DECISION SUPPORT FOR ESTIMATION OF THE UTILITY OF SOFTWARE AND E-MAIL

Anton Borg

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2012:07

School of Computing

Decision Support for Estimation of the Utility of Software and E-mail

Anton Borg

Blekinge Institute of Technology licentiate dissertation series
No 2012:07

Decision Support for Estimation of the Utility of Software and E-mail

Anton Borg

Licentiate Dissertation in
Computer Science

School of Computing
Blekinge Institute of Technology
SWEDEN

2012 Anton Borg
School of Computing
Publisher: Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
Printed by Printfabriken, Karlskrona, Sweden 2012
ISBN: 978-91-7295-236-2
ISSN 1650-2140
urn:nbn:se:bth-00533

Imagination will often carry us to worlds that never were. But without it we go nowhere.

Carl Sagan, *Cosmos*, 1980.

Abstract

Background: Computer users often need to distinguish between good and bad instances of software and e-mail messages without the aid of experts. This decision process is further complicated as the perception of spam and spyware varies between individuals. As a consequence, users can benefit from using a decision support system to make informed decisions concerning whether an instance is good or bad.

Objective: This thesis investigates approaches for estimating the utility of e-mail and software. These approaches can be used in a personalized decision support system. The research investigates the performance and accuracy of the approaches.

Method: The scope of the research is limited to the legal greyzone of software and e-mail messages. Experimental data have been collected from academia and industry. The research methods used in this thesis are simulation and experimentation. The processing of user input, along with malicious user input, in a reputation system for software were investigated using simulations. The preprocessing optimization of end user license agreement classification was investigated using experimentation. The impact of social interaction data in regards to personalized e-mail classification was also investigated using experimentation.

Results: Three approaches were investigated that could be adapted for a decision support system. The results of the investigated reputation system suggested that the system is capable, on average, of producing a rating ± 1 from an objects correct rating. The results of the preprocessing optimization of end user license agreement classification suggested negligible impact. The results of using social interaction information in e-mail classification suggested that accurate spam detectors can be generated from the low-dimensional social data model alone, however, spam detectors generated from combinations of the traditional and social models were more accurate.

Conclusions: The results of the presented approaches suggest that it is possible to provide decision support for detecting software that might be of low utility to users. The labeling of instances of software and e-mail messages that are in a legal grey-zone can assist users in avoiding an instance of low utility, e.g. spam and spyware. A limitation in the approaches is that isolated implementations will yield unsatisfactory results in a real world setting. A combination of the approaches, e.g. to determine the utility of software, could yield improved results.

Acknowledgments

I would like to thank my supervisors, *Bengt Carlsson*, *Niklas Lavesson*, and *Martin Boldt*, for their guidance and support.

I would also like to thank the rest of my colleagues at BTH. And I would like to thank my family and friends for their support.

This work was partly founded by .SE, the Internet Infrastucture Foundation.

Preface

This compilation thesis consists of four articles that have been peer reviewed and published at conferences or submitted for publication. The articles have been co-authored with senior colleagues. The term 'we' denotes the authors of the publication in question. The following publications are included.

1. Martin Boldt, Anton Borg, Bengt Carlsson, "On the Simulation of a Software Reputation System," pp.333-340, 2010 *International Conference on Availability, Reliability and Security*, IEEE.

2. Anton Borg, Martin Boldt, Bengt Carlsson, "Simulating Malicious Users in a Software Reputation System", *Secure and Trust Computing, Data Management and Applications*, 2011, Communications in Computer and Information Science, Volume 186, Part 1, 147-156, Springer.

3. Anton Borg, Martin Boldt, Niklas Lavesson, "Informed software installation through License Agreement Categorization," *Information Security South Africa*, 2011, pp.1-8., IEEE.

4. Anton Borg, Niklas Lavesson, "E-mail Classification using Social Network Information," *Accepted for publication*, 2012 *International Conference on Availability, Reliability and Security*, IEEE.

Publication 1 and 2 are related in that publication 2 is a continuation of 1. While both publications concern producing reliable score for a reputation system, the first publication concerns weighting user knowledge and the second concerns detecting malicious users. For publication 1, the thesis author was involved in the investigation, but was not the main driver. The author's involvement comprised setting up the experimental design, data analysis and writing the paper in cooperation with the other authors. For publication 2, the thesis author was the main driver in the investigation. The author's involvement comprised of extending the simulation and setting up the experiment design. The data analysis and writing of the article was done in cooperation with the co-authors.

Publication 3 extends previous research, adding automatic extraction and processing of EULAs. For this publication, the thesis author was the main driver in the investigation. The involvement comprised in setting up experiment design, writing the paper and analyzing the data. For publication 4, the thesis author was the main driver in setting up experiment design, writing the paper, analyzing data and designing the algorithm.

Contents

One

Introduction

Computer users often install software and use services such as e-mail without the assistance of professionals [1]. It can be difficult to differentiate between good and bad instances of e.g. software [2]. The impact of making bad decisions can be bad. People are susceptible to various unknown biases when weighing risk versus benefit [3]. As a consequence, users may benefit from using a decision support system if it enables them to make informed decisions concerning whether an instance is good or bad. This decision can be based on an estimation of the utility of the instance.

Utility is a measure of user satisfaction, e.g. when consuming a service or product [4]. Utility comes from utilitarianism, which concerns the maximization of happiness or satisfaction of the maximum amount of people [5]. The cost versus utility balance is used to determine the risk involved in financial planning. It can also be applied to risk analysis, or for estimating the utility of services and applications. Since utility is a measure of user satisfaction, its interpretation differs between users. The utility of spyware can be low as the intent of the developer is in contrast to what the user condones. The utility of spyware can also be perceived as high as the user consider the function of the program to outweigh the spyware aspect. In most cases though, the utility of spyware and spam are low when compared to legitimate applications and messages. Spam can be of a low utility as spam may be deceptive or contain spyware [6]. Labeling messages as spam is also subjective, since what is considered spam can differ between users and even for the same user over a span of time. A support system that can estimate the utility beforehand can be used to avoid spyware or spam, and a personalized system can provide personalized decisions.

This thesis investigates how decision support systems can be used for determining the utility of software and e-mail messages, consequently allowing users to avoid instances where the estimated utility is low. Two approaches are investigated. The first approach investigates using a *software reputation system* (SRS) as a complement to traditional anti-spyware tools. SRS users provide ratings of programs, thus providing a collaborative platform for judging applications. The second approach concerns text mining, which is applied to spyware and spam detection. The spyware detection is based on analyzing *End User License Agreements* (EULAs) as a representation for software behavior. The spam detection aims to create personalized spam detection models and investigates linking online social network data to e-mails for improved performance.

Chapter 2 presents the background and terminology. In Chapter 3, Section 3.2 lists the research questions. Section 3.5 concerns the contributions of the publications, as well as the authors involvement. The results are discussed and concluded in Section 3.6. Section 3.8 presents points to future work. Finally, the publications are presented in Chapter 4-7.

Two

Background

Malicious Software (malware) can be defined as "a set of instructions that run on your computer and make your system do something that an attacker wants it to do" [7]. The first virus surfaced in 1982, a year before the first computer virus definition was presented [7]. To counteract the new threat of viruses, anti-virus software was released. The first anti-virus programs were signature-based, but as malware became more and more sophisticated, so did anti-malware software. Since the first release, malware became increasingly complicated, going from individual developers to nation-level supported developers. An example of nation-level supported malware is the Stuxnet virus. The Stuxnet virus was described as a cyberwarfare weapon[1], due to it specifically targeting certain nuclear power plant control systems [8, 9]. Due to the evolution of malware, the removal of malware was a question of using the resources and techniques available at specific instances of time [10].

Two phases of anti-malware evolution were identified [11]. The first stage comprised *signature-based detection*, in which applications where compared against a digital signature in a database. The signature-based approach exhibited two weaknesses. It required a copy of the malware

[1]Kaspersky Lab's stated that it is likely that the creators of Stuxnet have nation-level support. `http://www.kaspersky.com/about/news/virus/2010/Kaspersky_Lab_provides_its_insights_on_Stuxnet_worm`

to extract a unique signature. In addition, it required that the updated signature database was distributed to all customers of the anti-virus software [10]. Consequently, anti-malware was one step behind malware. As the number of malware increased, the signature databases grew in size.

The second stage of anti-malware detection was needed to handle the amount of signatures and the sophistication of more recent viruses [11]. To address these new requirements, variations of signature-based detection, e.g. skipping NOP instructions, and *heuristics-based detection* was introduced. The latter was used to search for features found in malware, and if a certain number of features were found, the investigated program was considered to be a virus [7, 11]. Anti-virus manufacturers began investigating alternative techniques for solving the problem. Another technique used within this stage was *dynamic analysis*, which kept a suspicious program in captivity on a virtual machine, i.e. a sandbox, while monitoring its execution as a way to discover any irregular behavior [10, 12]. The development of malware and anti-malware can be described as an arms race in that the developers of each tried to be one step ahead of the other.

Spyware emerged during the late nineties [13]. One reason was that the Internet began to reach the general population, which resulted in a new market where personal information was used for distributing targeted online advertisements [14]. The main consequence was that a new type of grey-zone software appeared, which further complicated the separation between legitimate and illegitimate software. Even though dynamic analysis could be used for computer viruses, e.g. by detecting the self-replication routines, it was more difficult to distinguish spyware or adware applications from their legitimate counterparts [15]. Anti-spyware developers were sued for labeling a piece of grey-zone software as spyware [16].

The development of spam detection approaches evolved in a similar manner, beginning with a rule-based approach, i.e. filtering on the occurrence of a phrase or a word, into more complicated means of detection, even involving payment-based deterrents [17,18]. The first recorded instance of spam, in the form of an unsolicited commercial e-mail (UCE), occurred in 1978, when more than 500 e-mails were sent to ARPANET's users [6]. It was labeled UCE as the purpose of the messages was to announce a new mainframe computer. The administrators of ARPANET stated that the messages violated the user agreement of the network [6].

With an increased number of spam, customer complaints to ISPs increased, prompting the ISPs to find a solution for spam. Since no legislation existed, technological means, e.g. filtering, was applied [6]. Anti-spam techniques involved filtering e-mails, via either automatic or semi-automatic filtering [19]. Some of the filtering methods used white- or blacklisting sender IP addresses, URLs or keywords in the e-mail. Automatic filtering methods used e.g. machine learning to learn to differentiate between solicited and unsolicited e-mail patterns, as well as bouncing suspicion messages (a challenge response system) [17].

The U.S.A. passed the CAN-SPAM act (Controlling the Assault of Non-Solicited Pornography And Marketing) in 2003, regulating what can be sent as spam and what elements spam must contain in order to be compliant [6]. The CAN-SPAM act stated that users should have the option to opt-out, i.e. to stop receiving unsolicited messages[2]. In 2004 a study concluded that only 14.3% of spam messages were compliant with the CAN-SPAM act [20]. The EU and the Australian governments adopted an opt-in approach [21]. An opt-in approach, required the supposed recipient to give their consent before the messages can be sent. It has been argued that in order to effectively be able to regulate spam, the laws governing spam must be international [22].

[2]The Law have been referred to as 'you can spam' by critics, due to its permissiveness. `http://www.wired.com/techbiz/media/news/2004/01/62020`

2.1 Terminology

Utility can be defined as *value in use* or as a measurement of usage satisfaction [4]. Utility can be applied to products or services and the combined utility can be used to determine the generalized user satisfaction of an item or a service. The expected utility hypothesis, related to utility, can be defined as follows: faced with choices involving risk, individuals will choose as to maximize the expected utility [23]. Given a situation where the estimated utility is low, individuals can choose whether to consume a service or not [24]. As a consequence, utility is useful in decision support systems in the personalized context.

Decision support system (DSS) helps users to make decisions in uncertain situations. DSS can be organized into five groups, extending a categorization from 1980[3] [25, 26, 27]. DSS belonging to more than one group are denoted Hybrid DSS.

Communications-Driven DSS can be exemplified as a system that helps users reach a decision together, e.g. a reputation system.

Data-Driven DSS can be described as a system that allows easy access to data available in, e.g. files and databases, to help facilitate decision making. This can be exemplified by real-time monitoring systems or budget analysis systems.

Document-Driven DSS can be a system that helps users locate correct data, documents, files, or, e.g., web-sites. An example of this is a search-engine.

Knowledge-Driven DSS can be described as a system "that search for hidden patterns in a database", and can be seen as closely related to data mining [27]. This category requires a good understanding of a specific task.

[3]A more extensive look at the earlier categorization and how it relates to the reworked framework can be found online. Included are also additional examples. `http://dssresources.com/faq/index.php?action=artikel&id=167`

Model-Driven DSS uses "data and parameters provided by decision-makers to aid them in analyzing a situation" [27]. Examples of systems could e.g. include scheduling systems or risk analysis systems.

Reputation Systems are a form of Communication-Driven DSS. Reputation Systems allow users to rate and provide feedback concerning objects in a domain, often to filter objects or to establish trust between users [28, 29]. Depending on the purpose of the system, the objects can be either users of the system or e.g. nodes in a network.

Machine learning concerns the study of programs that learn from experience to improve the performance at solving tasks [30]. Machine learning comprises a large number of directions, methods, and concepts, which can be organized into learning paradigms. Usually, three paradigms are distinguished; supervised learning, unsupervised learning, and reinforcement learning. The suitability of a certain learning method or paradigm depends largely on the type of available data for the problem at hand.

Text classification, or text categorization, concerns the machine learning problem of associating a text document to one or more classes or categories [31]. Text categorization can be used for various purposes e.g. to detect spam [32].

Spam is synonymous to unsolicited e-mail messages, but is also referred to as *unsolicited bulk e-mail messages* or as *unsolicited commercial e-mail* [17]. The term SPAM, borrowed from a Monthy-Python sketch, can be described as unsolicited mass sending of messages to users, often with a commercial agenda [6]. The key term, unsolicited, indicates that users receive the messages without prior warning or without requesting the messages. Spam may be deceptive or have spyware attached [6].

Spyware originally referred to software tracking, reporting user behavior, and which is installed without the users informed consent [15, 33]. However, the term has taken on a broader meaning, to also include adware [15, 16]. Adware differs from spyware in that it primarily displays advertisements [15]. These programs are considered unsolicited and as a consequence probably unwanted [14,16,34]. What makes greyzone spyware different from other malware, is the requirement of user consent. User consent is often given through EULAs.

EULA can be defined as "...a form of legal contract between two or more parties in which a licensor allows certain use of rights to a licensee, normally for a fee [35]." EULAs regulate the terms under which users can make use of a product, as well as the terms of how the product may operate [36]. The idea is to protect vendors from possible repercussions from use of the product or to protect the intellectual property of the company [37]. The EULA describes the use terms, which can be divided into user rights and user restrictions [35, 36]. The user rights describe the terms under which the user is allowed to use the software and user restrictions describe what the user is not allowed to do. For example, a company can include a term in their license agreement that states that the user may not reverse engineer the product [35]. EULAs tend to be verbose, complicated and written as legal documents, and as a result, most users avoid reading them or fail to comprehend the underlying implications of the texts [38].

2.2 Related Work

Decision support systems (DSS) help users make decisions in uncertain situations. The research conducted has been summarized and reviewed in several surveys since the introduction of the term.These surveys cover the years of 1971-1988, 1988-1994, 1995-2001, as well as a trend analysis through the years 1971-1995 [25,39,40], [41].

Various types of DSS have been developed for or deployed in different real world settings. Research has been conducted on using DSSes to help construction tendering processes. Construction tendering processes are an early stage of construction projects dealing with biddings with regard to procurement of services or goods [42]. Even though the use of DSS has been viewed as beneficial to tendering, the current approaches mainly concern structured data and as a consequence do not provide decision support in regards to free-text documents, e.g. contracts [42]. DSSes require the problem or data to be either structured or fairly structured [43]. The presence of unstructured data, e.g. free-text, requires the decision maker to aid in the process.

Examples of other explored applications of DSS applies to the fields of tactical air combat, assisting in stock trading, water resource management, and within the health-care sector, operational assistance, triaging patients and hospital management [44, 45, 46, 47].

Communications-driven DSS have been implemented for e.g. spam detection and detecting malicious activities in peer-to-peer (p2p) networks [48, 49, 50]. In the case of spam detection, sender reputation and object reputation were investigated [48,50]. Sender reputation concerned establishing the identity of the sender, which allowed users to rate the identity over time. The problem of sender reputation based spam detection has been identifying the sender, as malicious users forged information or their online presence were short. As a consequence, sender reputation has been useful for honest senders and can be applied to whitelisting approaches [48]. The second approach, object reputation, allowed users to submit fingerprints or signatures of messages considered to be spam. New messages users received were compared against a catalog of message signatures [48]. The problem with object reputation has been the fingerprinting process, as the algorithm should be able to identify variations of messages and at the same time not match legitimate messages [48]. In p2p networks research similar to sender reputation, denoted peer reputation, and object reputation are identified [49, 51].

Knowledge-driven DSS, or rather machine learning based approaches, have been used for spam detection as well as spyware detection. The first ventures toward automatic spam detection were into automating the rule-based learning techniques [52]. The currently employed anti-spam techniques were summarized [32, 53, 54]. These studies provide coverage of learning-based spam detection and one of the main conclusions were that automated (machine learning-based) techniques are necessary in order to implement spam filtering. In regard to malware and spyware, the research have concerned the application of classifiers on static features of software and how binaries can be represented [55, 56]. Research has also investigated behavioral-based detection of malicious software, including heuristic-based models, data mining and classifier approaches [12]. The learning algorithms trained and classified based on extracted behavior signatures [12].

Three

Approach

3.1 Aim & Scope

This thesis aims to investigate suitable means for estimating the utility of software and e-mail. It is argued that unsolicited software and e-mail, in the form of spyware and spam, contain secondary objectives that can decrease the utility. By providing means to aid the user in estimating the utility of an instance of software or e-mails, unsolicited software or e-mail messages can be avoided. As a consequence, means for estimating the utility of software and e-mail can be incorporated into computer-based decision support systems. The scope is limited to grey-zone spyware and spam.

3.2 Research Questions

Spyware and spam can be said to reside within a legal grey-zone. As a consequence, traditional countermeasures are sometimes incapable of detecting borderline instances. Due to the legal status or differences in users opinions of spyware and spam, automated responses are sometimes impossible. Consequently, methods must be investigated that provide an automatic response, but which leave the final decision to the end user. Decision support systems have been used for similar problems before, helping users solve complicated problems. For similar problems concerning spyware and spam, communication- and knowledge-driven DSS have been used. Given this situation, the research questions explored in this thesis are:

RQ I. *With what accuracy can a reputation system converge user software ratings towards a true value for software installation support?*

This question is asked in the context of a reputation system that records numeric ratings of software objects, each with a true rating unknown to the system, and users with different knowledge levels. The ability to determine the utility of software prior to installation would allow users to identify solicited software such as spyware. Reputation systems have been used for providing quantitative feedback concerning products or services, as a mean of support for decision making related to the previously mentioned product or service. A software reputation system would allow users to help other users estimate the utility of software. The different levels of knowledge regarding software should be taken into account. As a consequence, whether reputation systems are applicable in the domain of software and a reputation system is capable of differentiating between users knowledge levels is investigated in the papers constituting Chapter 4 and Chapter 5.

RQ II. *What impact does preprocessing optimization have on classification performance for EULA classification?*

Traditional anti-virus and anti-spyware techniques are based on a variation of signature comparison and various heuristic methods. Recent research has investigated the use of EULA analysis [57]. License agreements are verbose, making it difficult for users to comprehend [38]. Using text categorization to find patterns and determine whether the EULA can be considered to represent benign software would help the user estimate the utility of software. The impact of feature selection or performance optimization of learning algorithms is investigated in Chapter 6.

RQ III. *What impact does social interaction information have on e-mail classification performance?*

Spam filtering can make use of text categorization techniques. The successful estimation of the utility of an e-mail message prior to reading allows a user to focus on high-utility messages. Similar to mining licenses to detect grey-zone spyware, Chapter 7 investigates using social network data and text classification to detect unsolicited e-mail and aiding users with regards to e-mail messages. Using social interaction information would allow filtering messages based on how users communicate, which in turn would allow the creation of a personalized classifier.

3.3 Research Methodology

The research approach applied in this thesis is based on quantitative methods such as simulation and quasi-experiments.

Simulation allows one to see how a system operates under a certain setting, without altering the actual system [58]. In order to do that a model of a system is often constructed. Simulations are used to evaluate models and estimate the performance of the system that the model represents. The type of simulator presented in Chapter 4 and 5 is a discrete-event simulation. The term discrete-event simulation is used to label a model of a system that changes over time, and which contains random variables that are changed during the simulation. Chapter 4 and 5 models a reputation system and simulates the outcome. User voting behavior is used as a random variable, and program popularity distribution as a semi random variable.

Experiments constitute a quantitative research approach "to test the impact of a treatment (or an intervention) on an outcome" [59]. This requires that factors affecting the experiment can be controlled, and can consequnetly be called a controlled experiment. Experiments can be used to compare for instance the performance of different techniques [59, 60]. Experiments use random assignment of study units, e.g. people, to ensure that the study units do not affect the outcome instead of the treatment [61]. *Exploratory data analysis*, or explorative research, is used to investigate little-understood problems, visualize data, and develop questions and hypothesis used in *confirmatory data analysis* (CDA) methods [62]. Experiments and quasi-experiments are CDA methods focusing on the testing of a hypothesis.

Quasi-experiments, compared to experiments, do not involve "random assignment of study units to experimental groups", but are otherwise similar [61, 62]. Random assignment is sometimes not optimal due to e.g. constraints concerning cost, participants, or the design of the experiment [61]. Chapter 6 and Chapter 7 use controlled experiments to compare the performance of using machine learning algorithms to differentiate between unsolicited and solicited software. In Chapter 6 the impact of feature selection algorithms is investigated as well.

3.4 Validity threats

Threats to validity can be divided into four main groups: internal, external, construction and conclusion [59,60]. Each group contains several threats, which sometimes, might not be applicable in all research designs [62].

External validity threats concern the generalizability of the results.

