], existing labels were separated into two complementary groups (i.e., train and test). Table 3 presents a description of this corpus.

It is important to keep in mind that those instances belonging to the undecided category cannot be used in our biclass (i.e., spam or ham) classification system. Another aspect to consider is the existence of domains containing pages with empty or meaningless content, such as redirections to error pages. Therefore, it is mandatory to preprocess the whole corpus in order to remove those useless domains. Table 4 shows the final distribution of each group used in the experiments carried out in the present work.

As we can observe from Table 4, the resulting corpus is unbalanced, containing 5,797 valid domains asymmetrically distributed (i.e., 321 spam and 5,476 legitimate) with an imbalance rate of 1 : 17. This result represents a common problem in many practical applications of machine learning, as it is also present in web spam filtering. In our problem domain, the troublesome situation is mainly characterized by the existence of a large amount of irrelevant pages with respect to those sites holding valuable contents.

4.2. Experimental Protocol. In order to demonstrate the utility of our WSF2 platform, the experiments carried out were focused on validating the framework by combining different well-known classifiers and regular expressions, with the goal of improving the accuracy of single techniques on web spam detection. In an effort to facilitate the understanding of the whole process, we separated the experimental protocol into two different stages: (i) the application of a web content resampling strategy to alleviate the effects of the class imbalance problem and (ii) the execution of different antispam techniques, either individual or combined, to test their accuracy of web content classification.

As commented above, we first applied a resampling strategy in order to reduce the skewed distribution of the selected corpus (described in Table 4). For this purpose, we used the random undersampling method proposed by Castillo and colleagues [44] given both its ease of use and good performance. In general, this method is based on randomly eliminating some instances from the majority class in order to achieve the desired ratio between classes. With the aim of reproducing different balancing scenarios in a straightforward manner, we executed the experiments of the second phase using five different configurations (i.e., 1 : 17, 1 : 8, 1 : 4, 1 : 2, and 1 : 1). Complementarily, with the goal of obtaining sound conclusions, all the experiments were repeated 10 times using random undersampling, guaranteeing that the training set is different in each round. The results presented in this work correspond to the average values obtained in the 10 independent iterations.

Table 2: Available datasets for web spam research.

Corpus name	Hosts	Pages	Domain crawled	%Spam	Available at
WebSpam UK2006	11,400	77.9M	United Kingdom (.uk)	26%	http://chato.cl/webspam/datasets/uk2006/
WebSpam UK2007	114,529	105M	United Kingdom (.uk)	5.30%	http://chato.cl/webspam/datasets/uk2007/
WebSpam UK2011	n/a	3,766	United Kingdom (.uk)	53%	https://sites.google.com/site/heiderawahsheh/home/web-spam-2011-datasets/uk-2011-web-spam-dataset
DC2010	99,000	23M	Europe (.eu)	3.2%	https://dms.sztaki.hu/en/letoltes/ecmlpkdd-2010-discovery-challenge-data-set
Webb Spam Corpus 2006	n/a	350,000	Links found in millions of spam e-mails	100%	http://www.cc.gatech.edu/projects/doi/WebbSpamCorpus.html
Webb Spam Corpus 2011	n/a	330,000	Links found in millions of spam e-mails	100%	http://www.cc.gatech.edu/projects/doi/WebbSpamCorpus.html

Table 3: Description of the Web Spam Challenge 2008 dataset.

	Spam	Ham	Undecided	
Training set	222	3,776	277	**4,275**
Test set	122	1,933	149	**2,204**
	344	**5,709**	**426**	**6,479**

Table 4: Preprocessed dataset used in the experiments of our WSF2 platform.

	Spam domains	Ham domains	
Training set	208	3,641	**3,849**
Test set	113	1,835	**1,948**
	321	**5,476**	**5,797**

In the second stage of our experimental protocol, different combinations of the selected classifiers (i.e., SVM and C5.0) together with regular expressions were tested in order to analyse their accuracy and global performance. In particular, the input of the SVM and C5.0 classifiers was a vector comprising 96 content features already included in the selected corpus [45], while the input used for regular expressions was the raw content of all pages belonging to each domain. As discussed in [45], the content-based feature vector used is formed by aggregating the 24-dimensional content-based attribute vector of each page taking into consideration (i) the home page, (ii) the page with the largest PageRank, and both (iii) the average and (iv) variance of all pages (i.e., $24 \times 4 = 96$ content features). In detail, the 24 attributes belonging to each page are the following: number of words in the page, number of words in the title, average word length, fraction of anchor text, fraction of visible text, compression rate, k-corpus precision and k-corpus recall ($k = 100$, 200, 500, and 1000), q-query precision and q-query recall ($q = 100$, 200, 500, and 1000), independent trigram likelihood, and entropy of trigrams. Both SVM and C5.0 classifiers were configured by default. Each classifier was separately executed in order to evaluate its individual performance. The obtained values were subsequently used as a basis for the comparison with their integration in the WSF2 platform, both with and without the use of regular expressions.

With the goal of directly comparing the results obtained from the experiments carried out, we use different receiver operating characteristic (ROC) analyses including the area under curve (AUC), sensibility, and specificity. This type of validation has been widely used by the spam filtering research community [18, 26, 44] because it underscores the theoretical capacity of a given classifier (in terms of sensitivity and 1−specificity) regardless of the cut-off point. In the same line, specific measures more focused on accuracy (e.g., precision, recall, and f-score) are not suitable when dealing with unbalanced data, since they do not consider the proportion of examples belonging to each class and therefore do not provide information about the real cost of the misclassified instances. Moreover, they are also sensitive to a chosen threshold, and thus they do not guarantee a reliable comparison concerning the global behaviour of the analysed models.

```
(a) Filter definition for C5.0 classifier
    (00) web_features TREE check_tree(0.50, 1.00)
    (01) describe TREE Classifies a web using C5.0 classifier
    (02) score TREE 5
    (03)
    (04) required_score 5
(b) Filter definition for SVM classifier
    (00) web_features SVM check_svm()
    (01) describe SVM Classifies a web using SVM classifier
    (02) score SVM 5
    (03)
    (04) required_score 5
```

Algorithm 3: WSF2 filter definition for C5.0 and SVM classifiers.