Even if the outcome is true in an experiment setting, the same outcome might be false for a larger scale or in a real world settings [60]. External validity threats are applicable in simulations, since simulations often concern a manmade model of a system [58]. For example, the choice of probability distributions of random variables to use in the simulation is important, which has been addressed in Chapter 4 and 5 by using distributions determined to be applicable in similar situations. The nature of the data set investigated in Chapter 7 impacts the generalizability, and as a consequence it needs to be studied further.

Internal validity concerns experimental procedures. Most internal validity threats concern changes in environment and in participants, and that such changes do not affect the outcome of the experiment [62]. The research investigated in this thesis is of the nature that many of the threats do not apply. Related to this thesis, internal validity threats can be exemplified by the selection threat, meaning that the selection of the population affects the results [59]. This can often be avoided by relying on random sampling from the population. In Chapter 6 this is mitigated by random manual inspection of instances from the population, and in Chapter 7 by using a recognized dataset.

Construction validity threats are the result of inadequate definitions and measurement of variables, e.g. variables defined well enough to be measured [59, 60]. This is less of a problem in any of the included publications as the data measured is not open to interpretations. The data is measured using, within the domain, standardized and accepted measurements. Chapter 4 and 5 use the average distance as a metric.

Conclusion validity threats concern inaccurately drawn conclusions from the data [59, 60]. This is also known as statistical conclusion validity [59]. Examples relevant to this thesis are, e.g. low statistical power or violated assumptions of statistical tests. The first is approached in Chapters 5 through 7 by having a large sample size to base our conclusions on. Throughout this thesis, where applicable, standardized statistical tests are used. In Chapter 6 and 7 this is mitigated using a paired t-test available through the Weka platform [63].

3.5 Contributions

The following sections outline the research contributions that have been published in the articles that constitute Chapters 4 through 7.

Chapter 4, titled *On the Simulation of a Software Reputation System*, presents a reputation system for software that is capable of providing quantitative ratings of software based on qualified user input. A software reputation system (SRS) consisting of expert, average, and novice users is proposed as a complement to let anti-malware companies decide whether questionable programs should be removed. A simulation of the variables involved is accomplished by varying the relative size of the user groups involved, modifying how each user is trusted by the system, specifying an upper limit of the trust factors and accounting for previous rating influence. As a proposed result, a balanced, well-informed rating of judged programs appears, i.e. a balance between quickly reaching a well-informed decision and by avoiding giving a single voter too much power. The contribution of this article is the algorithm used to differentiate between users based on knowledge and the simulator presented, as it is capable of evaluating the algorithm. This chapter contributes to RQ I, in that it allows a way of providing feedback of software behavior, based on qualified user input and allows the separation of users based on input.

Chapter 5, titled *Simulating Malicious Users in a Software Reputation System* is a continuation of the work done in Chapter 4. In this article the usage of a SRS is simulated to investigate the effects that malicious users have on the system. The results show that malicious users will have little impact on the overall system, if kept within 10% of the population. However, a coordinated attack against a selected subset of the applications may distort the reputation of these applications. The results also show that there are ways to detect attack attempts in an early stage.For this study, the simulator had to be extended with a more realistic software popularity distribution and a model of malicious user behavior. The contribution of this article is twofold. First, it suggests that the algorithm for differentiating between user knowledge can be used to detect malicious activities early on in an attack. Second, it suggest that the proposed system is capable of handling a small malicious user base. This chapter extends the answer of RQ I, by suggesting that a reputation system for software is still feasible even with a malicious population.

Chapter 6, titled *Informed Software Installation through License Agreement Categorization*, presents an automatic prototype for extraction and classification of EULAs. The extracted EULAs are used to generate a data set, which is used for comparing four different machine learning algorithms when classifying EULAs. Furthermore, the effect of feature selection is investigated and for the top two algorithms, optimizing the performance using systematic parameter tuning is investigated. The conclusion is that feature selection and performance tuning are of limited use in this context, providing limited performance gains. This shows the applicability of license agreement categorization for realizing informed software installation. There are two contributions of this chapter. First, the algorithm for automatic EULA extraction and processing, enabling less required user interaction. Secondly, the chapter investigates the impact of parameter tuning and feature selection, potentially increasing the performance and, regarding feature selection, adding further processing requirements. This chapter answers RQ II, in that it provides an automatic way of extracting and classifying EULA from software, and shows that the impact of preprocessing is neglible.

Chapter 7, titled *Social Network-based E-mail Classification*, presents an approach to detecting unsolicited e-mail messages using several data sources. A multi-source system that is capable of augmenting a classification model with data about a user gathered from social networks provides additional decision basis. This article presents a system using several social networks when classifying messages. The impact of using social network data extracted from an e-mail corpus is investigated and compared to traditional spam classification. The results suggested in this chapter answers RQ III, allowing users to discern unsolicited e-mails. The contribution of this paper is the use of personal online social network data for classification of e-mail messages, allowing the creation of personalized spam filtering and bootstrapping the learning process.

3.6 Discussion

The techniques investigated in this thesis belong to the category of Knowledge-driven DSS, in the case of Chapter 6 and Chapter 7, or Communication-driven DSS, in the case of Chapter 4 and Chapter 5. The premise was to bring forth expertise of the problem domain as the basis for the suggested decision, either through other users or through mining the instance based on an understanding of the problem. Using the techniques investigated in this thesis, the problem of determining whether an instance of e-mail or software is to be labeled as spam or spyware, can be considered to be semi-structured. Consequently, the final decision must still be made by the decision-maker and can consequently only be suggested by the system. This is also a consequence of the legal nature of the investigated problem. Since the legal status of spam and spyware differ between nations, the grey-zone may change and as a consequence, providing optimal decision support is sometimes impossible.

The utility of spyware can be perceived as low by a user if the primary function of the software does not outweigh the cost of the spyware part of the program. As a consequence, the utility of spyware depends on what behavior and consequences the user condones. This has been the difficulty of labeling grey-zone spyware. Spam can be of a low utility as spam may be deceptive or contain spyware [6]. Spam is also subjective, as what is considered spam can differ between users and even for the same user over a span of time. Processing e-mail messages has a certain cost to users, and aiding the user in estimating the utility of the e-mail message allows users to avoid spam messages.

Techniques capable of being included in a decision support system allowing users to estimate the utility of an instance of software or e-mail are investigated. Combining the techniques proposed in Chapter 4-6 would be a first step towards a better indicator for whether a user should install an instance of software, given the estimated utility. Instead of removing installed spyware — the damage can already be done once spyware is installed — one potential remedy for the problem of spyware could be to aid the user in making an informed decision before the installation process is complete. As a result the spyware application will never gain access to user information.

A software reputation system would most likely need to be attached to an already existing infrastructure, be it a social network or an app-store [1], as attracting users to an application concerned solely with reputation, could be hard. Similarly, the approaches investigated in this thesis should be considered a complement to traditional filtering and detection techniques. First, the approaches investigated dealt with instances where classification has a certain degree of uncertainty, or a personalized approach is preferable. In such instances, traditional techniques have a hard time coping. Second, in cases where the techniques investigated have a hard time making a decision, e.g. in cases where EULAs are not available, complementing techniques are necessary. The automated system presented in Chapter 6 was only able to extract EULAs corresponding to approximately 39% of the applications investigated. As a consequence, the proposed tool should not be used stand-alone, but rather in combination with other techniques. In such situations, to base the suggestion on multiple techniques would be beneficial to the user.

[1]Appstores are common for smartphones and are available for computers on e.g. OS X and soon windows 8, making them a good choice for incorporating a reputation system.

3.7 Conclusion

RQ I concerns how a reputation system could be used for software installation decision support. The main idea was that by providing users with a reputation for an application before installation took place, it was possible to avoid spyware or software that was of low utility. In Chapter 4, a system was presented and a way of weighting user-input based on historical knowledge was investigated. Chapter 5 investigated how the previously mentioned system work when facing a popularity distribution of the programs, and an increasing population of malicious users targeting specific programs. The results of these two chapters were that by weighting user votes according to knowledge it is possible to get reliable ratings of software and as a consequence users have a decision support system capable of determining the utility of a software. The results suggested that the system is capable, on average, of producing a rating approximately ± 1 from an objects correct rating. The system evaluated in simulation has not been tested in a real world setting, and might not necessarily produce similar results.

RQ II concerned preprocessing optimization of machine learning-based text-classification used to estimate utility of software. Chapter 6 used text-classification to investigate a system that is capable of automatic extraction, processing and classification of a EULA. As a consequence, given that an application contained a EULA, it is possible to discern whether the application is spyware with a minimal amount of user interaction. Further, the investigation in Chapter 6 of the impact of feature selection and parameter optimization suggest negligible improvements. The results suggest that EULA analysis, where a EULA is present, can be used to determine class belongings without tuning parameters.

RQ III concerned the impact of social interaction information on machine learning-based text-classification used to estimate utility of e-mail. Chapter 7 expanded on traditional spam-filtering by introducing social awareness to the classification process. Proposed is a system that uses multiple social networks as basis for determining the utility of e-mail messages. Using social networks as a basis for the classification, the results are personalized to each individual user. This system can be used as a complement to traditional spam filtering techniques, which would allow users to determine the utility of e-mail messages before opening them and provide a prioritized reading order. The results of the experiment suggest the social aspects to be useful when determining the utility, but with limited impact when combined with a traditional approach.

The aim of this thesis, put forth in Section 3.2, concerned how computer-based decision support systems can be used to assist users in estimating the utility of a software or e-mail instance. The presented approaches show that it is possible to provide decision support for detecting software that might be of low utility to users. Given the nature of the approaches, e.g. EULA missing in many applications, isolated implementations will yield unsatisfactory results in a real world setting. A combination of the approaches, e.g. to determine the utility of software, could yield improved results. The investigated research suggests the possibility of using machine learning techniques to analyze text and determine the utility of software and e-mail. In the case of messages, there is a benefit of adding a social awareness in that the classification becomes personalized. Additionally, allowing users to share opinions of software would allow informed decision making when trying to estimate the utility of said software. Reputation systems can presumably also benefit from using social network data as a basis for user impact. The conclusion that can be made from the presented research, suggests that it is possible for users to estimate the utility of instances either by sharing opinions with other users via a reputation system or automatically analyzing text embedded in instance.

3.8 Future Work

Interesting approaches could be investigating whether to combine multiple decision bases and how to combine outcomes, e.g. how to combine outcomes from EULA analysis and reputation systems. The use of social network based weighting of opinions in reputation systems is also an interesting aspect to investigate, i.e. whether the opinions of my friends should be more important than those of strangers. The system presented in Chapter 4 and 5 have not been tested in a real world setting, and the generalizability of the study should be investigated.

The research presented in Chapter 7 requires further investigation. Interesting research could include how to weight the data from different social networks, multi-label classification of messages, or context-based classification. The approach should also be investigated using data where identities can be linked to multiple social networks.

3.9 References

[1] Erika Shehan Poole, Marshini Chetty, Tom Morgan, Rebecca E Grinter, and W Keith Edwards. Computer help at home: methods and motivations for informal technical support. In *Proceedings of 27th international conference on Human factors in computing systems (CHI 2009)*. ACM, April 2009.

[2] Xiaoni Zhang. What do consumers really know about spyware? *Communications of the ACM*, 48(8):44–48, August 2005.

[3] Scott Plous. *The psychology of judgment and decision making.* Mcgraw-Hill, 1993.

[4] George J. Stigler. The development of utility theory. I. *The Journal of Political Economy*, 58:307–327, August 1950.

[5] John Stuart Mill. *Utilitarism.* Daidalos, 2003.

[6] Steve Hedley. A brief history of spam. *Information & Communications Technology Law*, 15(3):223–238, October 2006.

[7] Ed Skoudis and Lenny Zeltser. *Malware: Fighting malicious code*. Prentice Hall PTR, January 2004.

[8] James P Farwell and Rafal Rohozinski. Stuxnet and the future of cyber war. *Survival*, 53(1):23–40, February 2011.

[9] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51, 2011.

[10] Peter Szor. The art of computer virus research and defense. *Addison-Wesley Professional*, January 2005.

[11] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim. Evolution of computer virus concealment and anti-virus techniques: A short survey. *International Journal of Computer Science Issues*, 7(6):113–121, November 2010.

[12] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*, 4(3):251–266, 2008.

[13] Mark B. Schmidt and Kirk P. Arnett. Spyware: a little knowledge is a wonderful thing. *Communications of the ACM*, 48(8):67–70, August 2005.

[14] Steve Gibson. Spyware was inevitable. *Communications of the ACM*, 48(8):37–39, August 2005.

[15] Daniel Garrie, Alan Blakley, and Matthew Amstrong. Legal status of spyware. *Federal Communications Law Journal*, 59(1):161–218, January 2006.

[16] Paul McFedries. Technically speaking: The spyware nightmare. *Spectrum, IEEE*, 42(8):72, 2005.

[17] Simon Heron. Technologies for spam detection. *Network Security*, 2009(1):11–15, February 2009.

[18] Bogdan Hoanca. How good are our weapons in the spam wars? *IEEE Technology and Society Magazine*, 25(1):22–30, January 2006.

[19] Lorrie Faith Cranor and Brian A. LaMacchia. Spam! *Communications of the ACM*, 41(8):74–83, August 1998.

[20] Galen A. Grimes. Compliance with the can-spam act of 2003. *Communications of the ACM*, 50(2):56–62, February 2009.

[21] Mark Bender. Can spam. *Monash Business Review*, 3(3):36–40, 2007.

[22] Shari Lawrence Pfleeger and Gabrielle Bloom. Canning spam: Proposed solutions to unwanted email. *Security & Privacy Magazine*, 3(2):40–47, January 2005.

[23] Milton Friedman and Leonard J. Savage. The expected-utility hypothesis and the measurability of utility. *The Journal of Political Economy*, 60(6):463–474, December 1952.

[24] George E Monahan. Management decision making. *Cambridge University Press*, 2000.

[25] Sean B Eom and E. Kim. A survey of decision support system applications (1995-2001). *Journal of the Operational Research Society*, 57(11):1264–1278, 2005.

[26] Daniel J Power. A brief history of decision support systems. *DSSResourcescom, World Wide Web, http://DSSResources.COM/history/dsshistory.html, version 4.0,* March 2007.

[27] Daniel J Power. Supporting decision-makers: An expanded framework. In *Proceedings of Informing Science Conference*, pages 431–436, June 2001.

[28] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, March 2007.

[29] Vibha Gaur and Neeraj Kumar Sharma. A dynamic framework of reputation systems for an agent mediated e-market. *International Journal of Computer Science Issues*, 8(4):1–13, July 2011.

[30] Tom M. Mitchell. Machine learning. *McGraw-Hill*, March 1997.

[31] S M Kamruzzaman, Farhana Haider, and Ahmed Ryadh Hasan. Text classification using data mining. In *Proceddings of International Conference on Information and Communication Technology in Management (ICTM-2005)*, May 2005.

[32] Thiago S Guzella and Walmir M Caminhas. A review of machine learning approaches to spam filtering. *Expert Systems With Applications*, 36(7):10206–10222, 2009.

[33] George Lawton. Invasive software, who's inside your computer. *COMPUTER*, 35(7):15–18, 2002.

[34] Roger Thompson. Why spyware poses multiple threats to security. *Communications of the ACM*, 48(8):41–43, August 2005.

[35] Trisha L Davis. License agreements in lieu of copyright: Are we signing away our rights? *Library Acquisitions: Practice & Theory*, 21(1):19–28, 1997.

[36] Jamie J. Kayser. The new new-world: Virtual property and the end user license agreement. *Los Angeles Entertainment Law Review*, 27(59):59–85, 2006.

[37] Karl J. Dakin. Do you know what your license allows? *Software, IEEE*, 12(3):82–83, May 1995.

[38] Martin Boldt and Bengt Carlsson. Privacy-invasive software and preventive mechanisms. In *Proceedings of Internationcal Conference on Systems and Networks Communication (ICSNC 2006)*, 2006.

[39] Hyon B. Eom and Sang M. Lee. A survey of decision support system applications (1971-april 1988). *Interfaces*, 20(3):65–79, 1990.

[40] Sean B Eom, SM Lee, and EB Kim. A survey of decision support system applications (1988-1994). *Journal of the Operational Research Society*, 49(2):109–120, 1998.

[41] Sean B Eom. Decision support systems research: current state and trends. *Industrial Management & Data Systems*, 99(5):213–221, 1999.

[42] Rosmayati Mohemad, Abdul Razak Hamdan, Zulaiha Ali Othman, and Noor Maizura Mohamad Noor. Decision support systems (dss) in construction tendering processes. *International Journal of Computer Science Issues*, 7(2):35–45, March 2010.

[43] J.P. Shim, Merrill Warkentin, James F Courtney, Daniel J Power, Ramesh Sharda, and Christer Carlsson. Past, present, and future of decision support technology. *Decision Support Systems*, 33(2):111–126, 2002.

[44] Ren Jieh Kuo, C.H. Chen, and Y.C. Hwang. An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artificial neural network. *Fuzzy Sets and Systems*, 118(1):21–45, February 2001.

[45] Cong Tran, Lakhmi Jain, and Ajith Abraham. Adaptation of mamdani fuzzy inference system using neuro - genetic approach for tactical air combat decision support system. In *Proceedings of 15th Australian Joint Conference on Artificial Intelligence (AI'02)*, pages 672–679, 2002.

[46] Jaroslav Mysiak, Carlo Giupponi, and Paolo Rosato. Towards the development of a decision support system for water resource management. *Environmental Modelling & Software*, 20(2):203–214, 2003.

[47] Sean B Eom and Sandy Butters. Decision support systems in the healthcare industry. *Journal of Systems Management*, 43(6):28–31, 1992.

[48] Vipul Ved Prakash and Adam O'Donnell. Fighting spam with reputation systems. *Queue*, 3(9):36–41, November 2005.

[49] Kevin Walsh and Emin Gün Sirer. Fighting peer-to-peer spam and decoys with object reputation. In *Proceedings of 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 138–143, New York, NY, USA, August 2005. ACM Press.

[50] Hong Zhang, Haixin Duan, Wu Liu, and Jianping Wu. Ipgrouprep: A novel reputation based system for anti-spam. In *Proceedings of Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, pages 513–518. IEEE, 2009.

[51] Dongmei Jia. Cost-effective spam detection in p2p file-sharing systems. In *Proceedings of 2008 ACM workshop on Large-Scale distributed systems for information retrieval (LSDS-IR '08)*, pages 19–26. ACM Request Permissions, October 2008.

[52] William W Cohen. Learning rules that classify e-mail. In *Proceedings of AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25, 1996.

[53] Ray Hunt and James Carpinter. Current and new developments in spam filtering. In *Proceedings of 14th IEEE International Conference on Networks*, January 2006.

[54] M Tariq Banday and Tariq R Jan. Effectiveness and limitations of statistical spam filters. In *Proceedings of International Conference on New Trends in Statistics and Optimization*, 2008.

[55] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Technical Report*, 14(1):16–29, February 2009.

[56] Matthew G Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of symposiom on Security and Privacy, 2001 IEEE*, pages 38–49, 2001.

[57] Niklas Lavesson, Martin Boldt, Paul Davidsson, and Andreas Jacobsson. Learning to detect spyware using end user license agreements. *Knowledge and Information Systems*, 26(2):1–23, 2010.

[58] Averill M Law and W David Kelton. *Simulation modeling and analysis*. McGraw-Hill Science/Engineering/Math, 2000.

[59] John W. Creswell. *Research design: qualitative, quantitative, and mixed methods approaches*. Sage Publications, 2009.

[60] Claes Wohlin, Martin Höst, and Kennet Henningsson. *Empirical Research Methods in Software Engineering*, volume 2765 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[61] Vigdis By Kampenes, Tore Dybå, Jo E Hannay, and Dag I K Sjøberg. A systematic review of quasi-experiments in software engineering. *Information and Software Technology*, 51(1):71–82, 2009.

[62] Colin Robson. *Real world research*. a resource for social scientists and practitioner-researchers. Wiley-Blackwell, second edition, 2002.

[63] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Publications, January 2005.

[64] Fabrizio Sebastiani. Classification of text, automatic. *The Encyclopedia of Language and Linguistics*, 14:457–462, January 2006.

Four

On the Simulation of a Software Reputation System

Martin Boldt, Anton Borg, Bengt Carlsson

Abstract

Today, there are difficulties finding all malicious programs due to juridical restrictions and deficits concerning the anti-malicious programs. Also, a "grey-zone" of questionable programs exists, hard for different protection programs to handle and almost impossible for a single user to judge. A software reputation system consisting of expert, average and novice users are proposed as an alternative to let anti-malware programs or dedicated human experts decide about questionable programs. A simulation of the factors involved is accomplished by varying the user groups involved, modifying each user's individual trust factor, specifying an upper trust factor limit and accounting for previous rating influence. As a proposed result, a balanced, well-informed rating of judged programs appears, i.e. a balance between quickly reaching a well-informed decision and not giving a single voter too much power.

4.1 Introduction

Today several hundred thousands of software programs exist, making it almost impossible for a single user to by herself decide what is good and what is bad. Of course tools to prevent and remove viruses and spyware have existed for a long time, but not all malicious programs are found due to juridical restrictions, i.e. the legal status of these applications are questioned, placing them in an grey-zone between good and bad software. This results in a large amount of applications that anti-malware developers are being cautious about removing, due tot the potential for legal retribution. So, a "grey-zone" of questionable programs exists, hard for different protection program to handle and almost impossible for a single user to judge. Also, the availability of preventive software has been limited, already installed malicious software are found and removed but then the damage might already be done.

The inability of traditional anti-malware applications to handle, due to restrictions put upon them, the programs that exist in the previously mentioned grey-zone, leaves user unprotected. A complement, to using anti-malware software for deciding about unwanted programs, is to use a reputation system, i.e. ranking of new and previous unfamiliar software as a method for investigating the "true" appearance of a program. Using professional experts for doing this is both expensive and unrealistic due to the huge amount of non-investigated programs. Instead we propose a pool of ordinary users with different skills making necessary decisions about the quality of different software. However, there is still a need for more traditional anti-malware tools for targeting the clear-cut malware types that by no means could be placed inside the "grey-zone" between good and bad software, such as viruses and worms.

The purpose of this work is to investigate how many and what impact expert users need to have on a reputation system making it reliable, i.e. if it is possible to get a stable system by having few experts compensating for a vast majority of users with limited ability to rate an application. We simulate a reputation system with input from different skilled users and investigate a way of mitigating bad user ratings by using trust factors rewarding good users' good actions and punishing bad actions. The goal of the simulation is to find critical parameters for correctly classifying large number of different programs with a realistic base of different skilled users.

The remaining part of this paper is organized as follows. First we discuss the related work in Section 4.2 and introduce the software reputation system in Section 4.3. We continue in Section 4.4 by introducing the simulator software and in Section 4.5 we present the scenarios. In Section 4.6 we present our results, which then are discussed in Section 4.7. We conclude by stating our conclusions and suggestions for future work in Section 4.8 and 4.9 respectively.

4.2 Related Work

Recommender systems are used to provide the user with an idea of other users' thoughts about products, i.e. whether they are good or bad. These kinds of systems are mostly used in commercial websites suggesting additional products, which the user might consider buying, exemplified in Amazon [6]. Recommender systems are not limited to commercial services, but also exist in other recommendation services such as Internet Movie Database (IMDb) [4]. IMDb uses recommender systems to suggest movies to users based on the opinions of other users that have shown similar tastes. Adomavicius and Tuzhilin provide, in their survey on the subject, a deep introduction of recommender systems, as well as some of the limitations [13].

eBay [3] makes use of a reputation system that allows users to rate buyers and sellers within the system, as a way to establish reputation among users. This reputation system makes it easier for users to distinguish dishonest users from trustworthy users. Experiments conducted by Resnick et al. also show that users with a higher reputation have a higher likelihood to sell items [5], [8]. Since reputation systems rely on the input of the users to calculate the ratings, it has to be able to establish trust between users and towards the system [5], [9]. This is especially important when one considers the fact that the users of a software reputation system will have varying degrees of computer knowledge, and their ability to rate an application will thus be of different quality. There also exists the possibility of a user acting as several agents and actively reporting an erroneous rating in order to give a competitor a bad reputation or increase rating of a chosen object, i.e. a Sybil attack [2]. Even though this can be a potential problem to our proposed system, it is not within the scope of this paper to further analyze such scenarios. Furthermore there exist proposed solutions to this problem, for instance SybilGuard by Yu et al. [14].

The problem of erroneous ratings will, in a system such as IMDb, correct itself over time, but in a system such as the one proposed by Boldt et al. [1], where the intent is to advice on malicious software to users who might not be able to tell the difference, this presents a greater problem. Whitby et al. has put forth an idea of how to solve this problem [12] by filtering the unfair ratings, and their simulations show that the idea has merit. Traupman and Wilensky [15] try to mitigate the effects of false feedback in peer-to-peer applications by using algorithms to determine a users true reputation. However, these ideas might not be ideal under all circumstances, as they add another layer of complexity to the system, as well as another step of work to be done.

J£sang et al. [5] summarize, among other things, different ways of computing the rating of an object and one of the conclusions is that reputation systems originating from academia have a tendency to be complex compared to industrial implementations. We have opted for a simpler system, where the rating is weighted by trustworthiness of the user.

Among simulations done on the area of reputations systems, J£sang et al. has conducted a simulation on an e-market, concluding that reputation systems are necessary in order for the e-market to become healthy [10]. They also come to the conclusion that reputation systems should be configured to forget old ratings in order for new ratings to have impact, i.e. the system should be able to change opinion concerning an object.

4.3 Software Reputation System

As presented in the previous section, ranking of new and previous unfamiliar software is a common method for investigating the "true" appearance of a program before installing it. Using professional experts for doing this is both expensive and unrealistic due to the huge amount of software programs that are developed every year. Instead the opinions are gathered from a pool of ordinary voluntary users that agree to benefit from the common knowledge by providing ratings for the software they are most familiar with. In this way each participant is asked to rate software on a discrete scale (1 to 10) after they have used that software during a certain time-frame, i.e. the user have had time to form an opinion about that particular software program. The ratings given by the system users should be all-embracing, i.e. including different parts such as (but not limited to) the software's features, behavior, usability, stability, performance and privacy aspects.

4.3.1 System Design

We propose a client-server based system where each user has a small client software installed through which it is possible for the user to both send and retrieve information from the central server that handles all software reputation calculations. The client identifies software by calculating hash digests on the actual program file data, e.g. SHA-256. This means that a software reputation is associated with each new version of a program, but the reputation of several subversions can be propagated up to one major version that is then presented to the user. It is also possible to calculate for instance the average rating of a certain software vendor based on the individual reputations of all programs that a particular vendor has developed.

In an attempt to get as accurate ratings as possible from the user, the client software asks the user to rate the software he/she uses most frequently, i.e. the user is familiar with the software and has an opinion about it to base the rating on. Each rating includes one mandatory field that represent an overall rating on some grading scale, in this case [1,10] inclusive. It is also possible for the user to provide additional information, but this is optional. We believe it is of great significance not to ask the users to provide too much information since many users would find this most annoying, and therefore provide random or no feedback at all. However, we believe computer users would accept to rate one or two software per month if they in return get access to all previous users' ratings for software programs that the user is considering installing.

To address the ever-existing problem with participants that provide false information to a collaborative system we incorporate user-individual trust factors (TF). This means that each user is assigned a TF that states the amount of influence each user have in the system. New users are always given the lowest possible TF, but as they use the system and prove to be trustworthy this value increases. Each time a user uses the client program to submit a rating it is forwarded to the reputation server for further processing. On the server-side the rating is compared to the average of all previous ratings on that particular software and if it is close then the user's TF is increased, otherwise it is decreased. That way the TF of users that provide accurate ratings increase which give them more influence in the system, while it decrease for the rest of the users. Although the effect of this implementation is that the input from some users is amplified to dominate a large portion of the overall system, we believe it is important to include all users' votes when calculating the resulting software ratings in the system. This way, even non-expert users such as novice and new users can rest assures that their voice is listened to and is important.

It is of significant importance to make sure the users' privacy is sufficiently protected in a software reputation system, since it handles sensitive information about what software each user have installed on their computer and their associated ratings. A situation where it would be possible to combine IP addresses with the information about what software these computers include could for instance reveal which computers that are vulnerable to certain remotely exploitable vulnerabilities. In addition to this it is also important to protect users' privacy since one of the main goals of a software reputation system itself is to assist users in protecting against potentially malicious programs that invade privacy. We therefore need to make sure that the system does not intrude on users' privacy more than absolutely necessary. However, we still need to store some minimal amount of information about the user to address the problem with vote flooding, i.e. we need to distinguish between unique users' votes for each software to guarantee that duplicate votes do not occur. A thorough description of the techniques and design choices used for this software reputation system is available in [1].

4.4 Software Reputation System Simulator

In this section we start by describing the design and workings of the simulator and then move on to explain how we modeled the users in our experiments.

4.4.1 Simulator Design

The simulator itself was implemented in Java and all configuration of the simulator is carried out through configuration files that allow the operator to fine-tune every aspect of the scenario that should be simulated. The simulator is deterministic, meaning that it is possible to rerun a scenario several times and always get the same results, or more interestingly to change a certain variable in the scenario setup and be sure that the changes in the end-result are due to the alteration of that particular variable.

Individual objects represent users and software that are simulated, i.e. one Java object per simulated user and software. These objects are stored in two different linked lists that keep track of all user and software objects. A simulation basically consists of iterating through the linked list of all user objects in sequence, allowing each user to rate a randomly selected software object, until the correct number of votes has been simulated. An important addition to this process is that the linked list of all user objects is shuffled before each iteration proceeds. At certain intervals, for instance every 10% progress, the simulator outputs various degrees of statistics depending on the particular configuration.

Each software object includes variables that store information about its "correct" rating, the number of votes it has received and the sum of all weighted votes, which makes it possible to calculate the software's weighted average rating as explained in Equation 2 in the next subsection. The "correct" rating mentioned above is used for two purposes in our simulator. First, it is used for evaluating the accuracy of the simulated reputation system. Even though such a correct rating might not exist in the real world due to users' subjective beliefs, we use them as a way to evaluate the accuracy of the simulated reputation system. Secondly it is also used when constructing the user's vote as described in the next subsection.

The evaluation of a simulation consists of summarizing the absolute distance between software's correct rating and weighted average rating, and finally dividing it with the number of software included in the simulation. The resulting value is the evaluation score (ES), i.e. the average distance from all software programs' correct ratings. This score represent how accurate the simulated software reputation system is when providing software ratings to its users. One should always strive to reach an as low ES as possible, since an ES of 0.0 represent that the software reputation system on average provide its users with ratings that are 0.0 votes from its correct value. In other words bang on target. In the next subsection we present how the users in the simulations are modeled.

4.4.2 User Models

We have divided the simulated users into three groups based on their technical knowledge and accuracy in rating software. Each user simulated belongs to exactly one of these groups, which determines the users voting variance. Figure 4.1 shows each groups' voting variance (or error rate), which lies within the interval [+5, -5] inclusive. The expert users in Figure 4.1 rate software correctly 50% of the times, and in the remaining part rate the software either one step below or above its correct rating, i.e. the expert users always manage to rate a software within a 1 step wide window around its correct rating. The second group is the average users that tries to rate software correctly, but with lesser accuracy than the experts, i.e. they rate up to 3 steps above or below the correct rating due to lack of skills. Still an average user is better than a novice user that has an error margin of 5 steps above or below the correct rating. Figure 4.1 also shows, as lines, the actual outcome of the distribution during our simulation. The discrepancy of these values are due to problems of giving the worst rated grades for certain types of programs that already are close to one of the rating scale borders. In such cases a new vote variance is randomized based on the user's voting distribution in Figure 4.1, hence the greater probability for a voting variance close to 0.

In addition to the users' own ability in rating software it is also possible for the simulator to simulate that they are influenced by previous user's ratings. This would for instance occur when a user is unsure what rating to assign a certain software and therefore use that software's current average rating as guidance when making up his/her mind. In our

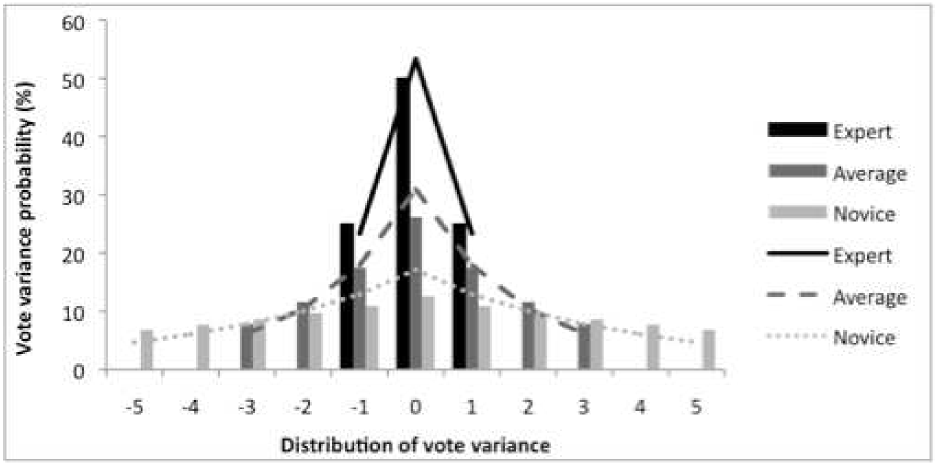


Figure 4.1: The voting variance for each of the three simulated user groups, which lies between +5 and -5 grades away from the software's correct rating. The modeled voting variances are shown as bars, and the actual ones as lines.

simulations we refer to this as the previous rating influence (PRI), which is represented as a number on the continuous scale [0-1] inclusive. We argue that the PRI effect increases as users become less confident about how to rate software. Experts are not very influenced by the already existing rating and thus have a relatively low PRI, in this case 6.25%. The group of average users is more likely to be influenced, thus earning them a PRI of 12.5%. The novice group on the other hand will most likely be very influenced by the already existing rating, and be more inclined to give a rating that is similar to the existing. To simulate this we give the novice group a PRI of 25%. As shown in Equation 4.1. below users' votes are generated by adding the user's vote variance to the software's correct rating, which results in that users from the different groups rate software differently.

$$Vote = (1 - PRI) * (CorrectRating + VoteVariation) + PRI * AverageRating \tag{4.1}$$

When no PRI is used a user's vote is calculated by simply adding the software's correct rating with the user's randomized vote variance. However, when PRI is used the vote is instead pushed towards the software's average rating to various degrees, based on the amount of PRI that is simulated.

An important aspect in the simulations is how to adjust the users' trust factors. The simulator allows its operator to tune four different variables that directly control how the TF is being calculated. First of all the operator has to decide if the TF should increase or decrease in an exponential or linear fashion. Secondly, decide what the change-rate should be, e.g. if a linear value of 2.0 is used then TF would increase or decrease by 2.0 based on whether the user manages to pinpoint the software's correct rating or not. If, on the other hand, an exponential value of 1.25 is used the user's TF will either increase or decrease with a factor 1.25 based on the current value. The third variable that is available to the operator is the potential to include a maximum level, or ceiling, which the TF cannot exceed. Finally it is also possible to have the TF decrease faster than it increases by enabling the decrease factor (DF), i.e. a DF of 1.5 will result in that a user's TF decreases with a factor 1.5 more than it increases. The DF could be used as a sanction method against misbehaving or cheating users. However, it has not been further investigated in the experiments presented in this paper.

4.4.3 Simulation steps

During the initialization of the simulator each program is randomly assigned its "correct" rating which is used for evaluation purposes. Next, the users are assigned to the simulated groups according the proportions defined in the configuration files. Then the simulation starts and executes according to the following steps:

1. Shuffle the list of users
2. Sequentially select each user from the list
3. Randomly select a software
4. Randomize new vote variance for user
5. Create vote (Equation 4.1.)
6. Increase vote counter by 1
7. Update software's weighted average (Equation 4.2)
8. Update user's trust factor
9. Repeat for each user in list
10. Repeat until specified number of votes are reached

The software's weighted average score is calculated based on both the user's vote and trust factor, as explained in Equation 2. This renders in that users' votes are being weighted differently based on their individual trust factors, i.e. amplifying the votes from trustworthy users.

$$AverageRating = \frac{\sum (vote_i * TrustFactor_i)}{\sum TrustFactor_i} \tag{4.2}$$

After each vote the user's TF is updated based on how far away from the software's current weighted average the vote is. If the vote is exactly the same as the weighted average the TF is increased, while it is kept as is if the vote is 1 step above or below. However, if the distance is further than 1 step the TF is decreased.

4.5 Simulated Scenarios

As seen in the background section, ranking of new and previously unfamiliar software is a common method for investigating the "true" appearance of a program before installing it. Using professional experts for doing this is both expensive and unrealistic due to the huge amount of non-investigated programs. Instead a pool of ordinary users with different skills is proposed where each participant repeatedly votes between 1 and 10 before new programs are installed. The voting is based on the user's skill, but also on the previous rating of the program. A skilled user may improve his own reputation by repeatedly giving votes close to the "true" value, i.e. similar to the professional expert. By doing so it is possible to increase the value of the vote either by a linear or an exponential increase.

In a recommendation system different actors may appear. We used the previous described groups of experts, average and novice users in the simulation of our reputation system. All users have one equal valued vote to start with and are supposed to repeatedly rate new software. All simulations in this work include a fixed population of different skilled users with 9.4% experts, 27.1% average users and 63.5% novice users. We decided to use these estimates based on the PEW Internet & American Life Project's statistics of user demography of information technology users [7].

Within this population all groups give a vote based on actual skills, and in some cases also based on the influences from previous voters, i.e. the weighted average for that particular software. Some of the simulations also measures what effects scaling up or down the proportion of different users have on the system, e.g. how system accuracy is affected when scaling down the number of expert users by half.

The above-mentioned groups were simulated for one million users voting for 100000 different programs, i.e. it is unlikely that one user will vote more than once for a single program. We chose to include 100000 programs based on the number of software application included in Web-based reputation systems, e.g. Softonic [11]. One million users is argued to be a realistic number due to the fact that such a system is globally accessible and therefore benefiting from network effects. The scenarios that we simulate in this paper include 48 and 96 million votes, which represent a two or four years use of a system with one million users, and an average voting frequency of two votes/month.

4.6 Results

First we investigate how big an impact we may give a single voter without looking at the result other voters have given for the population of voters described above. Next, we look at the impact of previous rating influence where the different groups of voters are more or less influenced by the judgment already done. Then the proportions of experts, average and novice users are varied. Finally we investigate how system accuracy is when the correct rating for 25% of all simulated software programs are known before the simulation is started, i.e. that they are bootstrapped with the correct rating.

4.6.1 Trust Factors and Limits

Different trust factors varying in range from 1.05 to 2.0 were investigated with either linear or exponential increase. Each group of users starts with a TF set to 1.0 with an increase of the chosen trust value for each vote that is placed within one step above or below the software's current average rating. If the vote is up to one step away from the correct value the TF is left unchanged. Otherwise, i.e. two steps or more, the TF is decreased by the same trust factor. A user can never have a TF lower than 1.0. Each participant voted 48 times each and in all 1 million users voted for 100000 programs in this simulation. Figure 4.2 shows the outcome for

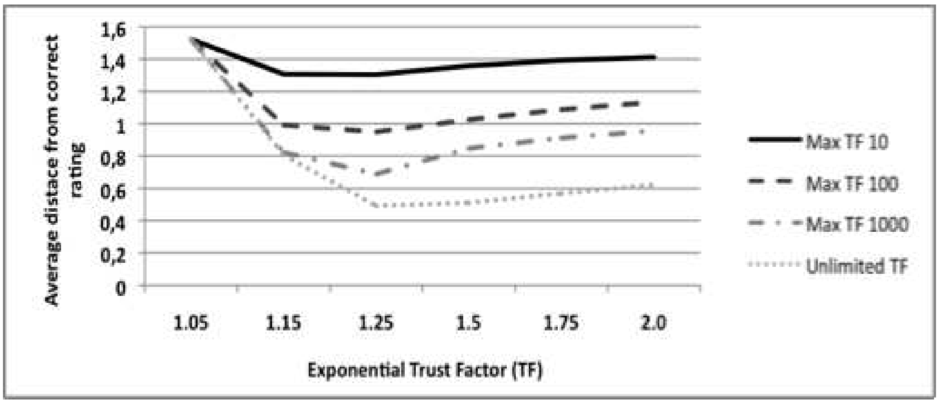


Figure 4.2: 9.4% expert, 27.1% average and 63.5% novice users voting with a varying trust factor during 48 votes each.

the chosen number of different users and various maximum TF limits. The linear outcome for different TFs shows a very limited improvement with increasing TFs and thus is not further investigated. The lowest average distance was reached for an exponential TF or 1.25 where both lower and higher TF showed worse performance. For this reason we decided to use a 1.25 exponential TF during the rest of the simulations.

Next we investigated the need for a maximum limit of the user's TF. Figure 4.3 shows what happens when the following limits are specified for the TF; 1, 10, 100 and 1000 and unlimited. An unlimited TF settles around 0.5 from the correct value but may give a single voter an unproportional big impact on the voting system. Limiting the TF to 1000 is a reasonable compromise where the first 25 votes will behave as unlimited and settles around 0.7 with a slightly increase during prolonged voting time. This has to do with a relative TF increase of less skilled average users due to the expert reaching the 1000 TF limit as shown in the next figure. Figure 4.4 shows the outcome for each category on a logarithmic scale. The novice user hardly reaches above 1 where the average user has a small but constant increase. However, the TF of the expert users reaches 10, 100 and 1000 respectively when these limits are

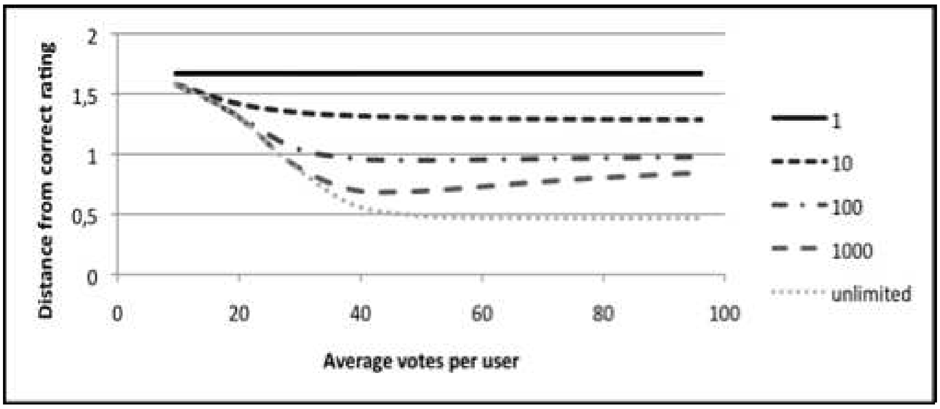


Figure 4.3: 9.4% expert, 27.1% average and 63.5% novice users voting with a 1.25 exponential trust factor during 96 votes each.

specified, and reaches 1 million after 48 votes when no limit at all is used. So, in all cases the expert voter has a dominant position, but not a 100% voting accuracy. As even expert users commit errors when voting they should not be given unrealistically high TF. In our setting an expert is predicting the correct value 50% of the time, i.e. the impact from an expert with huge TF giving a wrong value distort the result from the correct decision. The "knee" on the curve associated with the experts in Figure 4.4 is due to the lack of measurements for TF limits between 10000 and unlimited.