Table 5: AUC results for C5.0 and SVM classifiers under different balancing conditions.

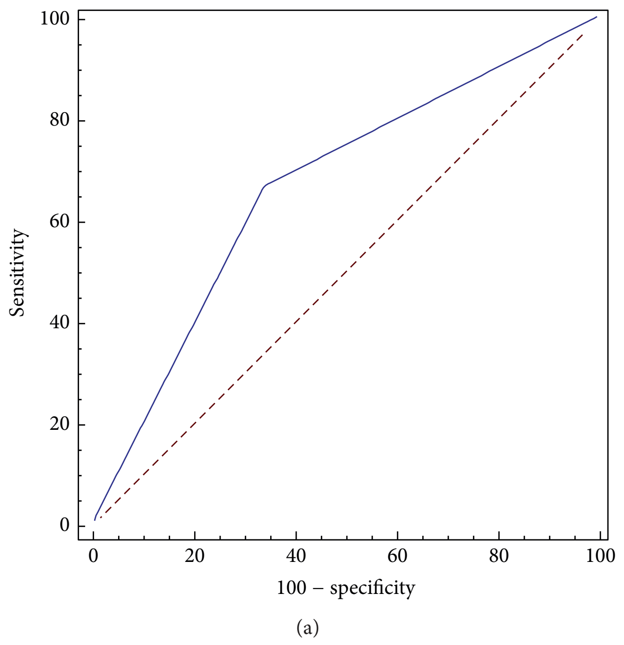
	Class-imbalance ratio				
	1 : 17	1 : 8	1 : 4	1 : 2	1 : 1
C5.0	*0.562*	0.649	**0.651**	*0.648*	*0.573*
SVM	0.534	*0.590*	0.602	0.604	**0.624**

4.3. Results and Discussion. As previously stated, in order to directly compare the outcomes generated from the different configurations, our benchmarking protocol was structured into three different scenarios consisting of (i) individually running each ML technique, (ii) combining these techniques using our WSF2 platform, and (iii) augmenting the second scenario with the incorporation of regular expressions to the WSF2 platform.

Under the first scenario, each algorithm (i.e., SVM and C5.0 classifiers) was executed separately in our WSF2 platform by defining the two elementary filters shown in Algorithm 3.

As we can observe from Algorithm 3, both filters are characterized by the same structure with the only exception being the rule definition (see line (00)). Algorithm 3(a) specifies a filter for triggering the C5.0 classifier, while Algorithm 3(b) introduces a filter involving the execution of the SVM algorithm. It is important to notice that the individual scores assigned to both rules (introduced in line (02)) are the same as the global *required_score* of the filter. This configuration guarantees that the execution of the filtering process is automatically aborted when the underlying algorithm matches the web content. AUC results obtained by each classifier under different balancing conditions are shown in Table 5.

As we can see in Table 5, the C5.0 classifier attains the highest score for AUC (0.651) when using an undersampling ratio of 1 : 4. Nevertheless, the SVM best score for AUC (0.624) is provided when the amounts of ham and spam documents are the same. From these results, we can conclude that C5.0 classifier is less sensitive than the SVM algorithm when dealing with unbalanced data. Figure 4 shows the best ROC curve achieved by both classifiers.



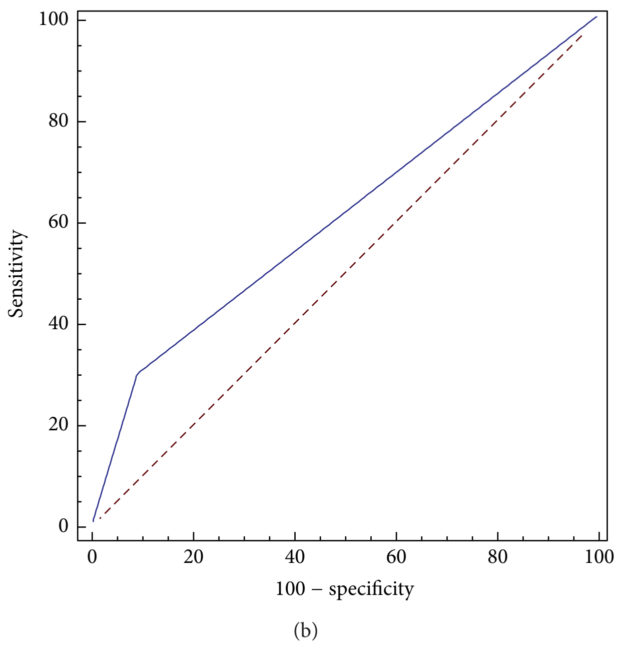


Figure 4: ROC curves for C5.0 and SVM classifiers individually executed. (a) ROC curve for C5.0 classifier (AUC = 0.651). (b) ROC curve for SVM classifier (AUC = 0.624).

```
(00) web_features SVM check_svm()
(01) describe SVM Classifies a web page as spam using Support Vector Machine classifier
(02) score SVM 5
(03)
(04) web_features TREE_00 check_tree(0.0, 0.25)
(05) describe TREE_00 Classifies a web page as spam if C5.0 probability is between 0.0 and 0.25
(06) score TREE_00 −1
(07)
(08) web_features TREE_25 check_tree(0.25, 0.50)
(09) describe TREE_25 Classifies a web page as spam if C5.0 probability between 0.25 and 0.50
(10) score TREE_25 3
(11)
(12) web_features TREE_50 check_tree(0.50, 0.75)
(13) describe TREE_50 Classifies a web page as spam if C5.0 probability between 0.50 and 0.75
(14) score TREE_50 4
(15)
(16) web_features TREE_75 check_tree(0.75, 1.00)
(17) describe TREE_75 Classifies a web page as spam if C5.0 probability between 0.75 and 1
(18) score TREE_75 5
(19)
(20) required_score 5
```

Algorithm 4: WSF2 filter definition for C5.0 and SVM algorithms working as a unique classifier.