4.6.2 Previous Rating Influences

Some commercial recommendation systems make previously given votes available to the users when they decide on their vote, i.e. the previous rating influence (PRI). This can give both positive and negative consequences that should be considered. The different user groups have different levels of knowledge, and thus are guided by the already existing rating to different degrees. To measure the effects that PRI have on a software reputation system we simulated a scenario where each user-group

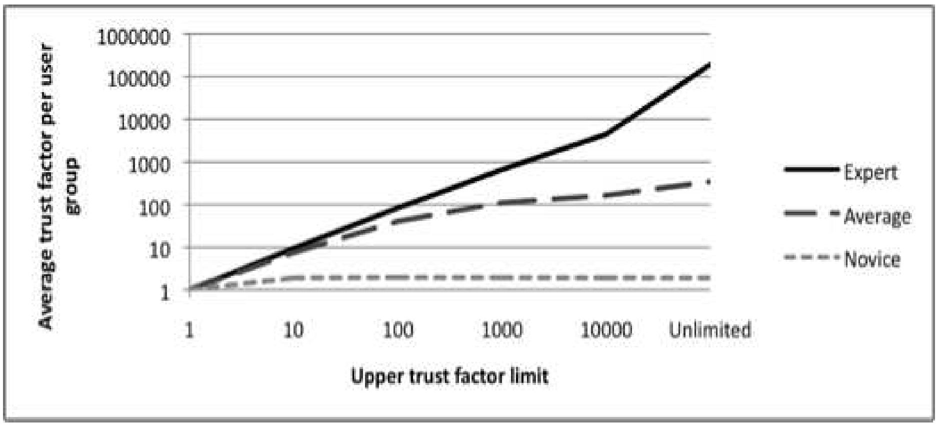


Figure 4.4: Number of votes for each user group with different trust factor limits during 48 votes per user.

were influenced by 1/16, 1/8 and 1/4 for expert, average and novice users respectively. When for instance a novice user rate a software his vote is to 75% decided based on his level of knowledge and to 25% based on that program's current average rating. Figure 4.5 shows the results of this simulation. As seen in Figure 4.5, there is a slight improvement in the beginning of the simulation when PRI is used. However, performance then stabilizes around the same levels as when no PRI is used. There is for instance no noticeable improvement in the case with a TF limit of 1000 compared to the results in Figure 4.3. When activating PRI in the simulation both novice and average users will improve their marking ability, which in turn result in that their TF increases. This can be seen in Figure 4.6 where the trust factor of both the novice and average users has increased several times compared to the results in Figure 4.4. The results in Figure 4.6 show that the TF of all user groups has increased when PRI is enabled. This can be attributed to the fact that for instance the novice group has a higher tendency to follow the already set ratings, i.e. the results from the average and expert users. Due to this, they will rate an application more accurately, which will increase their trust in the system. As already stated, this results in a higher TF

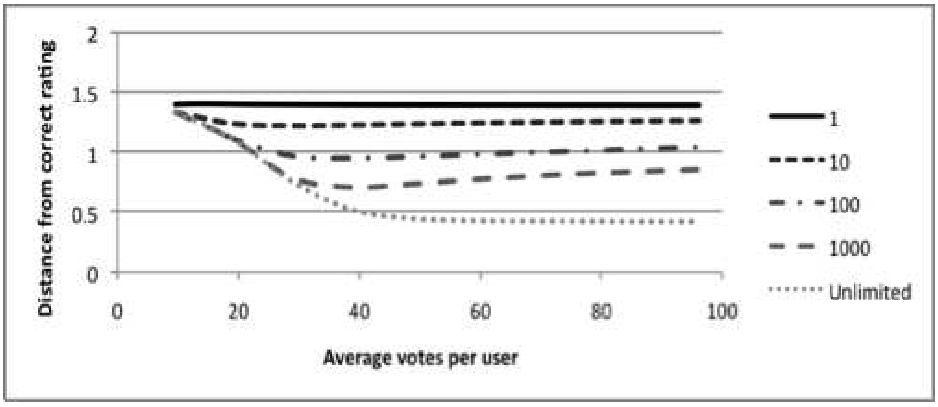


Figure 4.5: Simulation with a previous rating influence (PRI) of 6.25%, 12.5%, 25% for experts, average and novice users. Also a exponential trust factor (TF) of 1.25 was used and 96 votes per user, which corresponds to four years of system usage if each user votes on average twice a month.

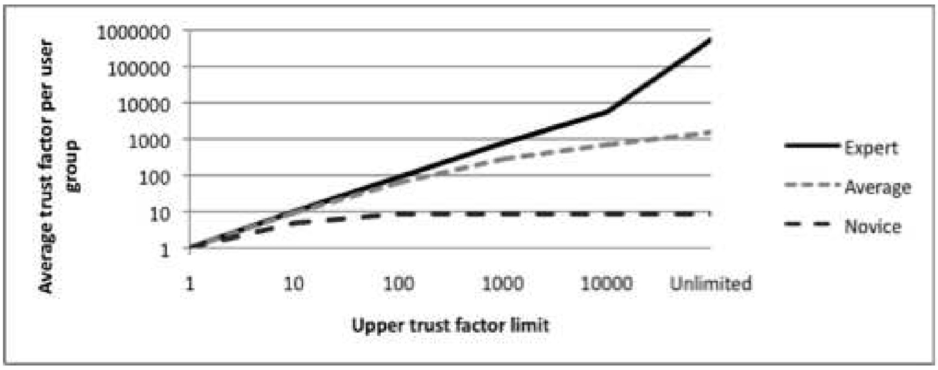


Figure 4.6: Development of the trust factor (TF) for each user group when previous rating influence (PRI) is enabled. These results are based on an exponetial trust factor of 1.25 without any TF limit.

for the novice group. Based on the size of the novice group they will have a higher impact on the system. Even though the novice group has a higher tendency to follow the average ratings, they will still introduce votes that are distanced from the correct rating of software, which deteriorates the overall system accuracy. We have also simulated scenarios with higher PRI values for all user groups, but without any significant improvement of the system accuracy. These results therefore show that the system accuracy is not significantly improved when PRI is enabled and that the reputation system is more stable without it.

4.6.3 Demography Variations and Bootstrapping

Figure 4.7 shows how the user demography of the simulated users affects the system accuracy. We varied the size of the different user groups around the survey results presented by PEW Internet & American Life Project. We can see that the size of the expert and average groups clearly make a difference in the beginning of the simulation, i.e. higher number of expert and average users increases system accuracy. However, as the simulation progresses and the TF of each simulated user is fine-tuned the system accuracy increases for all user constellations. An increased number of experts perform better than increasing the number of average users in relation to novice users. To improve the system accuracy further we also simulated a scenario where already available software reputations were collected from third parties and used to initialize the software reputation database before it was put into use. Figure 4.8 shows how the accuracy of the system is affected when 25% of the software inside the reputation database is bootstrapped with trusted data with a total value of 5000, i.e. equal to five votes from full-fledged experts. When compared with Figure 4.7 it is clear that the system accuracy is positively affected in the beginning of the simulation, but as the simulation continues the difference between whether bootstrapping is used or not decreases. We also ran a set of experiments without any expert users at all just to investigate whether or not the software reputation system

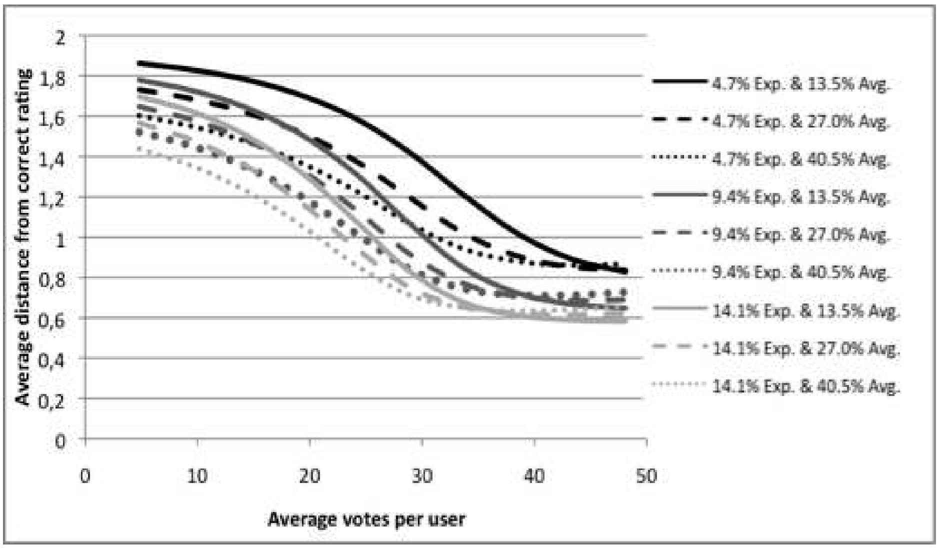


Figure 4.7: System accuracy for several different user group constellations with varying number of expert and average users. In this experiment the previous rating influence was turned off and an exponential factor of 1.25 was used together with a trust factor limit of 1000.

would still function properly in such a scenario. When omitting all expert users, leaving 27% average and 73% novice users, the system shows an accuracy score of approximately 1.8. in the beginning of the simulation which then improves towards 1.2. Finally we also simulated the use of a software reputation system during an extended period of time, in this case 1200 votes per user, without seeing any tendency of degraded system accuracy. Even though the scenario is questionable since all software programs are identical it doesn't show any evidence of decreased performance, i.e. the reputation system seams to be stable. In fact the accuracy is continuously improving through out the simulation during this extended simulation, and the overall system accuracy stops at 0.95. During the whole simulation the TF of the novice users are quite low, with an average of 2.8. At the same time the TF of the average users stabilize around 890 while the experts quickly reach the TF limit of 1000.

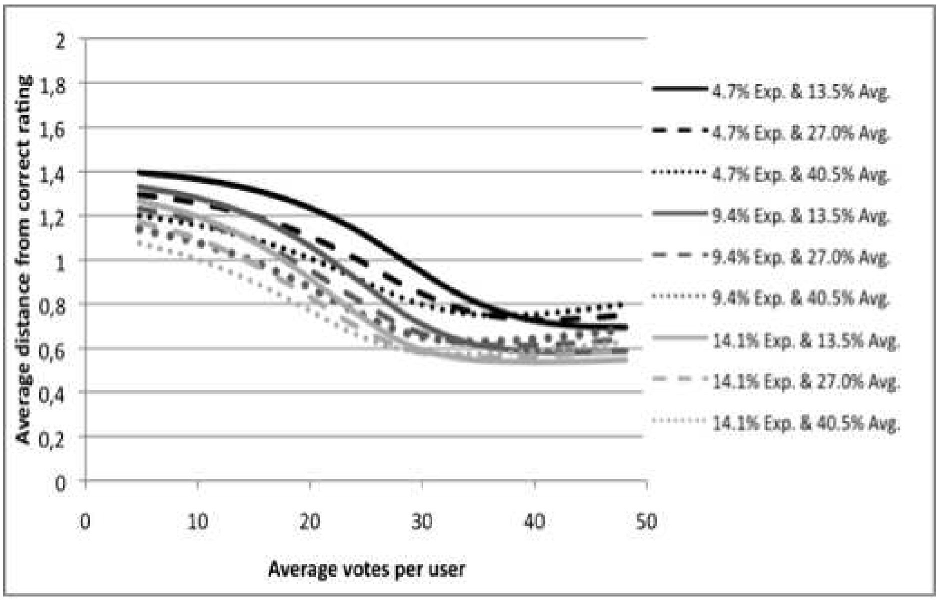


Figure 4.8: The same setup as in the previous diagram but in this experiment we bootstrapped the reputation for 25% of the software with reputation data from trusted sources.

4.7 Discussion

The idea behind the simulated reputation system is to reward users that provide accurate ratings and punish faulty or not properly thought through decisions, which will improve the software reputations within the system. The experiments assume that there exists a “correct” rating of each software for initial settings and evaluation purposes. However, whether such a correct rating exists in a real world setting is not necessarily true. It is therefore hard for a single person to define exactly what the correct rating for the particular software should be, but based on several users’ ratings it is possible to come to a common compromise, which then is used as a baseline when deciding whether or not to increase or decrease users’ TFs.

In this work we have used simulation as the means to identify how the users' TF should be adjusted within a software reputation system to reach an accurate, stable and sustainable system to mimic the view of a professional expert. In this investigation we primarily look at the behavior of the voter. From a single program's point of view different skilled voters may end up voting close to the "correct value", i.e. on average votes below and above this value are compensated. So, the average value for a single program is better than the user's performed absolute distance value. By introducing a trust factor and/or previous rating influence the overall performance for various groups of users was increased, i.e. the absolute distance value coming closer to the average value. This was true even in groups dominated by unskilled novice and average users making the outcome of the voting procedure more robust.

Unlike professional experts the simulated population consists of different skilled users voting twice a month during two or four years. In the beginning of the simulations each voter has given a very limited amount of votes, i.e. only a small fraction of all available programs are being rated by a single voter. The simulation results show an exponential increase of the TF to be better than a linear increasing, and that a factor of 1.25 was most promising for an accurate system. Through the simulations we also found that an upper TF limitation of 1000 is preferable. If this limitation is being omitted the TF of the expert users quickly increases to levels that make the whole system unstable because of these users' small, but still existing, rating fluctuations. Therefore the upper TF limit creates a balance between fast reaching and well-informed ratings, without giving a single user too much influence. In a real situation other factors, such as malicious behavior, makes this argument even more important within the system. If a single user can get extremely high TF values it is possible for malicious actors to use this to manipulate the rating of the particular software for their own personal gain.

Intuitively it might seem interesting to allow users to see all previous ratings (PRI) when they make up their mind about how to rate a certain software, since this at first could be thought to improve the decisions of the less skilled users. However, if the system makes use of TFs this is not the case since novice and average users will improve their TFs, i.e. their normal voting variance will exceedingly negative influence the program evaluation. Through the simulations we also investigated how the proportion of experts, average and novice users affects the system accuracy and stability. By increasing and decreasing the number of experts and average users we were able to draw the conclusions that the system accuracy improves as expert users sign up to the system. For all investigated populations the system accuracy improves from initial not-to-well judgments in the beginning of the simulation to closer to the correct ratings, for instance when 85% novice users and less than 5% experts are being simulated.

In our simulations we have also showed that it is possible to improve the initial system accuracy before the system is made publicly available by bootstrapping the database with trusted reputation data. Such bootstrapping data could for instance be gathered from web-based services available on the Internet. Furthermore we also show that the system is stable and accurate when users submit ratings with a higher frequency. In this case when each user submits on average 1200 votes over a four years period, i.e. about one votes per day.

When summarizing all simulated scenarios we come to the conclusion that it is possible for computer users to rely on a stable software reputation system that assist them when identifying well-reputed software and as well when avoiding questionable or malicious software programs.

4.8 Conclusions

A software reputation system consisting of expert, average and novice users was simulated as an alternative to let anti-malware programs or dedicated human experts decide about questionable programs. Within the simulated population, the different skilled users voted twice a month during at most four years. Each voter starts with a single vote and by varying the increase of the trust factor (TF), the upper TF limit and previous rating influence (PRI), with a resulting balanced, well-informed rating of judged programs as the proposed outcome. An exponentially increased TF was better than a linear, and a system that allowed voters to see previous users' votes performed better, i.e. with PRI. More precisely an exponential increase of 1.25 for the TF with a TF limit of 1000 within an experimental setting of less than 5% experts in a population exceeding 80% novices still performed well. Such a setup allowed a balance between quickly reaching a well-informed decision and not giving a single voter too much power. In our opinion the reputation systems will become a more commonplace advisory tool in the future with the possibility to provide an advice to the user that most likely will be helpful, i.e. being able to handle erroneous, good and bad ratings, and without losing the integrity of the system.

4.9 Future Work

Our simulated environment lacks some real world parameters that will be further investigated as the work progresses. First, we will simulate various attack schemes that are used by malicious actors to gain control over the reputation system. Secondly, we will investigate how system accuracy is affected when users are handled more dynamically, e.g. when new users join up during the simulation and that established users' leaves. Finally, this should also include scenarios where new and unknown programs are being added on the fly or when old programs are being overridden.

4.10 References

[1] M. Boldt, B. Carlsson, T. Larsson and N. Lindén, Preventing Privacy-Invasive Software Using Collaborative Reputation Systems, Vol. 4721 of Lecture Notes in Computer Science, 2007.

[2] J. Douceur. The sybil attack, In the Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

[3] eBay, `http://www.ebay.com` (2009-06-16).

[4] Internet Movie Database, `http://www.imdb.com` (2009-06-16).

[5] A. Jøsang, R. Ismail and C. Boyd, A Survey of Trust and Reputation Systems for Online Service Provision, Decision Support Systems, Vol. 43(2), pp. 618-644, 2007.

[6] Amazon, `http://www.amazon.co.uk/` (2009-11-13).

[7] PEW Internet & Americal Life Project: A Typology of Information and Communication Technology Users, `http://www.pewinternet.org/Reports/2007/A-Typology-of-Information-and-Communication-Technology-Users.aspx` (2009-06-16).

[8] P. Resnick et al., The value of reputation on eBay: A controlled experiment, Springer Journal on Experimental Economics, Vol. 9(2), pp. 79-101, 2006.

[9] S. Ruohomaa, L. Kutvonen and E. Koutrouli, Reputation management survey, In the Proceedings of the 2nd Second International Conference on Availability, Reliability and Security (ARES), 2007.

[10] A. Jøsang, S. Hird, and E. Faccer. Simulating the Effect of Reputation Systems on E-Markets, Vol 2692 of Lecture Notes in Computer Science, 2003.

[11] Softonic, `http://softonic.com` (2009-06-16).

[12] A. Whitby, A. Jøsang and J. Indulska, Filtering out Unfair Ratings in Bayesian Reputation Systems, In the Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems AAMAS, 2004.

[13] Adomavicius and Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on Knowledge and Data Engineering (2005) vol. 17 (6) pp. 734 - 749.

[14] H. Yu et al. Sybilguard: Defending against Sybil Attacks via Social Networks. ACM Proceedings of the 12th SIGCOMM, 2006.

[15] J. Traupman and R. Wilensky. Robust Reputations for Peer-to-Peer Marketplaces, Vol. 3986 of Lecture Notes in Computer Science, 2006.

Five

Simulating Malicious Users in a Software Reputation System

Martin Boldt, Anton Borg, Bengt Carlsson

Abstract

Today, computer users have trouble in separating malicious and legitimate software. Traditional countermeasures such as anti-virus tools mainly protect against truly malicious programs, but the situation is complicated due to a "grey-zone" of questionable programs that are difficult to classify. We therefore suggest a software reputation system (SRS) to help computer users in separating legitimate software from its counterparts. In this paper we simulate the usage of a SRS to investigate the effects that malicious users have on the system. Our results show that malicious users will have little impact on the overall system, if kept within 10% of the population. However, a coordinated attack against a selected subset of the applications may distort the reputation of these applications. The results also show that there are ways to detect attack attempts in an early stage. Our conclusion is that a SRS could be used as a decision support system to protect against questionable software.

5.1 Introduction

Among the vast horde of software available to the users today, the user is expected to differentiate between various types of software. For example, the user is supposed to be able to recognize software that bundle other programs in order to exploit the user. These programs are often of a dubious nature, monitoring the users habits and selling the information to third parties, alternatively displaying various advertisements that correspond to the users interests [14]. The nature of these programs often makes them fall under categories such as spyware or adware [5].

As the legal status of these programs differ between different juridical territories, these programs fall in a legal grey-zone and as a result the anti-virus industry has hesitated to remove them[1]. Regardless of this, there still exist tools that operate in a manner similar to anti-virus programs available, commonly known as anti-spyware programs. However, there are two main problems with these applications. First, due to the legal status of the programs in question, there is a concern that anti-spyware programs, because of pressure from software developers, will catch not all dubious applications. Second, many of today's anti-spyware programs are such that they will only remove already installed spyware, failing to prevent new spyware from installing. Thus the damage might already be done when anti-spyware applications detect the spyware.

We have proposed using a software reputation system (SRS) as a complement to traditional anti-spyware tools, in order to mitigate several of the questioned aspects of the traditional approach [2], [3]. The idea behind SRS is that users should provide ratings of programs that they use, thus providing a collaborative platform for judging applications whilst also removing the legal pressure that traditional anti-spyware developers are subject to. The users will, when trying to install applications be presented with information on how other users

[1] `http://www.benedelman.org/spyware/`

have rated the program in question, i.e. providing a preventive mean of allowing informed user decisions. This would provide a mean of preventing the damage done by spyware, as opposed to trying to clean up afterwards.

In this paper we rely on a custom-made simulator for simulating the use of a SRS with users entering and leaving the community dynamically. In this setting we investigate the effects of having a portion of the user-base participating in an attack on a small number of the applications, where they try to maximize different variables for distorting the rating within the SRS. We investigate both the effects on the overall system and for the targeted software programs.

In section 5.2 we present the related work, which in section 5.3 is followed by the background of the study. In section 5.4 we present the simulator and how the experiment was designed. In section 5.5 the results from the simulation are shown. The results are discussed in section 5.6 and conclusions are presented in section 5.7.

5.2 Related Work

Reputation and recommender systems have increased in popularity since their introduction in the early 1990s, much due to the rapid spread of Internet connectivity. One of the earliest systems was GroupLens that were presented in 1994 by Resnick et al. [12], which enabled automatic filtering of newsgroup content based on the community users' feedback.

Reputation systems and recommender systems are related to each other, but the main difference is that reputation systems take explicit ratings from the users [9]; while recommender systems instead rely on events such as previous purchases of some goods, e.g. books [13]. A well known example of a reputation system is the Internet Movie Database, IMDb, that compile overall movie scores from individual community users' movie ratings. One example of a recommender system is Amazon.com that recommend potentially interesting goods to customers based on for instance previous purchases.

Reputation systems are vulnerable to attack from stakeholders that have much to gain if being able to tweak such systems. As a result there exists several attacks, which Marmol and Perez elaborate on [10]. The attacks range from individual to collaborative in nature, which attack either the whole system or a subset thereof. Some attacks also include camouflaging techniques based on for instance oscillating or random behaviors that allow the malicious users to behave trustworthy except during brief raids. For each attack Marmol and Perez present the cost and detectability, and a conclusion is that an attacker that can cope with medium to high costs can generate attacks with low detectability.

Another taxonomy of the attacks within reputation systems are presented by Hoffman et al. [6], and one conclusion they draw is that it is the open nature of reputation systems that allow for various attacks, which are hard to defend against. Reputation systems are vulnerable to these attacks according to Hoffman et al.:

- Self-promoting, attackers falsely manipulate their own reputation by increasing it.
- Whitewashing, attackers abuse the system but manage to escape the consequences, e.g. by resetting their reputation through some system vulnerability or by rejoining with a new identity.
- Slandering, attackers manipulate the reputation of a victim by reporting false data to lower the victim's reputation.
- Orchestrated, attackers apply several of the above mentioned attack techniques in a sophisticated attack.
- Sybil, a single attacker manage to create a large number of sybil identities within the community that each represent one vote, i.e. allowing the attacker to have several votes.

- Denial of service, attackers manage to subvert the underlying mechanisms of the reputation system.

Both Hoffman and Marmol Perez also present defense strategies as a way to defend against the mentioned attacks. Such defense strategies are composed by a number of techniques, such as heuristics, statistics, randomization and cryptography. One example of an effective defense strategy that is used to protect against file pollution in file-sharing networks is Credence [15], which make use of heuristics, statistics, and cryptography. Finally, Josang and Golbeck stress the importance of robustness in reputation systems, and they also present some insights about the problems involved in developing a standardized test environment for evaluating all different types of reputation systems [8].

5.3 Background

In a previous study we investigated the feasibility of a software reputation system in regards to the individual trust factor of users as well as the accuracy of the system given the environment [2]. We found that using user-based trust factors allows separation between different user groups based on their experience, i.e. accuracy in rating software. The individual trust factors reward users with more experience, resulting in an overall stable system.

The previous study was done by implementing a simulator that incorporated user models with different skill levels, which allowed us to simulate various scenarios. In this study we will investigate the effect from malicious user within a SRS. Also, unlike the previous study, users are allowed to enter and terminate their participation within the system, i.e. we are simulating a dynamic system. These scenarios are further explained in section 4 and the simulator is thoroughly explained in [2].

One important difference between a SRS and for instance IMDb is the time constraints imposed on SRS, i.e. the occurrence of an erroneous rating would potentially trick users into installing spyware or keep them from installing clean software were the former having more impact. This could be done by faking a large number of community users that provide erroneous ratings on a single program, i.e. a Sybil attack that result in the rating of the program being altered to the liking of the attacker.

5.3.1 Simulator Tool

We have in our simulator divided users into different groups consisting of expert, average and novice, where the first group is most likely to vote correctly and the last is least likely. This is done in order to simulate a difference in knowledge between users. These users can rate a program on a 10-graded rating scale.

In order to differentiate different users based on knowledge, each user has a trust factor, which affects the impact of the vote. Every user has an initial trust factor of 1.0 and the factor can never be lower than that. In order to measure the accuracy we randomize a value for each application in the simulator to act as the "correct rating", which we evaluate against. The deviation between the correct rating and the end result we named evaluation score (ES). The trust factor is changed based on the vote cast, e.g. if a user cast a vote that is deemed "good", i.e. the user's vote is within 1 step from the current arithmetic average rating of the software object rated, then the trust factor is increased by a pre decided change factor. However, if the vote is deemed erroneous, i.e. the user votes outside the allotted steps, the trust factor is lowered by an equivalent amount. Thus users, who over time provide more accurate votes, have a bigger impact on the system than an unskilled user.

In previous simulations we found that using an exponential change factor of 1.25 together with a maximum trust factor limit of 1000 is to be preferred. This change factor was found to best differentiate between users, as well as producing an end result within one ES step from an ideal reputation system. The maximum trust factor of 1000 is set in order to prevent individual users from having too much impact.

We have made three improvements of our simulator compared to the previous study. First, we have implemented support for users leaving and entering the SRS during the simulation, i.e. using a non-static user base. This is what we call user change rate. In order to simulate this, we have designated a certain amount of users to be replaced every cycle in the simulation. However, to keep the conformity of the different user groups we have chosen to limit our simulation in the aspect that the number of users leaving the system is equivalent to the number of users entering the system. A subset of the users is randomly selected in the beginning of each simulation cycle, proportionally across the three different user groups. For the selected users, we simply reset their vote counters, i.e. the number of votes cast, and the trust factor, making the user object return to the state it was in when it was created. Of course, this will not affect any of the votes already cast by these users.

The second simulator improvement is that the users now choose which software objects to rate randomly according to a zipf-distribution with a skew of 0.9 [1]. As a result, 10% of the simulated software objects are more popular than the rest, i.e. are more popular and therefore also more likely to receive user ratings. The zipf-distribution was chosen because it has proven to be valid for popularity distribution in other computer science settings as well as various real world distributions [11], [4], [1].

The third improvement is the introduction of malicious users into the population, as we need to investigate how a SRS behaves when subjected to attacks. This is something that we have seen happen in several real world reputation systems [6], and as a result we need to investigate how a SRS behaves when subjected to attacks. The implementation of the malicious users is done as subgroups of the existing users groups. When voting maliciously, these users will vote as far from the software objects average rating as possible. The malicious users do not always vote maliciously, in an attempt to increase their reputation and impact within the SRS. That is why each malicious user is assigned a trigger chance, i.e. the probability of casting a malicious vote. However, all votes from malicious users target the same subset of software object.

It should also be noted that a user could be marked as both changeable and malicious, so some overlap will most likely occur. That means that some of the malicious users are also likely to be leaving and entering the system users, and as a result will be subject to their profile being reset.

5.4 Experiment Settings

The simulator we use is meant to reflect how software reputation system works, more importantly how one can with the aid of a reputation system decide if an application is to be considered good or bad. What we want to find out in this study is one of the problems with a collaborative rating system. What happens if a group of users votes bad out of ignorance, in which case the user can be considered a novice, or if a user votes bad out of malicious intent, in which case the user can be considered a malicious user. How do the system react when a portion of the user base vote maliciously, e.g. when an Orchestrated attack is carried out? In all simulations presented in this work we have a total of 10,000 software programs and 100,000 users, casting a total of 9,600,000 votes, with experts being the smallest group at 9.4%, average group at 27.1%, and the novice group at 63.5%, based on the results in the following survey [7].

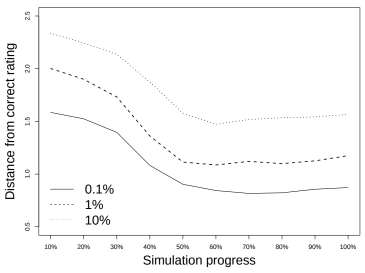
We use a user change rate of 1%, which means that for instance 94 expert users (1% of 9400) are replaced on average every cycle in the simulation. This results in an overall retention rate of approximately 37%, which corresponds to statistics from the early use of other social networks such as Facebook and Twitter[2].

5.4.1 Malicious Users

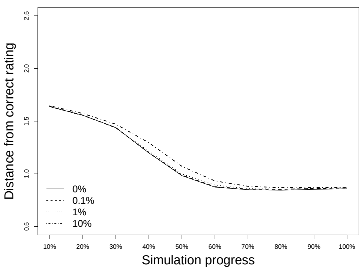
In order to estimate how well a software reputation system handles malicious users, we have chosen to simulate such scenarios. We simulate scenarios with the following three different constellations where 0.1%, 1% and 10% of the total number of users are being malicious, i.e. 100, 1000 and 10000 distinct users. All malicious users are distributed among the three user groups (expert, average and novice) based on their proportions.

Our attack scenarios can be said to mimic an orchestrated or slandering attack, i.e. a group of users intently using the same behavior in order to influence the system [8], as well as a Sybil attack. In this case the influence is of a malicious nature since the users wants to alter the ES for a subset of targeted software objects. In our scenarios 1% (or 100) of the total number of software objects are targeted by the attacks. These software objects are also chosen randomly in accordance with the popularity distribution. The malicious users in our scenarios are designed to be sleepers, i.e. they will appear to be normal users while building up their trust factor, and thus their influence, in the community before turning malicious. A user's trigger chance determines the possibility to cast a malicious vote, if too high the user will never gain enough influence in the SRS to affect its ratings, but on the other hand if too low there will simply not be enough malicious votes to affect the SRS. We therefore simulate four different scenarios that use the following trigger chances 25, 12.5, 6.25 and 3.125%.

[2]`http://blog.nielsen.com/nielsenwire/online_mobile/twitter-quitters-post-roadblock-to-long-term-growth/`



(a) Three different scenarios with varying number of malicious users targeting the same set of 100 software programs.



(b) Overall system accuracy based on four scenarios with different trigger chances and a population of 1% malicious users.

Figure 5.1: The impact of malicious users attacking the targeted list of software and the overall system.

5.5 Results

In the following section we present the results from our simulated scenarios with malicious users. We have found that the introduction of user change rate have a subtle effect on the system. Comparing the baseline of 1% user change rate with no change rate at all (that is a static user base) the results were found to be approximately the same.

5.5.1 Malicious Users

Figure 5.1a shows the results of malicious users attacking a small group of 100 programs. Every vote cast by the malicious users have a 12.5% chance of being malicious. The results for 1% malicious users show a deviation of approximately 2.0 ES from correct value in the beginning of the simulation. However, the ES quickly decreases and halfway, i.e. after 50% of the total number of votes has been cast, it has reached an ES of approximately 1.0 since the non-malicious users have built up their trust factor. Then, in the second half of the simulation there is a slight increase of ES as expert users begin to reach the maximum trust factor limit, i.e. their influence can not grow any further. As can be seen in Figure 5.1a there is a noticeable difference when comparing 1% malicious users with a population of instead 0.1%, which starts at an ES of 1.6 and reach 0.9 halfway through. However, in the latter scenario there is only a very subtle increase of the ES in the second half of the simulation.

When comparing the effect malicious users have on the overall system, as seen in Figure 5.1b, the malicious users do not have any noticeable affect on the overall SRS with a malicious population of 1.0% or less. For the scenario with a population of 10% malicious users there is at most only a slight difference of approximately 0.1 ES. However, even this minor difference vanish in the second half of the simulation as the increased impact of the regular users catch up.

In another scenario we investigate how different trigger chances of the malicious users affect the success of the attack. Figure 5.2 shows the ES for the 100 targeted software programs, when a population of 1% of malicious users make use of four different trigger chances. Our results show that even though there is a high chance of malicious voting taking place, the system is capable of suppressing the malicious votes, i.e. such user behavior stick out and is therefore possible to detect and maybe also address.

Our data also shows that we can detect the occurrence of such an attack by looking at the sum of the trust factors for any program and comparing them to other applications. The trust factor sum for an attacked program in such a scenario is, on average, quite a lot larger than for programs in general. In some cases up to 20-30 times larger. These tendencies can be seen early in the simulation progress, and is therefore a promising approach to detecting attacks.

Finally we made a comparison how the simulated reputation system using trust factors for all users would compare with a simpler reputa-

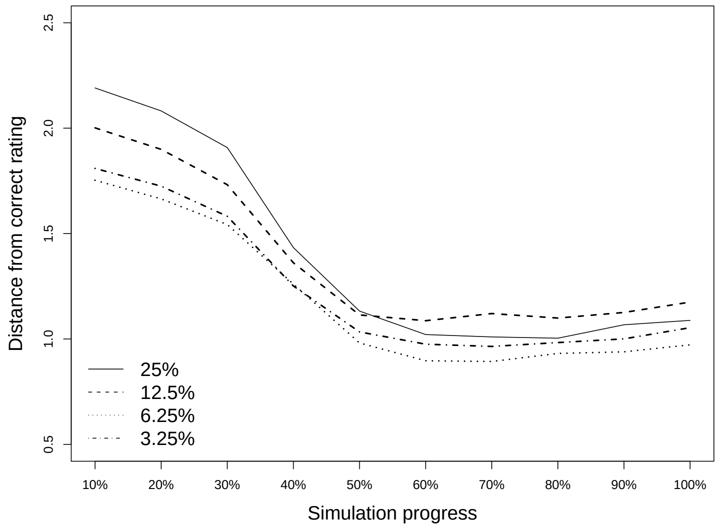


Figure 5.2: The effects of various chances of voting malicious with 1% malicious users targeting a subset of software.

tion system where all users are given the same impact, i.e. a system that calculates the software ratings as simple arithmetic mean based on all votes. In such a system the overall system accuracy would be 1.7 ES throughout the whole simulation, which could be compared with the SRS that stabilizes at about 1.0 ES as shown in Figure 5.1b. That is approximately a 70% improvement of accuracy once the SRS has stabilized. When only considering the attacked software programs, the ES are 1.7, 2.0 and 2.6 for 0.1, 1 and 10% malicious users respectively, i.e. also considerable improvements.

5.6 Discussion

The proposed simulated reputation system involves an environment of rated programs with varying popularity and with a differentiation of votes given by the users depending on their skills. More exactly we used a zipf-distribution and an exponentially increased trust factor with an upper bound of 1000 initial votes. Furthermore, a dynamic population of differently skilled users determine the accuracy within the reputation system. As a result, the trust factor allows an initially small amount of expert votes to later on outperform the votes from a far larger group consisting of for instance novice and malicious users. On the other hand, an expert leaving the reputation system will be replaced by another expert starting with an initial trust factor of 1.0, i.e. most probably a larger decrease of the trust factor than for an average or a novice user. The different parameters used in the simulation were either based on external scientific sources or by testing different variables, e.g. varying the trigger chance for malicious users.

Malicious users were introduced and by using a coordinated attack it is possible to substantially influence the rating of a selected subset of applications available in a SRS. The ES of the targeted software decreases in the middle of the simulation (between 30 and 60%) as seen in Figure 5.1a. This is explained by a proportional larger impact from malicious users in the beginning of the simulation (around 0-30%) in combination with continuously increasing trust factors for non-malicious users. If a deviation of ± 1 ES (or 10% on a 10 graded rating scale) in rating accuracy is regarded as acceptable, then the SRS can withstand around 1% malicious users focusing their combined attack on a small subset of all available programs. However, when instead looking at the overall system accuracy the SRS can withstand even more than 10% malicious users while still keeping an accuracy of ± 1 ES.

A SRS may be highly vulnerable to various attacks, which makes detection of attack attempts an important topic. An attack scenario with

1% malicious users attacking a limited number of 100 programs was also simulated. These malicious users adopt successful strategies in order to increase their trust factor before voting maliciously, i.e. improving their reputation by voting as ordinary novice, average or expert users at all other occasions. A favorable malicious strategy should get both a fast response and a fairly high evaluation score when voting on the attacked programs, i.e. limit the number of votes given in order to accumulate the trust factor. Looking at Figure 5.2 we see the result of four different strategies consisting of voting malicious on average 25% to 3.125% of given votes. These results show that the system stabilizes halfway into the simulation and in the second half there is only a slight increase in the overall ES. An attacker would probably consider the case with a trigger chance of 12.5% to be the "best" attack strategy due to the overall high ES. An important note here is that the users in an SRS does not have to know what trust factor they are currently holding, which would make it harder for malicious users to find the most suitable trigger chance for the attack.

Malicious users can be counteracted in two ways, either automatically based on the trust factors within the reputation system, or calculate the distance between the logged votes and the average rating of the program, where a long distance could indicate an attack. The final measure to protect the reputation system could be to temporarily freeze the programs that are under attack, i.e. not to register any new votes for these programs over some limited period of time. However, it should be noted that these indications not always point out an attack, and should therefore be used more as a warning system that requires for instance a moderator to look into the program in question. Given these circumstances, reputation systems can withstand severe collaborative attacks from a fairly high proportion of malicious users.

5.7 Conclusion and Future Work

A dynamic reputation system involving different skilled users was simulated. Malicious users were introduced and by using a coordinated attack it was possible to substantially influence the rating of a selected subset of the applications. The malicious strategies simulated included both scenarios that prioritized a fast response as well as an optimal damage to the attacked programs.

Since any reputation system is open for attacks from unscrupulous users it is important to implement measures to mitigate such effects. Our simulated reputation system that is based on individual trust factors shows an increased overall accuracy of approximately 70% compared to an ordinary reputation system using no trust factors at all, i.e. where every user have the same impact. In addition to this we also conclude two ways of detecting attack attempts. First by monitoring the trust factor sum for programs, where increased sums of up to 20-30 times the average could indicate an attack. Secondly, by measuring the distance between users votes and the average rating for software. Potential response measures could be to either silently reject ratings from suspected malicious users, or to temporarily freeze the programs that are under attack while manual investigation is conducted. Based on these measures it is our conclusion that a software reputation system can withstand severe collaborative attacks from a fairly high proportion of malicious users.

5.8 References

[1] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. Glottometrics 3, pages 143 - 150, Jun 2002.

[2] M. Boldt, A. Borg, and B. Carlsson. On the simulation of a software reputation system. ARES '10 International Conference on Availabil-

ity, Reliability, and Security, 2010, pages 333 - 340, 2010.

[3] M. Boldt, B. Carlsson, T. Larsson, and N.Linden. Preventing privacy-invasive software using collaborative reputation systems. Lecture Notes in Computer Science, Volume 4721, pages 142 - 157, 2007.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Jan 1999.

[5] N. Good, R. Dhamija, J. Grossklags, D. Thaw, S. Aronowitz, D. Mulligan, and J. Konstan. Stopping spyware at the gate: a user study of privacy, notice and spyware. Proceedings of the 2005 symposium on Usable privacy and security, page 52, 2005.

[6] K. Hoffman, D. Zage, and C. Nita-Rotaru. A survey of attack and defense techniques for reputation systems. ACM Computing Surveys, Jan 2008.

[7] J. Horrigan. Pew internet & americal life project: A typology of information and communication technology users. http://www.pewinternet.org/Reports/2007/ATypology-of-Information-and-Communication-Technology-Users.aspx, May 2007.

[8] A. J£sang and J. Golbeck. Challenges for robust of trust and reputation systems. 5th International Workshop on Security and Trust Management (STM 2009), Sep 2009.

[9] A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. Decision Support Systems, 43(2):618 - 644, Mar 2007.

[10] F. Marmol and G. Pérez. Security threats scenarios in trust and reputation models for distributed systems. Computers & Security, Jan 2009.

[11] J. Pitkow. Summary of www characterizations. World Wide Web, Jan 1999.

[12] P. Resnick, N. Iacovou, and M. Suchak. Grouplens: an open architecture for collaborative filtering of netnews. Proceedings of the 1994 ACM conference on Computer supported cooperative work, Jan 1994.

[13] P. Resnick and H. Varian. Recommender systems. Communications of the ACM, Jan 1997.

[14] J. Sipior, B. Ward, and G. Roselli. A united states perspective on the ethical and legal issues of spyware. ICEC'05, Jan 2005.

[15] K. Walsh and E. G. Sirer. walsh. 3rd Symposium on Networked Systems Design & Implementation, pages 1 - 14, Apr 2006.

Six

Informed Software Installation through License Agreement Categorization

Anton Borg, Martin Boldt and Niklas Lavesson

Abstract

Spyware detection can be achieved by using machine learning techniques that identify patterns in the End User License Agreements (EULAs) presented by application installers. However, solutions have required manual input from the user with varying degrees of accuracy. We have implemented an automatic prototype for extraction and classification and used it to generate a large data set of EULAs. This data set is used to compare four different machine learning algorithms when classifying EULAs. Furthermore, the effect of feature selection is investigated and for the top two algorithms, we investigate optimizing the performance using parameter tuning. Our conclusion is that feature selection and performance tuning are of limited use in this context, providing limited performance gains. However, both the Bagging and the Random Forest algorithms show promising results, with Bagging reaching an AUC measure of 0.997 and a False Negative Rate of 0.062. This shows the applicability of License Agreement Categorization for realizing informed software installation.

6.1 Introduction

This work addresses the problem of uninformed installation of spyware and focuses on analysing End User License Agreements (EULAs). Malicious software (malware) vendors often include (disguised) information about the malicious behavior in the EULAs to avoid legal consequences. It would therefore be beneficial for the user to get decision support when installing applications. A decision support tool that can give an indication whether an application can be considered spyware or not would presumably make the installation task simpler for regular users and would enable the user to be more secure when installing downloaded applications. We present an automated method that extracts and classifies EULAs and investigate the performance of this method. More concretely, the proposed method is based on the use of machine learning techniques to categorize previously unknown EULAs, as belonging to either the class of legitimate or malicious software. Machine learning, in this context, enables computer programs to learn relationships between patterns in input data (EULAs) and the class of output data (malicious or legitimate software). These relationships can be used to make classifications of new (unseen) EULAs.

6.1.1 Aim and Scope

The primary aim of this study is to present a method for automatic EULA extraction and classification. Additionally, we, using this method, obtain and prepare a large data set of EULAs. This data set is used for benchmarking four different algorithms. Evaluating the impact of feature selection and machine learning algorithm parameter tuning is also done[1].

[1]A web link to the actual database will be provided in a potential camera ready version

6.1.2 Outline

In Section 6.2 we present the background, definitions and related work. Section 6.3 describes an automated system for classifying EULAs as well as the steps needed for creating a database of classified EULAs. This database is then used for the experiments, which are defined in Section 6.4. The experimental results are presented in Section 6.5 and discussed further in Section 6.6. The paper is then concluded with some suggestions for future work in Section 6.7.

6.2 Background and Related Work

6.2.1 Background

Malware, e.g. viruses, originated from a rather small set of software, with the primary goal of generating revenues for the attacker, or creating chaos among infected computer systems [20]. To protect users from these types of software, anti-virus tools emerged. As the malware at this point were illegal, removing malware was a question of using the resources and techniques available at the time [21].

At the end of 1990, a new type of malware emerged, known as spyware, with the purpose of gathering personal information. Due to the increase of the number of Internet users, a market for targeted online advertisement developed. Spyware was not considered explicitly illegal, which complicated malware removal and resulted in the creation of a legal grey zone.

A common technique for detecting malware was to blacklist these applications through the use of signatures, i.e., by statically dividing between legitimate and malicious software. However, this required a copy of the malware to first be captured on the Internet in order to create a unique signature, and then being distributed to all customers of the anti-virus tool [21]. The main drawback with this technique is the fact that the anti-virus tools were one step behind the creators of malware. Another drawback is related to the vast amount of malware that spread on the Internet, increasing the size of the signature database rapidly and resulting in significantly decreased performance when used by customers.