Taking into consideration the AUC values displayed in Table 5 and the ROC curves for both classifiers shown in Figure 4, we can state that the C5.0 algorithm exhibits a better performance than the SVM. However, neither is good enough to be used as a single algorithm for detecting and filtering spam web content.

Given the fact that there is room for improvement, and taking advantage of the potential to combine different antispam techniques provided by our WSF2 platform, the second scenario investigates the suitability of hybridizing C5.0 and SVM classifiers into a unique filter. Algorithm 4 shows the definition of the WSF2 filter used to jointly execute both classifiers.

As shown in Algorithm 4 (lines (04) to (18)), the C5.0 classifier has been divided into four intervals (one per rule) in order to cover all the possible ranges of probabilities. Moreover, each interval is associated with a different score value, which varies depending on whether there is spam. According

Table 6: AUC results for C5.0 and SVM classifiers working as a unique classifier.

	Class-imbalance ratio				
	1 : 17	1 : 8	1 : 4	1 : 2	1 : 1
C5.0	*0.562*	0.649	**0.651**	*0.648*	*0.573*
SVM	*0.534*	0.590	0.602	*0.604*	**0.624**
C5.0 + SVM	*0.579*	0.658	**0.713**	*0.684*	*0.646*

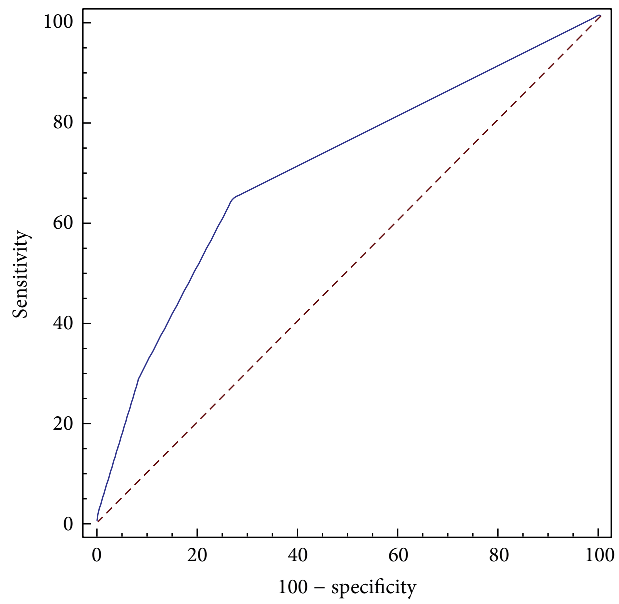


Figure 5: ROC curve for C5.0 and SVM classifiers combined in a unique classifier (AUC = 0.713).

to this circumstance, those C5.0 rules with intervals of low probability of spam have been assigned lower scores.

Table 6 presents the AUC results obtained by jointly executing both C5.0 and SVM classifiers (using the filter introduced in Algorithm 4) in addition to the results shown in Table 5 in order to easily compare the performance of each scenario.

As we can observe from Table 6, the simple combination of both classifiers achieves a better result than their individual counterparts in all the balancing conditions. In this regard, it is important to notice that although the individual execution of the SVM algorithm attained its best result with a 1 : 1 ratio, the combination of both classifiers exhibits a better performance under a 1 : 4 ratio. Figure 5 shows the best ROC curve achieved by the combination of classifiers.

Finally, in the last scenario, we measured the global performance achieved by the combination of both classifiers together with the use of regular expressions. To accomplish this task, and starting from the filter introduced in Algorithm 4, we defined the filter presented in Algorithm 5.

As we can observe from Algorithm 5, the filter contains 14 new rules (lines (00) to (56)) associated with the use of regular expressions. Additionally, it is important to notice that the first 7 rules are assigned to nonnumeric values (lines (00) to line (26)). As previously commented, these types of rules are defined as definitive rules and are used in order to take advantage of the Smart Filter Evaluation (SFE) feature of WSF2. This functionality is inherited from our previous Wirebrush4SPAM platform [11] and enables the possibility of interrupting the execution of the filtering process when a definitive rule matches the content. Every time the filtering execution is aborted, the incoming item is classified as spam (+) or ham (−) depending on the value of the definitive score.

Table 7: AUC results for C5.0 and SVM classifiers working together with regular expressions.

	Class-imbalance ratio				
	1 : 17	1 : 8	1 : 4	1 : 2	1 : 1
C5.0	0.562	0.649	**0.651**	0.648	0.573
SVM	0.534	0.590	0.602	0.604	**0.624**
C5.0 + SVM	0.579	0.658	**0.713**	0.684	0.646
C5.0 + SVM + REGEX	0.673	0.768	**0.798**	0.759	0.736

Table 7 presents the results of this final scenario in addition to the results shown in Table 6 for purposes of comparison.

As we can observe from Table 7, the reinforcement of the filter by using regular expressions allows us to obtain the best results regardless of the balancing conditions. Moreover, the 1 : 4 ratio achieves the best AUC value showing a theoretical improvement of the filtering capability by 0.085 when compared to the second scenario (C5.0 + SVM). Figure 6 shows the best ROC curve achieved by the combination of both classifiers plus the use of regular expressions in a single filter.