Anti-virus manufacturers therefore began researching alternative techniques for solving the problem. For example, agent-based approaches [15] and artificial neural networks [1] [9] was investigated. Another technique used was dynamic analysis, which kept a suspicious program in captivity within a so-called sandbox, e.g. a virtual machine, while monitoring its execution as a way to discover any deviant behavior [8] [21]. Even though dynamic analysis could be used for computer viruses, e.g. by detecting the self-replication routines, it was much harder to distinguish spyware or adware applications from their legitimate counterparts. The reason is that adware and spyware applications simply show information on the screen or transmit rather small quantities of data over the network, i.e. behaviors that are shared by most legitimate programs as well.

6.2.2 Machine Learning

The machine learning discipline concerns the study of programs that learn from experience to improve the performance at solving tasks [14]. A large number of directions, methods, and concepts, which can be organized into learning paradigms. Usually, three paradigms can be distinguished; supervised learning, unsupervised learning, and reinforcement learning. The suitability of a certain learning method or paradigm depends largely on the type of available data for the problem at hand.

From a machine learning perspective, the main problem studied in this paper can be described as that of learning how to classify software applications on the basis of their associated EULAs by generalizing from known associations of EULAs and software application classifications [12].

More formally, and based on suitable definitions [13], we assume the existence of a universal set, I, of EULA instances. Each EULA instance, $i \in I$, is defined by a set of features (e.g., words, strings, values, and so on). Furthermore, we assume that the EULAs can be categoried into a limited number of categories or classes, C.

The learning task is then to generate a function (or mapping) between I and C. This function, $c : I \rightarrow C$, is known as a classifier or generalization. In practice, however, one does not have access to the complete set, I, or the correct classification of each element of that set. Instead, a common case is to have a limited set, $J \subset I$, of instances (inputs) and correct classifications (outputs).

Thus, the practical objective, is to generate a classifier by generalizing from J (or a subset of J) and the associated known classifications for each instance of J. Since we are interested in generating a classifier that will indeed be able to classify unknown instances (instances from I but which are not included in J), we need to estimate the theoretical classification performance on I by calculating the classification performance of J (or, again, a subset of J).

The common practice in data mining and machine learning is to divide J into two distinct sets; the training set, J_{train}, and the testing set, J_{test}. This way, a classifier can be generated from J_{train} and the prediction performance can be estimated by computing the classification performance on J_{test}.

There are many learning algorithms available that can perform the task of generating a classifier from input data associated with known outputs. However, each algorithm has its own learning bias, which is used to define the search space (the set of available classifiers) and the traversal of the search space. A completely unbiased learner would have access to the complete set of possible classifiers and would be able to traverse this set in any possible way. Of course, this search is practically infeasible in real-world situations. It is therefore necessary to select an algorithm, or a set of algorithms, whose learning biases are most suitable for the problem at hand.

6.2.3 Related work

Previous research on the use of text classification techniques within the context of EULAs is quite sparse. It have been shown that it is possible to use machine learning techniques to address the problem of EULA classification [12] [2]. State-of-the-art within commercial tools involve one stand-alone application called EULAlyzer[2] and one website[3] called EULA Analyzer that includes the ability to analyze a EULA. Both of these services are proprietary and therefore lack information regarding their design and internal construct. However, it seems as if they make use of blacklisted words to simply highlight any sentence within a EULA that contains one of the blacklisted words. The computer user then has to read through the highlighted sentences to try to come to a conclusion whether the particular EULA should be considered legitimate or spyware.

A comparison is made between EULA Analyzer and 15 machine learning algorithms [2], with the conclusion was that both the Support Vector Machines [6] and Naive Bayes Multinominal [10] algorithms performed significantly better than the state-of-the-art tool. Finally, it could also be added that the performance of these two algorithms have later been improved even more when utilized on an extended data set of EULAs [12]. However, the previous research conducted requires user interaction when gathering the EULA, which can be considered infeasible in a large-scale setting. Performance tuning have also been overlooked

[2]EULAlyzer, http://www.javacoolsoftware.com/eulalyzer.html

[3]License Analyzer, http://www.spywareguide.com/analyze/analyzer.php

in this context, and should be investigated as it has proven beneficial in other cases [19].

6.3 Approach

We have gathered 7,041 applications, where approximately 21% of the applications are malicious, from which we extract EULAs to form an extended data set. The EULAs were extracted using the automated tool described in Section 6.3.1. The malicious applications, counting 1,530 applications, have been provided by Lavasoft[4] and the legitimate applications, counting 5,511 applications, have been downloaded from CNET's download.com site[5]. Download.com thoroughly checks for malware among the applications made available to the public and can thus be said to be a good source of legitimate applications.[6] We also, in Section 6.4, test the data of the two sources to find differences.

In order to extract the licenses from the applications, we make use of the automated system presented in section 6.3.1. From the applications we managed to extract a number of license agreements for use in our experiments. As not all applications have licenses and our extractor does not support all file structure, as described in Section 6.3.1, this has left us with 810 malicious licenses and 1,961 legitimate licenses. For software with localized EULAs, only the English version were kept. These numbers means that our automatic tool is currently capable of extracting licenses from approximately 53% of the malicious applications and 35% of the benign applications. With more extractors implemented, this number is likely to increase. Many benign applications have similar licenses, but since minor details still vary and they help with the categorization, similar licenses are kept.

[4] Ad-Aware by Lavasoft, http://www.lavasoft.se

[5] CNET Download.com, http://download.com

[6] Download.com Software policy, http://www.cnet.com/download-software-policies/

6.3.1 Automated System

We have for this study developed a system for automated license classification, using a binary file as in-parameter and presenting the user with a classification of the binary file based on the EULA. Earlier research has used a manual process of extracting EULAs from the binary, which is infeasible in an real world setting. We have implemented an automated system using machine learning techniques for this purpose. In its current incarnation, it supports the standard installer types, e.g. NSIS, MSI, Inno setup, as well as standard archive formats, and makes use of publicly available programs as subsystems. Our proposed system is divided into three stages, extraction, transformation and classification. A flowchart of how the proposed system works can be seen in Figure 6.1 and pseudo code for the identification stage is shown in Algorithm 1. The system is implemented in Ruby[7] and in a way designed to make it easy to extend, thus adding support for more types of applications is fairly simple.

The system is based on the premise that a wide range of installers are roughly equivalent to a compressed archive. In order to know which extraction routine to use, the system identifies the binary file. To do this, the system makes use of the program TrId[8]. The system then tries to extract the EULA from the binary file. Depending on the result of the identification, this is done using different extraction routines. An example of this is the MSI installer. MSI installers store licenses in Rich Text Format(RTF) inside their string data. The system locates the RTF data inside the string data file and extract it. To perform the decompression of the binary file, we have built our system around the 7zip extractor[9]. 7zip supports a large number of filetypes and are thus suitable for our system.

The transformation stage is divided into two substages, conversion

[7]Ruby Programming Language, http://www.ruby-lang.org/en/

[8]Mark Pontello's Home, http://mark0.net/soft-trid-e.html

[9]7-Zip, http://www.7-zip.org/

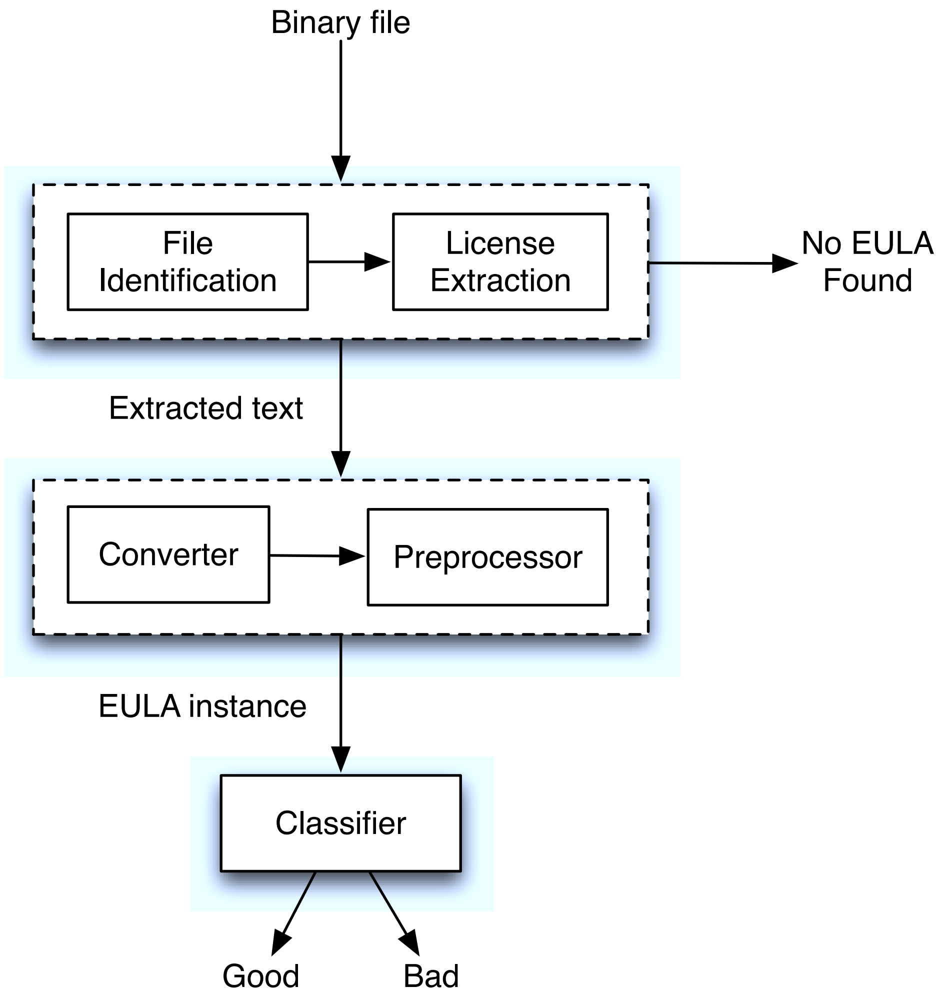


Figure 6.1: A conceptual view of EULA extraction and classification.

Algorithm 1 File Identification

```
function CHECK($path$)
    if $path$ is a file then
        $type$ = typeIdentifier($path$)
        if $type$ is an installer then
            Extractor($path$, $type$)*
        else if $type$ is a document then
            if $path$.name contains license or eula then
                SaveDocument($path$)
            end if
        else
            $extracted$ = Extractor($path$, default)
            if $extracted$ is not NULL then
                for all $file$ in $extracted$ do
                    check($file$)
                end for
            end if
        end if
    else if $path$ is a directory then
        for all $file$ in $path$ do
            check($file$)
        end for
    end if
end function
```

* The extractor chooses a suitable extraction routine based on the type. The extraction of an MSI installer is described in Section 6.3.1.

and preprocessing. In the conversion stage, the system converts the license to plain text. This is for the preprocessing to be able to read the license agreement, as the different installers store the license agreements in different formats, e.g. MSI uses the RTF format. The RTF licenses for example is converted to plain text files using the UnRTF[10] program, stripping everything but the text from these files. UnRTF is specifically designed to convert from RTF to other formats. The preprocessing substage is described thoroughly in section 6.3.2. The result of this stage is a EULA instance that the classifier can categorize.

The EULA instance is then passed to the classifier stage where it is categorized using machine learning algorithms. The result of the categorization is then presented to the user, helping the user to decide if the application is either good or bad, and whether or not to install the application.

6.3.2 Data Preprocessing

We conduct our experiment using the Weka machine learning workbench [22], a commonly applied suite of algorithms and evaluation methods. In order for Weka to be able to process the EULAs, we have opted to remove special characters and to only keep the standard latin characters. As few machine learning algorithms can process strings, we transform the strings to a more suitable representation. Ways for representing text include, e.g.: meta data (such as word length, frequency or the number of words) [11], bag-of-words (where each word in the text is defined as a feature) [17] and *n*-grams [5]. [17] also looks at phrase based features, where words would form a phrase which is able to better convey the meaning of the sentence, and Hypernym based features, where relationships between words is taken into account. The study found that the bag-of-words model outperformed the more complex text representation methods [17]. In the bag-of-words model, strings are tokenized to

[10]UnRTF, http://www.gnu.org/software/unrtf/unrtf.html

words and represented by word vectors. In Weka, this transformation is carried out using the *StringToWordVector* filter, which we apply to the licenses. We employ the following filter configuration: a maximum of $2,000$ words are stored per category, TF IDF (Term Frequency-Inverse Document Frequency) is used for word frequency calculation, and the Iterated Lovins stemmer is used to reduce the number of words by keeping only the stems of words.

Software licenses can contain a large amount of text and as a result, yields a large number of features. Many of these features are not useful to the learning algorithm and have, in some cases, even been shown to deteriorate the performance of the classifier [22]. We have therefore chosen to remove some of the attributes left by the StringToWordVector filter.

Feature Selection is the process of reducing the number of features available in a data set in order to increase either the classification and/or the computational performance [16]. This is done by, using an algorithm, removing features that are deemed unlikely to help in the classification process. It has been shown that classification accuracy have been improved when reducing the number of features using feature selection algorithms [23]. Several comparisons between feature selection algorithms applied on text categorization have been done in the past. The results is that χ^2 is often considered to be the most efficient algorithm [16] [23]. However, when compared to Categorical Proportional Difference(CPD), CPD have been shown to outperform traditional feature selection methods, e.g. χ^2 [19]. As a result we choose to use CPD as the feature selection algorithm of our choice. However, as we do not know which is the best cutoff point, i.e. how many attributes CPD should remove, we have defined a keep ratio interval and selected a step size. In the presented study, we use a keep ratio interval of 100% to 10% together with a step size of 10%. After applying CPD, we are left with 10 data sets, where the attributes range from 10% of the attributes kept to 100% of the attributes kept.

6.4 Experimental procedure

We want to determine whether the classification results obtained in previous research are valid for a larger data set. We also investigate if the classification performance can be increased using feature selection or by tuning problem-specific algorithm parameters.

Four algorithms have been chosen as a basis for our experiments. The algorithms are Bagging, Random Forest, Naive Bayes Multinomial and Support Vector Machine (SVM). The three first were selected on the basis of previous experimental results [12], and we chose to include only one algorithm from each family of algorithms. SVM was also included since it has been proved to work well in other text categorization tasks [18]. In both experiments, the performance is estimated by using the 10-fold cross-validation test. 10-fold cross validation is the process of dividing the data set into 10 subsets (folds), using 9 folds for training and 1 fold for testing. This is then repeated 10 times, switching the testing fold each time. Before running our experiments we executed preliminary experiments, which indicated that feature selection combined with parameter tuning do not yield any specific performance boost when used together. Therefore, we conduct separate experiments for feature selection and parameter tuning. Also, we made an attempt to validate that the learning algorithms included in our experiments indeed detect the differences between benign and spyware EULAs. Therefore, we divided the data set into two dummy classes that each included 50 % of the benign EULAs and 50 % of the spyware EULAs. Then we used the Naive Bayes Multinomial learning algorithm to generalize from these two dummy classes to make sure there were no patterns separating them, i.e. patterns included in both the randomly selected benign and spyware EULAs. This resulted in a AUC score of 0.526, which is very close to random guessing (AUC is explained in Section 6.4.3). Therefore the results indicate that there does not seem to be any other hidden patterns tying the classes together, and thus that our data set is valid for further exploration.

Table 6.1: Feature Selection

Feature Set Size[a]		Weighted AUC			
Relative	Absolute	SVM	Bagging	Random Forest	NB[b]
10%	140	0.929	0.932	0.941	0.802
20%	278	0.946	0.976	0.986	0.863
30%	417	0.964	0.989	0.995	0.891
40%	555	0.968	0.991	0.994	0.956
50%	693	0.968	0.992	0.996	0.973
60%	832	0.975	0.990	0.993	0.956
70%	970	0.978	0.992	0.997	0.936
80%	1,109	0.977	0.991	0.994	0.903
90%	1,247	0.977	0.995	0.995	0.896
100%	1,385	0.977	0.995	0.992	0.897

[a] The fraction of attributes to keep after CPD attribute ranking
[b] NaŢve Bayes Multinominal

6.4.1 Experiment 1: Feature Selection

In this experiment we investigate what effects feature selection have on the performance results of the four machine learning algorithms mentioned previously. In order to evaluate any potential performance gains we create ten new data sets that all are subsets of the initial data set containing 2,771 EULAs. This gives us data sets ranging from 10 to 100% of attributes kept, with a step size of 10%. The number of attributes for 10% and 100% are 140 respectively 1,385, as can be seen in Table 6.1.

6.4.2 Experiment 2: Parameter Tuning

In the second experiment we investigate if it is possible to increase the performance using parameter tuning of the two algorithms with the highest performance measure from the previous experiment. The two algorithms included in this experiment were Bagging and Random

Forests. We opted to select the top two performers rather than the top performer, since these two algorithms both showed fairly similar results. Moreover, the ways in which the two algorithms can be configured are quite different from each other. The variables that we use for parameter tuning is discussed below. However, it should be mentioned that both algorithms are ensemble algorithms, meaning that they each use several different learning algorithms that votes on the classification of each instance. It is then the task of the ensemble algorithms to reach a decision based on the results from the different learning algorithms used.

Random Forests

Random forest contains two main variables that we tune in order to determine if we are able to increase the performance. The first variable is the number of trees created in the forest. Each tree gets to vote for the instance, and the class with most votes is picked. Thus, the higher number of trees, the more votes are used as a basis for the classification. The second variable is the number of attributes used to build each tree. The number of attributes is a subset of all available attributes within the data set, and is chosen randomly for each tree. A higher number of attributes decreases the errors produced by the forest. However, it also makes each tree more similar [4]. For both these values we have chosen a symmetric range of values to use, based on the default values available in Weka.

The default value for the number of trees in Weka is 10. Based on this value, the range we have chosen is between 4 and 16 trees (inclusive) with a step interval of 2. The default value for the number of attributes is $\log^2(n) + 1$, where n is the number of attributes available in the data set. Working from this we have calculated our ranges of values for the number of attributes with $\log^2(n) \pm x$, where x is a range from -3 to 3 with a step size of 1. The number of attributes for the data set with 100% of the features left, seen in Table 6.1, is 1385 and based on this we get that the default value for our data set is 11.

Bagging

In Bagging we have chosen to investigate how tuning the bag size based on the training set used, as well as the number of iterations that the bagging algorithm performs affect the performance of the algorithm.

The first variable is the size of the bags from which the trees in the model is build. The sizes of these bags are set as a percentage of the training set size. The bags are filled randomly by sampling with the replacement from the original training set. This means that in each bag, there will be duplicates, as each instance in the training set can be selected more than once. The second variable is the number of iterations, which decides how many trees should be created within the current bag [3]. The vote result of each tree is then used as the result for the current bag.

The default value for the number of iterations is 10, and we have chosen to investigate the range 6 to 14 with a step interval of 2. The training sizes investigated is an five percent step ranging from 100%, which is the default value in Weka, to 75%.

6.4.3 Evaluation Metrics

We represent EULAs associated with spyware programs as positives, while benign EULAs are represented as negatives in our experiments. Used metrics are True Positive (TP), False Negative (FN), False Positive (FP) and True Negative (TN). A TP is a spyware instance classified as spyware and a FP is a benign instance classified as spyware. A TN is a benign instance classified as benign and a FN is a spyware instance classified as benign. The True Positive Rate (TPR) and False Negative Rate (FNR) are used to see how the spyware EULAs were classified. TPR is defined as $TP/(TP + FN)$ and FNR is defined as $FN/(TP + FN)$.

When evaluating the performance of the algorithms, we have chosen

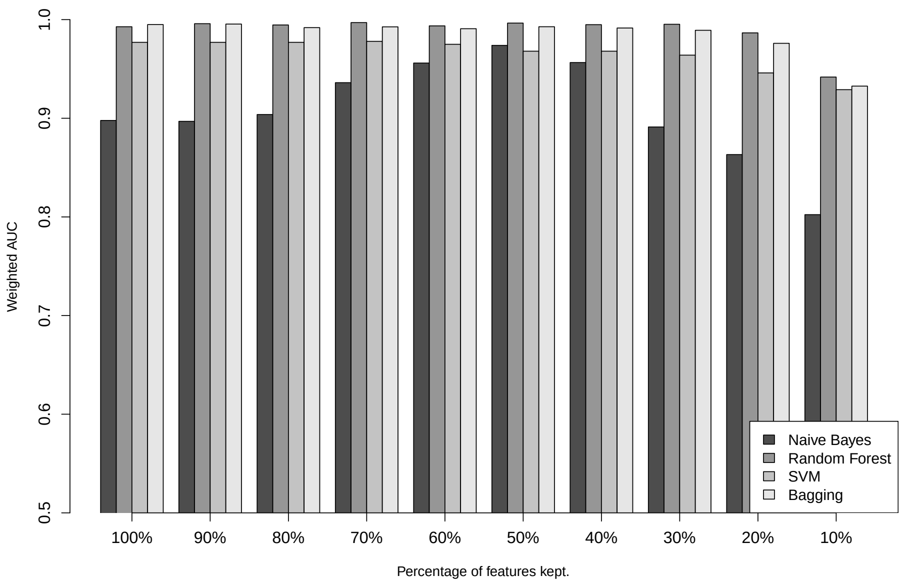


Figure 6.2: AUC for Random Forest, Bagging, Naive-bayes Multinomial and SVM on data sets with different amount of kept attributes.

to use the weighted area under the ROC curve (AUC) single point measure, which is based on TPR and the FPR. Two important properties of the AUC metric is that it is not depend on equal class distribution or misclassification costs [22]. The calculation of, and motivation for, AUC is described in detail in [7].

6.5 Results

6.5.1 Experiment 1

As can be seen in Figure 6.2, Random Forests and Bagging outperform Naive Bayes Multinomial. The latter performs best when only 50% of the attributes are kept.This suggests the use of feature selection when employing Naive Bayes Multinomial. However, Bagging and Random Forests still perform better than Naive Bayes Multinomial. SVM also performs well, but there is a clear loss of performance as the number of attributes is decreased, and the algorithm does not perform as well as Bagging and Random Forests, as can be seen in Table 6.1.