As we can realize from Figure 6, the true combination of different antispam techniques achieved by our WSF2 platform significantly improves the performance of the final classifier.

From another complementary perspective, filter specificity provides an assessment of the ability to correctly classify negative instances (i.e., avoiding FN errors). Taking into account our target problem and the importance of FN errors in this domain, this evaluation metric is particularly suitable for checking the potential usability of any filter. Moreover, the measurement of specificity and its comparison to sensitivity (i.e., the ability to detect positive instances) for the best cut-off configuration are especially interesting for a precise assessment of the filter performance. Therefore, in order to complement the global study previously presented, Table 8 combines both sensitivity and specificity for the best cut-off threshold together with the AUC value for each individual test carried out.

The results shown in Table 8 indicate that a better balance is achieved between sensibility and specificity as much as individual techniques/classifiers are aggregated into a single filter. We also detect a similar behaviour when analysing the corresponding AUC values.

4.4. Current Limitations. Although obtained results (discussed in detail in the previous section) have demonstrated the suitability of our novel approach to adequately combine different techniques for identifying web spam contents, there

```
web_body HAS_GRATIS_ON_BODY eval("[gG][rR][aA][tT][iI][sS]")
describe HAS_GRATIS_ON_BODY Finds if web page contains references to "Gratis" on content.
score HAS_GRATIS_ON_BODY +

web_body HAS_GORGEOUS_ON_BODY eval("[gG][oO][rR][gG][eE][oO][uU][sS]")
describe HAS_GORGEOUS_ON_BODY Finds if web page contains references to "Gorgeous" on content.
score HAS_GORGEOUS_ON_BODY +

web_body HAS_FOXHOLE_ON_BODY eval("[fF][oO][xX][hH][oO][lL][eE]")
describe HAS_FOXHOLE_ON_BODY Finds if web page contains references to "Foxhole" on content.
score HAS_FOXHOLE_ON_BODY +

web_body HAS_TRANSEXUAL_ON_BODY eval("[tT][rR][aA][nN][sS][eE][xX][uU][aA][lL]")
describe HAS_TRANSEXUAL_ON_BODY Finds if web page contains references to "Transexual" on content.
score HAS_TRANSEXUAL_ON_BODY +

web_body HAS_GODDAM_ON_BODY eval("[gG][oO][dD][dD][aA][mM]")
describe HAS_GODDAM_ON_BODY Finds if web page contains references to "Goddam" on content.
score HAS_GODDAM_ON_BODY +

web_body HAS_SLUTTY_ON_BODY eval("[sS][lL][uU][tT]{1,2}[yY]")
describe HAS_SLUTTY_ON_BODY Finds if web page contains references to "Slutty" on content.
score HAS_SLUTTY_ON_BODY +

web_body HAS_UNSECUR_ON_BODY eval("[uU][nN][sS][eE][cC][uU][rR]")
score HAS_UNSECUR_ON_BODY +

web_body HAS_BUSINESSOPPORTUNITY_ON_BODY eval("[bB][uU][sS][iI][nN][eE][sS]{1,2}[
][oO][pP]{1,2}[oO][rR][tT][uU][nN][iI][tT][yY]")
describe HAS_BUSINESSOPPORTUNITY_ON_BODY Finds if web page contains references to "Business Opportunity" on content.
score HAS_BUSINESSOPPORTUNITY_ON_BODY 5

web_body HAS_GAY_ON_BODY eval("[gG][aA][yY]")
describe HAS_GAY_ON_BODY Finds if web page contains references to "Gay" on content.
score HAS_GAY_ON_BODY 5

web_body HAS_CHEAP_ON_BODY eval("[cC][hH][eE][aA][pP]")
describe HAS_CHEAP_ON_BODY Finds if web page contains references to "Cheap" on content.
score HAS_CHEAP_ON_BODY 5

web_body HAS_BLONDE_ON_BODY eval("[bB][lL][oO][nN][dD][eE]")
describe HAS_BLONDE_ON_BODY Finds if web page contains references to "Blonde" on content.
score HAS_BLONDE_ON_BODY 5

web_body HAS_BARGAIN_ON_BODY eval("[bB][aA][rR][gG][aA][iI][nN]")
describe HAS_BARGAIN_ON_BODY Finds if web page contains references to "Bargain" on content.
score HAS_BARGAIN_ON_BODY 5

web_body HAS_RESORT_ON_BODY eval("[rR][eE][sS][oO][rR][tT]")
describe HAS_RESORT_ON_BODY Finds if web page contains references to "Resort" on content.
score HAS_RESORT_ON_BODY 5

web_body HAS_VENDOR_ON_BODY eval("[vV][eE][nN][dD][oO][rR]")
describe HAS_VENDOR_ON_BODY Finds if web page contains references to "Vendor" on content.
score HAS_VENDOR_ON_BODY 5

web_features SVM check_svm()
describe SVM Classifies a web page as spam using Support Vector Machine classifier
score SVM 5

```

Algorithm 5: Continued.

```
(60) web_features TREE_00 check_tree(0.0, 0.25)
(61) describe TREE_00 Classifies a web page as spam if C5.0 probability is between 0.0 and 0.25
(62) score TREE_00 −1
(63)
(64) web_features TREE_25 check_tree(0.25, 0.50)
(65) describe TREE_25 Classifies a web page as spam if C5.0 probability between 0.25 and 0.50
(66) score TREE_25 3
(67)
(68) web_features TREE_50 check_tree(0.50, 0.75)
(69) describe TREE_50 Classifies a web page as spam if C5.0 probability between 0.50 and 0.75
(70) score TREE_50 4
(71)
(72) web_features TREE_75 check_tree(0.75, 1.00)
(73) describe TREE_75 Classifies a web page as spam if C5.0 probability between 0.75 and 1
(74) score TREE_75 5
(75)
(76) required_score 5
```

Algorithm 5: WSF2 filter definition combining C5.0 and SVM algorithms together with regular expressions.