Bagging and Random Forests perform fairly similarly as long as the percentage of kept attributes is 30% or higher. This shows that it is possible to remove a fairly large amount of the attributes before starting performance is degradated. However, since the feature selection does not provide any performance enhancement, in fact there seem to be a small performance loss in most cases, there is no obvious argument for applying feature selection during pre-processing.

6.5.2 Experiment 2

The second experiment concerned the impact of parameter tuning on Random Forests and Bagging. In the following two subsections, we present the results of our experiments.

Random Forest

Table 6.2a and 6.2b shows the result of the parameter tuning done on the algorithms. Looking at Table 6.2a we can see that the performance of the algorithm correlates to the number of trees used in the forest. However, although an increased number of trees, the actual performance gain, when measured in AUC, is minimal. In Table 6.2b, showing the effects of using different amounts of attributes used when creating each tree, we see that for each number of attributes the results vary. These results do not seem to be related in any way, indicating that one cannot argue that tuning this variable is beneficiary to the overall performance.

Bagging

The effects of tuning the Bagging algorithm is presented in Table 6.3a and 6.3b. The results in Table 6.3a indicates that it is possible to lower the bag size, up till a certain point, and gain performance. However, the results are contradictory and should not be taken as a certainty. However, compared to Random Forest, the TPR and FNR values are worse.

Table 6.3b shows us that increasing the number of iterations do in fact produce a better result, going from 0.995 to 0.996, concerning AUC. The TPR and FNR values indicate that an increased number of iterations provide a minimal performance gain. The TPR in Table 6.3a indicates that decreasing the bag size can result in decreased performance.

6.6 Discussion

We have in this work presented an automated tool that (based on the EULA) decides if an application is considered malicious or benign. The use of this tool would help the user to decide whether or not an application can be considered malicious. In order to make this tool work we had to implement a number of extractors capable of extract the EULA for the program. As there exists several different installer formats and packers, we have in this study focused on the ones prevalent in our data set. However, we have built our tool in such a way that it is quite easy

Table 6.2: Results for Experiment 2: Random Forest

(a) Results of tuning the number of trees in Random Forest algorithm

Trees	Weighted AUC mean (STD)	FNR mean (STD)	TPR mean (STD)
4	0.993(0.006)	0.019(0.019)	0.980(0.016)
6	0.993(0.006)	0.021(0.017)	0.978(0.016)
8	0.994(0.006)	0.021(0.017)	0.980(0.015)
10 *	0.994(0.006)	0.022(0.018)	0.979(0.015)
12	0.994(0.006)	0.022(0.018)	0.979(0.015)
14	0.995(0.006)	0.022(0.018)	0.979(0.015)
16	0.995(0.006)	0.022(0.018)	0.979(0.015)

(b) Results of tuning the number of attributes in Random Forest algorithm

Attributes	Weighted AUC mean (STD)	FNR mean (STD)	TPR mean (STD)
8	0.994(0.006)	0.020(0.016)	0.980(0.016)
9	0.995(0.007)	0.021(0.015)	0.979(0.015)
10	0.994(0.006)	0.021(0.015)	0.979(0.014)
11*	0.994(0.006)	0.021(0.015)	0.979(0.015)
12	0.994(0.005)	0.021(0.015)	0.979(0.015)
13	0.995(0.005)	0.020(0.015)	0.980(0.015)
14	0.995(0.005)	0.021(0.015)	0.979(0.015)

Default value is marked with an asterisk.
All other parameters are left as per default.

Table 6.3: Results for Experiment 2: Bagging

(a) Results of tuning the bag size in Bagging algorithm

Bag size	Weighted AUC mean (STD)	FNR mean (STD)	TPR mean (STD)
100% *	0.995(0.005)	0.061(0.017)	0.942(0.023)
95%	0.997(0.003)	0.062(0.018)	0.942(0.023)
90%	0.996(0.004)	0.061(0.022)	0.938(0.022)
85%	0.996(0.004)	0.068(0.024)	0.934(0.031)
80%	0.994(0.005)	0.062(0.023)	0.935(0.025)
75%	0.995(0.005)	0.074(0.010)	0.928(0.021)

(b) Results of tuning the number of iterations in Bagging algorithm

Iterations	Weighted AUC mean (STD)	FNR mean (STD)	TPR mean (STD)
6	0.994(0.005)	0.063(0.015)	0.944(0.019)
8	0.994(0.005)	0.062(0.018)	0.943(0.019)
10*	0.995(0.005)	0.061(0.017)	0.942(0.023)
12	0.995(0.004)	0.057(0.020)	0.944(0.024)
14	0.996(0.004)	0.057(0.020)	0.944(0.024)

Default value is marked with an asterisk.
All other parameters are left as per default.

to extend it with more formats. It is also written in a way that makes it possible to use it for bulk tasks (e.g. in a bash script) or calling it from a program (e.g. an antivirus tool), the program is flexible and possible to use in both a server environment and desktop environment. Earlier research has shown the feasibility of using machine learning techniques to perform text classification on EULAs to distinguish between malicious and benign applications. However, earlier tools have required manual input when extracting EULA and submitting it for classification. One of the main contributions of this paper is the presentation of an automated tool that is able to extract EULAs, classify them, and then give decision support. The level of performance reached by the algorithms clearly shows the potential of an automated system for EULA classification.

6.6.1 Data Set Content

This work involves a significantly extended data set of 2,771 EULAs compared to the two previous studies that contained 100 and 996 instances, respectively. By using this data set of increased size it is our intention to more accurately mimic a real world setting. During the work of extracting the EULA texts from both the collected spyware and legitimate programs we could see a clear trend that malicious applications to a higher extent contained EULAs compared to benign applications. In this case 53 % of the malicious programs contained EULAs compared to 36 % of the benign programs, which strengthens the claim that developers of malicious programs make use of EULAs as a means to avoid legal consequences.

Even though this is a large data set, gathering it revealed that of the 7,041 applications gathered, we were only able to extract 2,771 EULAs, i.e. around 39%. This can mostly be attributed to the number of extractors implemented and can thus be corrected by implementing more extractors. However, another contributing factor could be that some applications will not contain a EULA, but as the goal of this is detecting malicious software that uses EULA, this can be considered acceptable.

6.6.2 Proposed System Vulnerabilities

Our proposed automated system classifies EULAs as belonging to either malicious or benign programs, and then presents the results to the user. If the classified EULA is previously unknown by the system it could be considered that the system ask the user if the EULA content, together with meta-data about the associated binary program, could be collected. With this information it is then possible to automatically retrain the classifier with the new and slightly extended data set, i.e. using online learning. The alternative approach is offline learning, which involves collecting previously unseen EULAs by other means, and thereafter manually regenerating a new classifier at certain time-intervals, e.g. once a month. Regardless which method is being used the classifier would still automatically detect any attempt by the developers of malicious programs to fool the system by reformulating the content in their EULAs. The reason for this is that they always need to express their software's behavior in the text, and by doing so they distinguish their EULA content from the benign EULAs.

However, it could be argued that the overall classification performance probably would be slightly higher if an online learning approach is used instead of an offline, since the classifier would be continuously re-trained using new EULAs. Of course, a prerequisite for the online learning approach is that the integrity of the new EULAs and the associated meta-data can be guaranteed. Otherwise, it could be possible for external parties to feed the learning component with false EULA content and meta-data in an attempt to reach a certain conclusion for a specific EULA. The fundamental problem in such a scenario is whether the input from the clients to the server really can be trusted to be untampered with. Using cryptographic techniques it could be set up so that the data could not be modified in transit, but unfortunately it is harder to protect the client software from any unauthorized tampering.

6.6.3 Experimental Results

Our results show that both Bagging and the Random Forest algorithms handles the EULA categorization problem well, which is surprising as SVM often is considered the algorithm most suited for text categorization problems. Both Random Forest and Bagging outperforms SVM as can be seen in Figure 6.2. Naive Bayes Multinomial was also out performed since it showed quite poor performance results except when around 50 % of the features were reduced.

As shown in Figure 6.2, both Bagging and Random Forest show equivalent results when their default configurations were used, with a slight peak when 10 % of the features were reduced using CPD. Parameter tuning for both algorithms showed only slight alterations in performance. The results also indicate that configuring the Random Forest algorithm using 14 trees and 14 attributes present the best result within the context of EULA classification. For the Bagging algorithm the use of a bag size of 95 % and 14 iterations result in best performance. The Bagging algorithm reached the highest performance with an AUC measure of 0.997 and a standard deviation of 0.03, together with a low FNR of 0.062. The latter is important as trust in the proposed system otherwise could be lost if the proposed system suggests to the user that he/she should install a malicious application. From a user perspective it is less critical if the system suggest that the user shouldn't install a legitimate program. It should be noted, that when novel instances, from outside our data set, are applied to the classifier, a certain performance degradation is expected.

6.7 Conclusion and future work

We have implemented and presented an automated tool for classification of binaries, based on the bundled EULA. We have created an algorithm, also presented in the paper, which handles the extraction. As the number of malicious programs increases, the presented system could assist users in separating between malicious and benign programs based on their EULA.

Furthermore, we have, as a result of using our automated tool created a dataset consisting of 2,771 EULAs. To the best of our knowledge, this is the largest collection of labeled EULAs available today. Using this dataset, we investigate the performance of four different learning algorithms, strongly suggesting the suitability of using Bagging and Random Forest to classify EULAs. The compilation and use of this extended dataset compared to previous datasets used in our prior experiments is a major contribution in this paper.

Using this dataset, we have investigated whether or not performance tuning of the learning algorithms provide better results than the standard settings. The results makes us conclude that the use of performance tuning is of limited use for the problem at hand. Similarly, we investigated the impact of using the feature selection algorithm CPD, which in other settings have proven very effective for increasing the prediction of the learning algorithm. However, we've found that, excluding Naive Bayes Multinomial, prediction is of almost no difference or even worse. In the case of Naive Bayes Multinomial, CPD increased the prediction, but otherwise performed worse than the other algorithms evaluated.

For future work we plan to carry out experiments where computer users evaluate the use and benefit of a fully automated decision support tool when installing software. We will also investigate the occurrence of EULAs in a real world setting.

6.8 References

[1] W. Arnold and G. Tesauro. Automatically generated win32 heuristic virus detection. In *Proceedings of the 10th International Virus Bulletin Conference*, Orlando, USA, September 2000.

[2] M. Boldt, A. Jacobsson, N. Lavesson, and P. Davidsson. Automated spyware detection using end user license agreements. In *Proceedings of the Second International Conference on Information Security and Assurance*, Busan, Korea, April 2008.

[3] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, Jan 1996.

[4] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, Jan 2001.

[5] W. Cavnar and J. Trenkle. N-gram-based text categorization. In *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval*, Hong Kong, China, Jan 1994.

[6] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, West Nyack, NY, 2000.

[7] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, Jan 2006.

[8] G. Jacob, H. Debar, and E. Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4(3), 2007.

[9] J. O. Kephart. Biologically inspired defenses against computer viruses. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, MontrÃĹal, Canada, January 1995.

[10] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes. Multinomial naive bayes for text categorization revisited. In *Proceedings of the 17th Australian joint conference on artificial intelligence*, Cairns, Australia, December 2004.

[11] L. Larkey. Automatic essay grading using text categorization techniques. In *Proceedings of the 21st International Conference on Research and Development in Information Retrieval*, Melbourne, Australia, Jan 1998.

[12] N. Lavesson, M. Boldt, P. Davidsson, and A. Jacobsson. Learning to detect spyware using end user license agreements. *Knowledge and Information Systems*, 2(3):285–307, 2011.

[13] N. Lavesson and P. Davidsson. Evaluating learning algorithms and classifiers. *Intelligent Information & Database Systems*, 1(1):37–52, 2007.

[14] T. M. Mitchell. Machine learning. page 414, Jan 1997.

[15] T. Okamoto and Y. Ishida. A distributed approach to computer virus detection and neutralization by autonomous heterogeneous agents. In *Proceedings of the 4th International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, March 1999.

[16] M. Rogati and Y. Yang. High-performing feature selection for text classification. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, McLean, USA, Jan 2002.

[17] S. Scott and S. Matwin. Feature engineering for text classification. In *Proceedings of the Sixteenth International Conference on Machine Learning*, Seattle, USA, Jan 1999.

[18] F. Sebastiani. Classification of text, automatic. In *The Encyclopedia of Language and Linguistics*, pages 457–463. Elsevier Science Publishers, 2006.

[19] M. Simeon and R. Hilderman. Categorical proportional difference: A feature selection method for text categorization. In *Proceedings of the Seventh Australasian Data Mining Conference*, Glenelg, Australia, Jan 2008.

[20] E. Skoudis. *Malware - Fighting Malicious Code*. Prentice Hall PTR, Upper Saddle River NJ, 2004.

[21] P. Szor. *The Art of Computer Virus Research and Defence*. Pearson Education, Upper Saddle River NJ, 2005.

[22] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, San Francisco, USA, 2005.

[23] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, San Francisco, USA, Jan 1997.

Seven

Social Network-based E-mail Classification

Anton Borg, Niklas Lavesson

Abstract

A majority of E-mail is suspected to be spam. Traditional spam detection fails to differentiate between user needs and evolving social relationships. Online Social Networks (OSNs) contain more and more social information, contributed by users. OSN information may be used to improve spam detection. This paper presents a method that can use several social networks for detecting spam and a set of metrics for representing OSN data. The paper investigates the impact of using social network data extracted from an E-mail corpus to improve spam detection. The social data model is compared to traditional spam data models by generating and evaluating classifiers from both model types. The results show that accurate spam detectors can be generated from the low-dimensional social data model alone, however, spam detectors generated from combinations of the traditional and social models were more accurate than the detectors generated from either model in isolation.

7.1 Introduction

The occurrence of spam has grown rapidly and it has been suggested that the majority of all E-mails are spam [1]. This development has resulted in the widespread use of spam filters, a use which can also be attributed to the inability of the current legislation to make an impact [2]. The legal inability has mainly been due to the differences in jurisdiction of various countries. Since most spammers only stay online for a limited amount of time it is considered hard to enforce the legislations [3], which increase the importance of automatic spam detection techniques.

This paper presents a method for E-mail spam detection that uses social information. This Online Social Network (OSN) supported spam detection method is compared with traditional spam detection. The paper also contributes with three metrics that have been adapted for social network data.

7.1.1 Aim and Scope

The aim is to investigate a method for spam classification using multiple OSN supported decision models. This paper implements a detction method based on using data from one OSN and compares it with a traditional spam detection method. The scope is limited to the study of social relationships mined from a public E-mail corpus.

7.1.2 Outline

In Section 7.3 research done in behavioral spam detection and in extension, OSN supported spam detection, is discussed. Section 7.4 presents a method for E-mail spam detection using social information. Section 7.5 details the method used for OSN data extraction, as well as the OSN data metrics. Section 7.6 outlines the experimental procedure. The results are presented in Section 7.7 and discussed in Section 7.8. Finally, conclusions and future work is presented in Section 7.9.

7.2 Background

Internet users today use a number of media to share information. Communication media comprise SMS, MMS, OSNs, E-mail, and instant messaging services. It has been stated that in 2010, around 250 billion E-mails were sent each day[1]. As much as about 90% of the E-mails sent are suspected to be spam[1]. E-mail is used along with OSNs as the two main forms of communication today. Large OSNs can attract around 100 million users, with Facebook surpassing 500 million. People use OSNs to exchange messages, media and information concerning social activities. Users of these services suffer from information overload as a result of the amount of data presented to them.

E-mail and OSNs are heavily used, but the data sources are rarely linked today, and thus are unable to improve through an exchange of information. Some work is focused towards this area, but are still in the initial phase. An example being that a medium can use information provided by a second medium, e.g. OSNs, to combat the problem of information overload in another, e.g. E-mail.

E-mail overload can be considered a specific form of information overload, a user receives more E-mails than he can process. Woods et al [4] have found that people tend to characterize information overload in three different ways. These three ways are listed below, with descriptions of how they apply to the problem of E-mail overload.

Clutter is when there is too much information on the screen. A proposed solution is to remove data available, Woods et al argues that

[1]How many emails are sent everyday, `http://email.about.com/od/emailtrivia/f/emails_per_day.htm`, 2012-02-26

the removing agent still have to know which data to remove, making this solution less ideal [4].

Workload Bottleneck occurs when a user is unable to properly deal with all the messages available within a timespan. Solutions are to have systems that summarize or prioritize the messages.

Significance of data concerns how to recognize which E-mails are important in a certain context. Some suggested solutions to this is, e.g. cognitive buoyancy, i.e. relevant information floating to the top or message constellations, i.e. how a set of message relates to each other [5].

One attempt at addressing E-mail overload, is the improvement of spam detection.

7.3 Related Work

A number of reviews on the existing anti-spam techniques jointly conclude that automatic techniques are necessary to implement spam filtering [6] [7].

Some approaches based on the use of ontologies to classify E-mails based on content and previous messages, have aimed at generating personalized classifiers [8] [9]. Over time users will have gathered large amounts of E-mails. By constructing a profile based on E-mail habits, it is possible to detect outliers, i.e. spam [10]. Other research have investigated profiling a user's E-mail sending behaviors using histograms to detect outliers [11] [12].

What can be inferred from previous work is a tendency towards using data from OSNs as a basis for anti-spam techniques. Researchers have previously investigated the use of OSN-based techniques for E-mail classification by using previous E-mail conversations to create a correspondence graph, and from that graph, creating a model for classification [13] [14]. Most of the research so far has focused on building social networks from data, e.g. graph analysis, instead of gathering data from OSNs. E-mail OSN analysis has been used to detect suspected accounts in a internal network by finding accounts that should have been removed but were still used [15].

Learning algorithms have been investigated for prioritizing messages by building OSN from previous E-mail conversations [13]. The data represent messages submitted by volunteers. The result soy the study show s the feasibility of the approach. Two caveats with the study are that the data sets can not be considered representative and the training of the model is irregular. The problem of representation occurs as the voluntarily submitted messages have been screen and selected by submitter. The second problem is that the training is done on the same amount of messages, regardless of the size of the data set. The amount used in the training set is the least common denominator for the data sets, i.e. a data set with a size greater than 1,000 instances will still use the same size of the training set as a data set with a size of 200. No practical reasons for this are mentioned.

Tran et al. have conducted research on providing a social context to E-mail correspondence [16]. A system that calculates the trust of the social path and also visualizes the path, have been implemented. This system provides an trust estimate between the corresponding parties. The data are based on social relationships from OSNs, in this case Facebook. OSN-based techniques can be used to enable the creation of a personalized spam filter and also allows the prioritization of messages, something which have been initially investigated [17].

By using the methods used to mine E-mail-based OSN and instead use other OSN sources as the basis of the classification, it is possible to address the problem of having a large E-mail based history, thus enabling extended classification for new users as well, given that said information is available on other OSN. Using OSN data sources as a

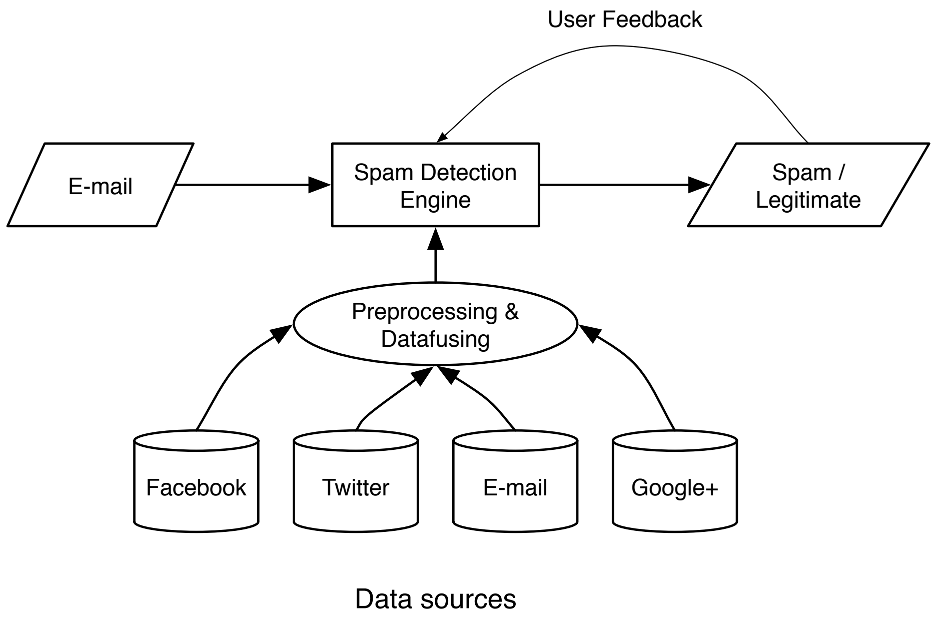


Figure 7.1: The proposed method uses Social Network data sources in order to personalize and improve the classification of incoming messages.

complement to mining OSN data from E-mail corpora, removes the requirement of users having a large E-mail corpus to mine from, to be able to create the graphs required. However, whether this approach is capable of detecting spam messages and which data is necessary needs to be investigated further.

7.4 Theoretical Model

Workload Bottleneck and *Significance of data* can be considered to be closely related. By solving *Significance of data* the likelihood of *Workload Bottle-*

neck can be reduced. By using automatic tools to determine cognitive buoyancy and message constellations, the E-mail overload can be reduced by classifying and prioritizing the messages. However, the question of which data to use as a basis for making these decisions is relevant. As users, in various contexts, use E-mails for different reasons, each user has to create personalized, context-aware classifiers. A classifier is an application that assigns labels to, in this problem domain, an E-mail, e.g. spam or ham.

The personalized and context-aware classifier uses, as a basis for its decision, several data sources that can be linked to the user. By using data available from different data sources, a classifier is able to interpret content and header information in a message and compare it with how a user communicates using the various datasources.

7.4.1 Data Sources

In this section a method is proposed that is capable of leveraging information from one medium of communication against a message received on another medium. As such the method needs to be able to gather information from several different data sources. These sources can be various services that a user has been linked to, e.g. various OSNs or E-mail history. The use of multiple data sources forms a classification basis that can be considered more personalized. For example, the content of the E-mail could be matched against a user's profile information or against the corresponding party's profile information, as well as earlier messages exchanged via the OSN. E-mail header information could be used to check whether a connection exists to a certain company or person via OSNs.