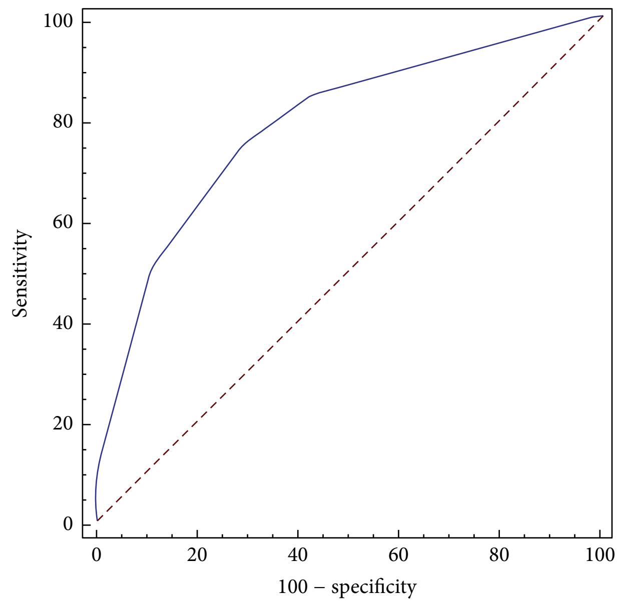


Figure 6: ROC curve for C5.0, SVM, and regular expressions combined in a single classifier (AUC = 0.798).

Table 8: Sensitivity, specificity, and AUC values obtained for each classifier executed in our WSF2 platform.

	Ratio	AUC	Sensitivity	Specificity
C5.0	1 : 17	0.562	14.2	98.4
	1 : 8	0.649	50.4	79.5
	1 : 4	**0.651**	66.4	65.9
	1 : 2	0.648	77.0	52.7
	1 : 1	0.573	82.3	32.4
SVM	1 : 17	0.534	7.1	99.7
	1 : 8	0.590	26.5	91.6
	1 : 4	0.602	29.2	91.2
	1 : 2	0.604	30.1	90.8
	1 : 1	**0.624**	34.5	90.3
C5.0 + SVM	1 : 17	0.579	17.7	98.1
	1 : 8	0.658	52.2	79.4
	1 : 4	**0.713**	*63.7*	*72.9*
	1 : 2	0.684	77.0	52.7
	1 : 1	0.646	34.5	90.2
C5.0 + SVM + REGEX	1 : 17	0.673	44.2	87.7
	1 : 8	0.768	78.8	69.8
	1 : 4	**0.798**	*75.3*	*71.8*
	1 : 2	0.759	54.9	85.7
	1 : 1	0.736	62.8	82.3

exist some practical limitations to deploy a fully functional filtering solution based on our WSF2 framework. In this line, during the development of this work, we have identified three different weaknesses of WSF2: (i) the small number of ready-to-use classification and feature extraction techniques, (ii) the lack of initial setup options to select those features that will be later used by each classifier, and (iii) the need of expert knowledge to build a fully functional WSF2 filter.

At this point, the most important WSF2 limitation is the lack of a large number of classification models. In fact, we have found it necessary to include several complementary ML classifiers (e.g., Bagging approaches [46], Boosting techniques [46, 47], Random Forest [48], or different Naïve Bayes algorithms [49]) as well as other domain specific techniques, such as URI Blacklists.

Additionally, all the available ML classifiers could be tested using different sets of features. In this context, as some learners perform better when using a certain type of features, the framework would allow users to indicate those features to be used by the classifiers (or automatically select the best characteristics for each ML technique). In addiction, the number of available features to train classification models could be also enlarged.

Currently, the triggering condition and the score of all the rules that compose a given filter are manually configured. This complex task is routinely accomplished by an expert having considerable experience in the domain. Although there are currently some enterprises that could commercialize services to provide accurate filter configurations, in the near future, our WSF2 framework should incorporate a fully functional knowledge discovering module able to (i) automatically define the triggering condition of all the rules, (ii) discover and delete outdated rules, and (iii) mechanically adjust both the specific score of each rule and the global filter threshold, with the goal of maximizing performance and safety.

5. Conclusions and Future Work

This work presented WSF2, a novel platform for giving specific support to filter spam web contents. WSF2 provides a reliable framework in which different algorithms and techniques can be easily combined to develop strong and adaptable web content filters. In order to ensure its extensibility, WSF2 supports the usage of plug-ins to develop and integrate new filtering approaches, introducing the concept of rule to support their execution. Using our WSF2 filter model, any filter can be easily implemented as a combination of different weighted rules (each one invoking separate classification algorithms) coupled with a global filtering threshold. The current architecture design of WSF2 emerged from popular e-mail filtering infrastructures including SpamAssassin [10] and Wirebrush4SPAM [11].

Through the combination of different but complementary techniques, we will be able to develop novel classifiers that outperform the capabilities of the original algorithms. Thus, in order to demonstrate the real value of our WSF2 platform, we have successfully integrated SVM, C5.0, and regular expressions to build up an ensemble filter able to outperform the individual performance of those algorithms.

Regarding the software engineering experience gained through the development of this project, we can state that the flexible architecture used to create the WSF2 platform facilitates the integration of novel and/or existing techniques while maximizing filtering speed. Although some key aspects concerning the WSF2 architecture design and source code were borrowed from SpamAssassin and Wirebrush4SPAM projects, respectively, the distinctive nature of the web spam filtering domain involved the redesign of different data interchange schemes to support the true interaction with search engines and to provide a benchmark framework for academic environments. In addition, the implementation of specific algorithms and parsers to support the target domain (web spam filtering) was also required to achieve the full set of features currently offered by the WSF2 platform.

In order to improve the functionality and performance of our WSF2 framework, some new spam filtering techniques should receive further support. In this line, we highlight that most common supervised ML classification techniques can be successfully imported in our WSF2 framework. To this end, we will specifically evaluate some ML libraries such as VFML [50] and other implementations of ML approaches like AdaBoost [47]. Moreover, we also believe that our WSF2 framework can take advantage from URI Blacklists (URIBL), commonly used in the e-mail filtering domain. In addition to the obvious technical development, we believe that the use of different filter optimization heuristics (e.g., tuning up rule scores, finding and removing irrelevant features, or detecting counterproductive rules) would be very appropriate to complement the current state of the art [51–54]. Finally, the lack of effective tools for web spam dataset management and maintenance also suggests an interesting option for future research activities.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was partially funded by the Projects [15VI013] Contract-Programme from the University of Vigo and [TIN2013-47153-C3-3-R] "Platform of Integration of Intelligent Techniques for Analysis of Biomedical Information" from the Spanish Ministry of Economy and Competitiveness.

References

[1] R. Jaslow, "FDA shuts down over 1,600 online pharmacies," CBSNews, 2013, http://www.cbsnews.com/news/fda-shuts-down-over-1600-online-pharmacies/.

[2] D. McCoy, A. Pitsillidis, G. Jordan et al., "Pharmaleaks: understanding the business of online pharmaceutical affiliate programs," in *Proceedings of the 21st USENIX Conference on Security Symposium (Security '12)*, p. 1, USENIX Association, 2012.

[3] N. Christin, "Traveling the silk road: a measurement analysis of a large anonymous online marketplace," in *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*, pp. 213–223, May 2013.

[4] Y.-M. Wang, M. Ma, Y. Niu, and H. Chen, "Spam double-funnel: connecting web spammers with advertisers," in *Proceedings of the 16th International World Wide Web Conference (WWW '07)*, pp. 291–300, ACM, May 2007.

[5] Google, "Fighting Spam—Inside Search—Google," 2014, http://www.google.com/insidesearch/howsearchworks/fighting-spam.html.

[6] K. P. Karunakaran and S. Kolkur, "Review of web spam detection techniques," *International Journal of Latest Trends in Engineering and Technology*, vol. 2, no. 4, pp. 278–282, 2013.

[7] N. Spirin and J. Han, "Survey on web spam detection: principles and algorithms," *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 2, pp. 50–64, 2011.

[8] M. Erdélyi and A. A. Benczúr, "Temporal analysis for web spam detection: an overview," in *Proceedings of the 1st International Temporal Web Analytics Workshop*, pp. 17–24, March 2011.

[9] L. Han and A. Levenberg, "Scalable online incremental learning for web spam detection," in *Recent Advances in Computer Science and Information Engineering: Proceedings of the 2nd World Congress on Computer Science and Information Engineering*,

vol. 124 of *Lecture Notes in Electrical Engineering*, pp. 235–241, Springer, Berlin, Germany, 2012.

[10] SpamAssassin, "The Apache SpamAssassin Project," 2011, http://spamassassin.apache.org/.

[11] N. Pérez-Díaz, D. Ruano-Ordas, F. Fdez-Riverola, and J. R. Méndez, "Wirebrush4SPAM: a novel framework for improving efficiency on spam filtering services," *Software: Practice and Experience*, vol. 43, no. 11, pp. 1299–1318, 2013.

[12] C. Castillo and B. D. Davison, "Adversarial web search," *Foundations and Trends in Information Retrieval*, vol. 4, no. 5, pp. 377–486, 2010.

[13] Z. Gyöngyi and H. Garcia-Molina, "Web spam taxonomy," in *Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web (AIRWeb '05)*, pp. 39–47, Chiba, Japan, May 2005.

[14] S. Ghiam and A. N. Pour, "A survey on web spam detection methods: taxonomy," *International Journal of Network Security & Its Applications*, vol. 4, no. 5, pp. 119–134, 2012.

[15] D. Fetterly, M. Manasse, and M. Najork, "Spam, damn spam, and statistics: using statistical analysis to locate spam web pages," in *Proceedings of the 7th International Workshop on the Web and Databases (WebDB '04)*, pp. 1–6, ACM, Paris, France, June 2004.

[16] D. Fetterly, M. Manasse, and M. Najork, "Detecting phrase-level duplication on the world wide web," in *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '05)*, pp. 170–177, ACM, Salvador, Brazil, August 2005.

[17] 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 '06)*, pp. 83–92, ACM, May 2006.

[18] M. Erdélyi, A. Garzó, and A. A. Benczúr, "Web spam classification: a few features worth more," in *Proceedings of the Joint WICOW/AIRWeb Workshop on Web Quality (WebQuality '11)*, pp. 27–34, Hyderabad, India, March 2011.

[19] G.-G. Geng, X.-B. Jin, X.-C. Zhang, and D. X. Zhang, "Evaluating web content quality via multi-scale features," in *Proceedings of the ECML/PKDD 2010 Discovery Challenge*, Barcelona, Spain, September 2010.

[20] A. Sokolov, T. Urvoy, L. Denoyer, and O. Ricard, "Madspam consortium at the ECML/PKDD discovery challenge 2010," in *Proceedings of the ECML/PKDD Discovery Challenge*, Barcelona, Spain, September 2010.

[21] V. Nikulin, "Web-mining with wilcoxon-based feature selection, ensembling and multiple binary classifiers," in *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD '10)*, September 2010.

[22] B. Davison, "Recognizing nepotistic links on the web," in *Proceedings of the AAAI Workshop on Artificial Intelligence for Web Search*, pp. 23–28, Austin, Tex, USA, July 2000.

[23] G.-G. Geng, C.-H. Wang, Q.-D. Li, L. Xu, and X.-B. Jin, "Boosting the performance of web spam detection with ensemble under-sampling classification," in *Proceedings of the 4th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD '07)*, pp. 583–587, Haikou, China, August 2007.

[24] L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates, "Web spam detection: link-based and content-based techniques," in *Proceedings of the European Integrated Project Dynamically Evolving, Large Scale Information Systems (DELIS '08)*, pp. 99–113, Barcelona, Spain, February 2008.

[25] R. M. Silva, T. A. Alimeida, and A. Yamakami, "Machine learning methods for spamdexing detection," *International Journal of Information Security Science*, vol. 2, no. 3, pp. 1–22, 2013.

[26] K. M. Svore, Q. Wu, C. J. C. Burges, and A. Raman, "Improving web spam classification using rank time features," in *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb '07)*, pp. 9–16, Banff, Canada, 2007.

[27] J. Abernethy, O. Chapelle, and C. Castillo, "Web spam identification through content and hyperlinks," in *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb '08)*, pp. 41–44, ACM, Beijing, China, April 2008.

[28] M. Najork, "Web spam detection," in *Encyclopedia of Database Systems*, pp. 3520–3523, Springer, 2009.

[29] B. Wu and B. D. Davison, "Detecting semantic cloaking on the Web," in *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*, pp. 819–828, ACM, May 2006.

[30] K. Chellapilla and D. 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 '06)*, pp. 17–24, Seattle, Wash, USA, August 2006.

[31] G.-G. Geng, X.-T. Yang, W. Wang, and C.-J. Meng, "A Taxonomy of hyperlink hiding techniques," in *Web Technologies and Applications*, vol. 8709 of *Lecture Notes in Computer Science*, pp. 165–176, Springer, 2014.

[32] B. Wu and B. D. Davison, "Cloaking and redirection: a preliminary Study," in *Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web (AIRWeb '05)*, pp. 7–16, Chiba, Japan, May 2005.

[33] K. Chellapilla and A. Maykov, "A taxonomy of javascript redirection spam," in *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb '07)*, vol. AIRWeb, pp. 81–88, Alberta, Canada, May 2007.

[34] D. Ruano-Ordás, J. Fdez-Glez, F. Fdez-Riverola, and J. R. Méndez, "Effective scheduling strategies for boosting performance on rule-based spam filtering frameworks," *Journal of Systems and Software*, vol. 86, no. 12, pp. 3151–3161, 2013.

[35] B. Liu and F. Menczer, "Web crawling," in *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*, pp. 311–362, Springer, 2nd edition, 2011.

[36] C. Olston and M. Najork, "Web crawling," *Foundations and Trends in Information Retrieval*, vol. 4, no. 3, pp. 175–246, 2010.

[37] V. Shkapenyuk and T. Suel, "Design and implementation of a high-performance distributed web crawler," in *Proceedings of the 18th International Conference on Data Engineering*, pp. 357–368, March 2002.

[38] L. Araujo and J. Martínez-Romo, "Web spam detection: new classification features based on qualified link analysis and language models," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 3, pp. 581–590, 2010.

[39] M. Mahmoudi, A. Yari, and S. Khadivi, "Web spam detection based on discriminative content and link features," in *Proceedings of the 5th International Symposium on Telecommunications (IST '10)*, pp. 542–546, Tehran, Iran, December 2010.

[40] C. Castillo, D. Donato, L. Becchetti et al., "A reference collection for web spam," *ACM SIGIR Forum*, vol. 40, no. 2, pp. 11–24, 2006.

[41] H. Wahsheh, I. Abu Doush, M. Al-Kabi, I. Alsmadi, and E. Al-Shawakfa, "Using machine learning algorithms to detect content-based arabic web spam," *Journal of Information Assurance and Security*, vol. 7, no. 1, pp. 14–24, 2012.

[42] S. Webb, J. Caverlee, and C. Pu, “Introducing the Webb spam corpus: using email spam to identify web spam automatically,” in *Proceedings of the 3rd Conference on Email and AntiSpam (CEAS ’06)*, 28, p. 27, July 2006.

[43] D. Wang, D. Irani, and C. Pu, “Evolutionary study of web spam: Webb Spam Corpus 2011 versus Webb Spam Corpus 2006,” in *Proceedings of the 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom ’12)*, pp. 40–49, Pittsburgh, Pa, USA, October 2012.

[44] C. Castillo, K. Chellapilla, and L. Denoyer, “Web spam challenge 2008,” in *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb ’08)*, Beijing, China, April 2008.

[45] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri, “Know your neighbors: web spam detection using the web topology,” in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’07)*, pp. 423–430, ACM, Amsterdam, The Netherlands, July 2007.

[46] T. G. Dietterich, “An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization,” *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.

[47] Freund Lab Wiki, Adaboost.c, 2014, http://seed.ucsd.edu/mediawiki/index.php/AdaBoost.c.

[48] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[49] V. Metsis, I. Androutsopoulos, and G. Paliouras, “Spam filtering with naive Bayes-which naive bayes?” in *Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS ’06)*, Mountain View, Calif, USA, July 2006.

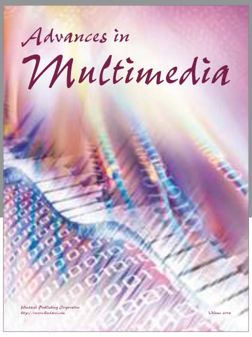
[50] VFML, VFML 2013, http://www.cs.washington.edu/dm/vfml/.

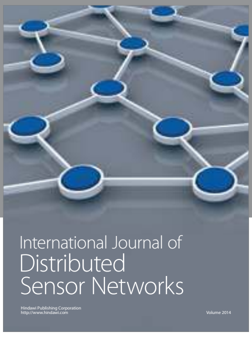
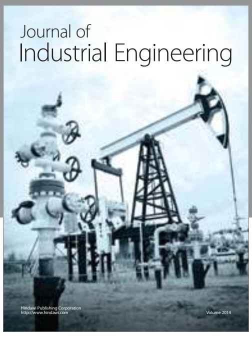
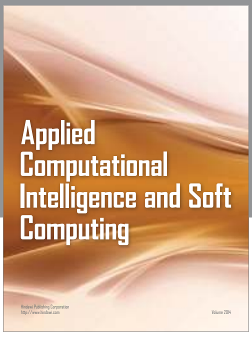
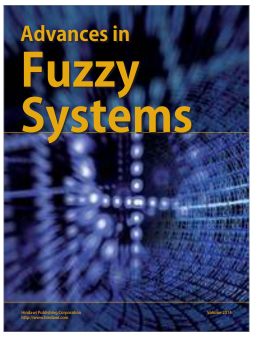
[51] I. Yevseyeva, V. Basto-Fernandes, and J. R. Méndez, “Survey on anti-spam single and multi-objective optimization,” in *Proceedings of the 3th Conference on ENTERprise Information Systems (CENTERIS ’11)*, pp. 120–129, Vilamoura, Portugal, October 2011.

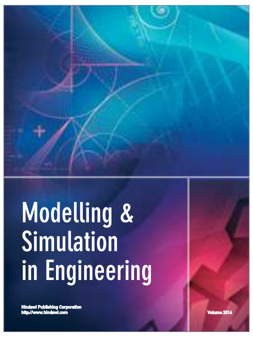
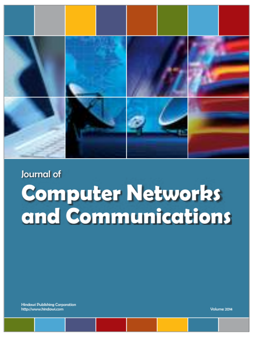
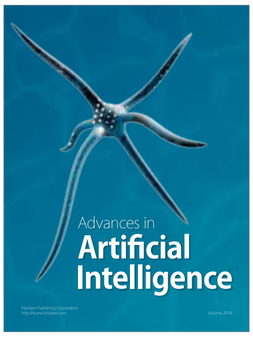
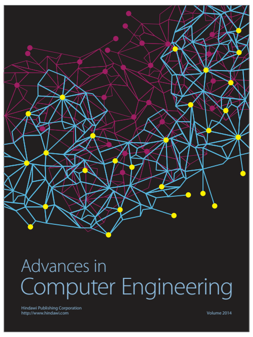
[52] V. Basto-Fernandes, I. Yevseyeva, and J. R. Méndez, “Optimization of anti-spam systems with multiobjective evolutionary algorithms,” *Information Resources Management Journal*, vol. 26, no. 1, pp. 54–67, 2013.

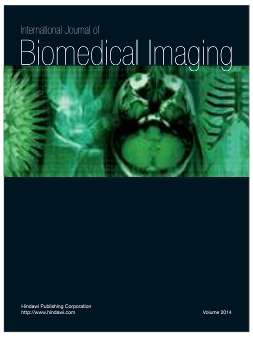
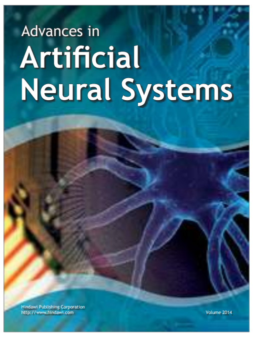
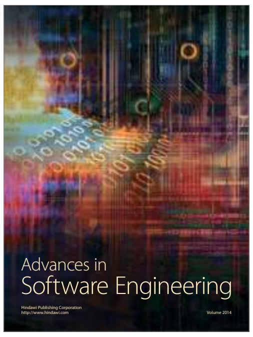
[53] I. Yevseyeva, V. Basto-Fernandes, D. Ruano-Ordás, and J. R. Méndez, “Optimising anti-spam filters with evolutionary algorithms,” *Expert Systems with Applications*, vol. 40, no. 10, pp. 4010–4021, 2013.

[54] J. R. Méndez, M. Reboiro-Jato, F. Díaz, E. Díaz, and F. Fdez-Riverola, “Grindstone4Spam: an optimization toolkit for boosting e-mail classification,” *Journal of Systems and Software*, vol. 85, no. 12, pp. 2909–2920, 2012.

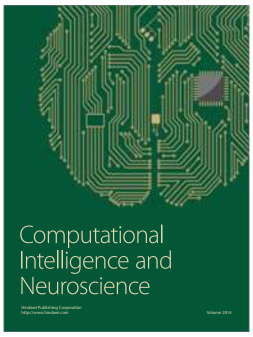

Advances in Multimedia

The Scientific World Journal

International Journal of Distributed Sensor Networks

Journal of Industrial Engineering

Applied Computational Intelligence and Soft Computing

Advances in Fuzzy Systems


Modelling & Simulation in Engineering

Journal of Computer Networks and Communications

Advances in Artificial Intelligence

Advances in Computer Engineering

International Journal of Computer Games Technology

International Journal of Biomedical Imaging

Advances in Artificial Neural Systems

Advances in Software Engineering

Journal of Robotics

Advances in Human-Computer Interaction

Computational Intelligence and Neuroscience

International Journal of Reconfigurable Computing

Journal of Electrical and Computer Engineering