7.4.2 Context-driven Classification

The purpose and nature of social networks may vary. Some are used as a way of communicating short messages, some as a way of keeping in touch with friends, some for professional relationships. As a result it is possible to use these social networks to distinguish between contexts. If context is taken into account, the importance can be estimated based on where the user are, what time it is and/or a specified user mode (e.g. work mode).

7.4.3 Knowledge-based Classification

Another aspect that can be taken into consideration, is the level of knowledge of the contacts. By using OSN data, one can extrapolate, using e.g. work information or group memberships, a users knowledge areas. Given such an approach, messages could be tagged as more relevant or less relevant depending on the perceived knowledge held by the author. One field of application where this aspect is useful to consider would be E-mail conversations involving multiple correspondents where the user want the most interesting reply in the thread to be the first read, for example replies in an list discussion.

7.4.4 Automatic E-mail Classification

E-mail classification can be done automatically. Let I represent a set of E-mails represented as feature vectors. Each E-mail can be transformed into a vector of word frequencies. Let $C = \{spam, ham\}$ represent possible classifications. Given a set of examples, represented as pairs $E = \{< i, c >|\ i \in I, c \in C\}$, it is possible to generate an approximation, $\hat{f}$ of $f : I \to C$ using a supervised learning algorithm that generalizes from E.

Let T be a similar example set, $T \cap E$. It is now possible to estimate the performance on f by evaluating the performance on T.

7.5 Method

The method is based on the idea of using several data sources as input to an engine that classifies a message as either spam or ham. These data sources could comprise pieces of information from several social media. Given data from these data sources, the engine creates a graph of users and extracts the social information. This social information is then used as a basis for the classification of incoming messages, regardless of which medium is used to transfer the message. The proposed method can be seen in Fig. 7.1.

This paper focuses on one particular data source. The proposed engine uses a supervised learning algorithm to generate spam detection models from both E-mail content, header data, and, social information. Using a model that is extended with OSN data that can help in determining the behavior of a user (for example, the relationship between the sender and receiver) can be regarded as a OSN-based model.

7.5.1 Social Data Generation

As no public E-mail corpora explicitly include social meta data, e.g., the explicit relationship between the sender and receiver, models are generated from existing E-mail headers. Thus, even if there is a lack of explicit OSN attributes in the data, it is possible to extrapolate certain social information from the data set.

The motivation for extracting OSN data from the E-mail corpus instead of using OSN data as a data source, is that previous research on OSN based classification has used private data sets which have been altered in undisclosed ways. A public data set that has been peer-reviewed have been chosen for use. As such, it is hard to link users in the data set to OSN profiles and extract OSN data, requiring the social information to be mined from the E-mail corpus.

7.5.2 Social Data Metrics

In order to add social information to the data set, data from the corpus is mined and social information is constructed. This paper focuses on three social attributes; the number of messages exchanged between users, the number of common contacts, and the number of participants. These metrics have been adapted from available OSN metrics. The number of exchanged messages indicates whether two users can be considered friends. Equation 7.1 describes the process of calculating the message-exchange score (MES) for a set of users associated with a message.

$$\mathrm{MES}(m) = \frac{\sum_{i=1}^{t_n} M_{s,t_i} + M_{t_i,s}}{t_n} \quad . \tag{7.1}$$

For a given message, m, which contains a sender (s) and a set of receiving users (t), the number of messages to and from each user (t_i) and s is counted and an average for the number of receiving users (t_n) is calculated. In Equation 7.1, M is a matrix containing the number of messages between users.

Common Contacts Score (CCS) groups users, see Equation 7.2 It is calculated by listing the counting the users that t_x to t_y both have exchanged bidirectional messages.

$$\mathrm{CCS}(t_x) = \frac{\sum_{i=1}^{t_n} A \bigcap B}{t_n} \quad . \tag{7.2}$$

For a given user, t_x, let set A contain the users that t_x have exchanged messages with, given that said exchange of messages is two-way. Let the same be true for set B for user t_i. The sum of the intersection between A

and B is used as the CCS. If this is done for several users (t_i), summarize the score and divide it by the number of participants (t_n) to get a mean.

The number of participants is equivalent to $t_n + 1$, as it includes the sender as well. These three attributes are added as separate attributes to the combined data set.

7.6 Experiments

The aim is to evaluate the classification performance impact of social data in comparison to traditional content-based spam detection. For this purpose, a public E-mail corpus is used to generate a social, combined and traditional data set. The social data set contains only the social information extracted from the E-mail corpus. The social extended set contains the message body, the available E-mail headers as well as the social information. The traditional data set contains only the headers and the message body. The Weka machine learning workbench [18] version 3.7.0 is used as the software platform for conducting the experiment.

7.6.1 Data Collection

The selected corpus is the TREC 2007 Public Corpus(Trec07)[2]. The Trec07 corpus was selected on the basis of the size as well as the feature set. Compared to other public domain corpora, such as the Enron Spam Corpus[3], Trec07 contains header data in addition to the content data. The Trec07 was collected in 2007 and the corpus consist of 25,220 ham and 50,199 spam.

[2]TREC 2007 Public Corpus, `http://plg1.cs.uwaterloo.ca/cgi-bin/cgiwrap/gvcormac/foo07`, 2012-02-26

[3]Enron Spam Corpus, `http://www.aueb.gr/users/ion/data/enron-spam/`, 2012-02-26

Table 7.1: Attributes extracted from the Trec07 corpus

Attribute	Description	Type
Category	Classification: spam or ham	Nominal
To	Recipients	String
From	Sender	String
MES	Message-Exchange Score*	Numeric
NP	Number of participants*	Numeric
CCS	Common Contact Score*	Numeric
Received from	E-mail route description	String
Other headers	The remaining headers	String
Content	E-mail body, including any attachments	String

*Attributes only available in the Social and the Combined Data set.

7.6.2 Data Preprocessing

On a conceptual level, the data extracted from Trec07 can be divided into the list of attributes displayed in Table 7.1. However, few supervised learning algorithms can process strings. Thus, the string attributes must be transformed to a suitable representation. A common data model for this purpose, which has proven to be at least as effective as more complex solutions, is the bag-of-words model. In this model, strings are tokenized to words and represented by word vectors. In the first step, all special characters are removed, i.e., only the plus and minus signs, comma symbol, colon symbol, full stop symbol, the white space characters, alphabetical and numeric characters, as well as the @ sign are kept. The special characters preserved, are required to tokenize the E-mail header data and the whitespace is used to tokenize text strings into words.

In Weka, this transformation is carried out with the *StringToWordVector* filter, which is applied to the *To, From, Received from, Other headers* and *Content* attributes. The following filter configuration is employed: a maximum of $2,000$ words are stored per category, term frequency-inverse document frequency (TF-IDF) is used for word frequency calculation, and the Iterated Lovins stemmer is used to reduce the number of words by keeping only the stems of words.

Artifacts in the data set among the attributes have been identified and removed. The X-headers are not included in the data set as X-headers can be considered artifact attributes. The artifact attributes have been added by software and have low predictive power.

A stratified sample of 10% of the original instances is then generated, which leaves 7,541 instances. This is a size chosen to ensure a representative and large enough sample size, while maintaining a reasonable trade-off between computational effort and generalizability of the results.

7.6.3 Feature Selection

Categorical proportional difference (CPD), has been shown to outperform traditional feature selection methods, such as χ^2, information gain, and odds ratio on several text categorization corpora [19]. Thus, CPD seems to be a suitable feature selection technique for the task at hand. The search for a suitable cutoff point for CPD is computationally expensive due to the possible non-linearity of the function of the number of kept words and the resulting performance [19]. A keep ratio interval is therefore defined and selected with a reasonable step size. In the presented study, a keep ratio interval of 1.0 to 0.5 together with a step size of 0.1 is used. This configuration yields 5 iterations for each data set, which lets the possible performance gain for each data set be determined, by keeping from 50% to 100% of the attributes.

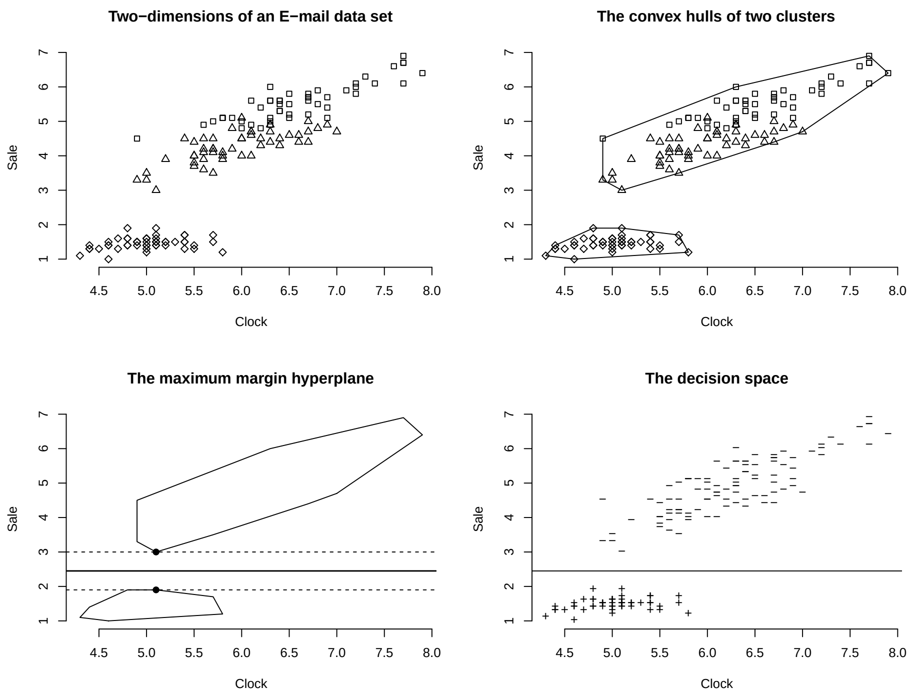


Figure 7.2: Example of SVM training process.

7.6.4 Algorithm Selection

The main objective is to compare different data models available for detecting spam, hence the comparison of multiple learning algorithms to determine the optimal algorithm is out of scope. The Support Vector Machine (SVM) is a reasonable candidate, since it has been shown to work well with similar data models [20].

Given a set of examples, E, a SVM model, $\hat{f}$, is generated by mapping each example, $e \in E$, as a point on a plane [21]. The SVM model uses a kernel function for mapping the examples, enabling the instances to be separated per class by a hyperplane. The hyperplane with the largest margin between the points of the classes is often chosen. Class prediction, P, for instances in T are a result of which side of the hyperplane they are mapped to. These steps can be seen in Fig. 7.2. SVM's predicted class is either 0 or 1 and as such, prediction probabilities are distorted. For this paper SVM, as implemented in the SMO algorithm available in Weka, is chosen with the default values used. To produce proper predictions, i.e. prediction probabilities between 0 and 1, *buildLogisticModels* is set to true.

7.6.5 Performance Evaluation

The true positive rate (TPR) and false positive rate (FPR) is defined as follows.

$$TPR = \frac{TP}{TP + FN} \ . \tag{7.3}$$

$$FPR = \frac{FP}{FP + TN} \ . \tag{7.4}$$

Given a binary classifier, the Receiver Operating Characteristic (ROC) is the plot of the TPR versus the FPR, on the y-axis and x-axis respectively, for a set of instances, T. In the domain of machine learning, given a T and a corresponding set of predictions, each prediction is plotted with a one step distance relative to the previous point [21]. If $p \in P$ is equal to c the instance is plotted along the y-axis, otherwise along the x-axis.

The area under the ROC curve (AUC) single point measure is used for evaluating the classifier performance, consisting of the portion beneath the ROC curve of the plot area. The larger portion of the plot area, i.e. higher AUC, denotes a higher performance. The AUC does not depend on an equal class distribution and misclassification cost [22]. In this paper, the weighted AUC (the average AUC of the classes) is used as a single point measure.

Table 7.2: Data model comparison

Model	TPR (STD)	FPR (STD)	AUC (STD)
Social	0.953 (0.010)	0.380 (0.032)	0.926 (0.011)
Combined	0.992 (0.004)	0.000 (0.001)	0.999 (0.001)
Traditional	0.990 (0.004)	0.002 (0.003)	0.999 (0.001)

The results of the performance of SMO on the different models.

7.7 Results

Table 7.3 shows that, even though the number of attributes are lower, the classifier is still capable of producing good results. Compared to the traditional model, the results in Table 7.3 show a lower FPR for the combined model.

Table 7.2 shows a comparison between the different data models, with the advantage of showing OSN metrics as a individual data model. While the social data model produces a good weighted AUC of 0.926, it has still got quite a high FPR of 0.380. Nevertheless, the result of the social model is good.

The performance of the different models are shown in Fig. 7.3, depicting the weighted AUC. The performance of the social model shows the feasibility of using the metrics and method suggested. To improve the model, the FPR need to be decreased. While the traditional and Combined model have high AUCs, this is most likely due to the time span that the messages was collected. A longer collection time, should result in a lower score as similarities between messages are fewer.

Nevertheless, the FPR of the combined model is lower than the traditional model. When calculating the statistical significance of the TPR for the different models, combined and traditional model ranges from 0.984 to 0.999 and 0.982 and 0.997 respectively.

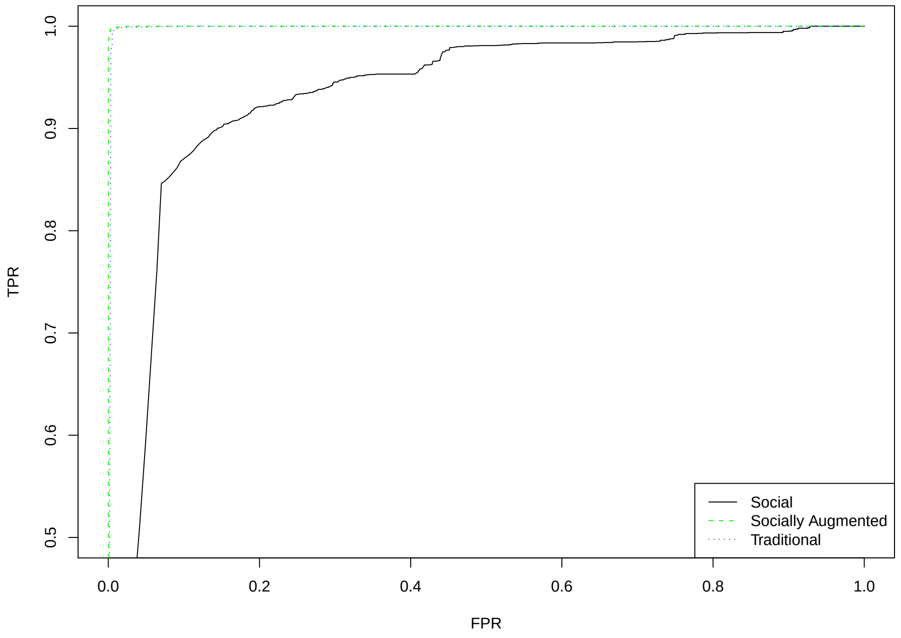


Figure 7.3: AUC for the Social, Combined and Traditional model. The y-axis is limited to show the differences between the Traditionally and Combined set.

Table 7.3: Feature Selection impact

Model (CPD*)	TPR (STD)	FPR (STD)	FNR (STD)	AUC (STD)	Nr. Attributes
Combined (0.5)	0.998 (0.003)	0.026 (0.016)	0.002 (0.003)	0.995 (0.004)	1445
Traditional (0.5)	0.989 (0.011)	0.007 (0.005)	0.011 (0.011)	0.999 (0.002)	1442
Combined (0.6)	0.993 (0.012)	0.015 (0.017)	0.007 (0.012)	0.996 (0.003)	1734
Traditional (0.6)	0.981 (0.016)	0.004 (0.003)	0.019 (0.016)	0.997 (0.003)	1730
Combined (0.7)	0.968 (0.010)	0.000 (0.001)	0.032 (0.010)	0.998 (0.001)	2023
Traditional (0.7)	0.969 (0.010)	0.003 (0.003)	0.031 (0.010)	0.998 (0.001)	2018
Combined (0.8)	0.990 (0.005)	0.001 (0.002)	0.010 (0.005)	0.999 (0.001)	2311
Traditional (0.8)	0.988 (0.005)	0.003 (0.003)	0.012 (0.005)	0.999 (0.001)	2307
Combined (0.9)	0.992 (0.004)	0.001 (0.001)	0.008 (0.004)	0.999 (0.001)	2600
Traditional (0.9)	0.990 (0.004)	0.002 (0.003)	0.010 (0.004)	0.999 (0.001)	2595
Combined (1.0)	0.992 (0.004)	0.000 (0.001)	0.008 (0.004)	0.999 (0.001)	2888
Traditional (1.0)	0.990 (0.004)	0.002 (0.003)	0.010 (0.004)	0.999 (0.001)	2882

* The cut off point for the feature selection algorithm.

7.8 Discussion

Many of the OSN based techniques can be used in conjunction with traditional spam filtering techniques, to reduce the amount of E-mails that need to be analyzed by the traditional technique. As such, even though a user cannot be linked to OSN, message can still be classified by traditional means.

A point that can be made is that a traditional, e.g. a Naive Bayes-based, spam classifier on a single users computer, given time and feedback, will have evolved into a personalized spam classifier. However, OSN-based classifiers do not require the same time span in order to become personalized since OSN-based classifiers use OSN data to bootstrap the feedback.

7.8.1 Social Network Information

The methods for socially aware classifications are promising, but most of the research has been done by creating Social Networks from the E-mail corpus. While the method has been successful and shows feasibility, it requires a large E-mail corpus. For Social Network data to be extracted and built from the E-mail corpus requires a large E-mail corpus for there to be enough data available.

Social information could be extended using OSN data. For example, given that a user is active on a OSN, extracting and incorporating social information, similar to data that was used in the experiments, can be done. The information available on such networks has the potential to be of significantly larger quantities. Given that an E-mail classifier gain access to a users OSN data, that data could help to classify E-mails with none or only a few E-mail messages to aid in the classification process. A consequence of this type of anti-spam filter, would be that bypassing the filter would require the spammers to personalize their spam to an infeasible extent, e.g., it would require the spammers to build a profile of every possible target behavior and E-mail relationship.

7.9 Conclusion and Future Work

This paper investigates the impact of spam classification based on social network data. The results show that accurate spam detectors can be generated from the low-dimensional social data model alone, however, spam detectors generated from combinations of the traditional and social models were more accurate than the detectors generated from either model in isolation. A theoretical model using several social network sources is presented. The social network metrics presented and used are adaptions meant to provide a normalized value for data extracted from various social networks. The performance of the social model show that the theoretical method presented merits further investigation.

For future work, a data set consisting of a larger amount of messages that can be linked to various OSN needs to be created. Given such a data set, investigating the use of OSN data sources may yield more reliable results. The generalizability of the theoretical model in this paper should be investigated on other data sets to verify the results found. The performance of additional social metrics in a low-dimensional model could also be investigated.

7.10 References

[1] S. Hedley, "A brief history of spam," *Information & Communications Technology Law*, vol. 15, no. 3, pp. 223–238, 2006.

[2] G. Grimes, M. Hough, and M. Signorella, "Email end users and spam: relations of gender and age group to attitudes and actions," *Computers in Human Behavior*, vol. 23, no. 1, pp. 318–332, 2007.

[3] Z. Duan, K. Gopalan, and X. Yuan;, "Behavioral characteristics of spammers and their network reachability properties," *IEEE International Conference on Communications*, pp. 164 – 171, 2007.

[4] D. Woods, E. S. Patterson, and E. M. Roth, "Can we ever escape from data overload? A cognitive systems diagnosis," *Cognition, Technology & Work*, vol. 4, pp. 22–36, Jan. 2002.

[5] S. Weil, D. Tinapple, and D. Woods, "New Approaches to Overcoming E-mail Overload," *Human Factors and Ergonomics Society Annual Meeting*, Jan. 2004.

[6] T. Guzella and W. Caminhas, "A review of machine learning approaches to spam filtering," *Expert Systems With Applications*, 2009.

[7] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artificial Intelligence Review*, vol. 29, no. 1, pp. 63–92, 2008.

[8] S. Youn and D. McLeod, "Efficient Spam Email Filtering using Adaptive Ontology," *ITNG '07. Fourth International Conference on Information Technology*, pp. 249–254, Mar. 2007.

[9] ——, "Spam Email Classification using an Adaptive Ontology," *Journal of Software*, vol. 2, no. 3, pp. 43–55, Sep. 2007.

[10] W. Li, S. Hershkop, and S. Stolfo, "Email archive analysis through graphical visualization," *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, 2004.

[11] M. Wang, Z. Li, L. Xiao, and Y. Zhang, "Research on behavior statistic based spam filter," *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on*, vol. 2, pp. 687 – 691, 2009.

[12] R. Nussbaum, A. Esfahanian, and P.-N. Tan;, "History-based email prioritization," *ASONAM '09: Social Network Analysis and Mining, 2009.*, pp. 364 – 365, 2009.

[13] S. Yoo, Y. Yang, F. Lin, and I.-C. Moon, "Mining social networks for personalized email prioritization," *15th ACM SIGKDD international conference on Knowledge discovery and data mining*, Jan. 2009.

[14] P. O. Boykin and V. P. Roychowdhury, "Leveraging social networks to fight spam," *Computer*, vol. 38, no. 4, pp. 61–68, Jan. 2005.

[15] A. O'Donnell, W. Mankowski, and J. Abrahason, "Using e-mail social network analysis for detecting unauthorized accounts," *Third Conference on Email and Anti-spam*, 2006.

[16] T. Tran, J. Rowe, and S. Wu, "Social email: a framework and application for more socially-aware communications," *Second International Conference on Social Informatics*, pp. 203–215, 2010.

[17] S. Rezaee, N. Lavesson, and H. Johnsson, "E-mail prioritization using online social network profile distance," *Computer Science & Applications, TMRF*, Jan. 2011.

[18] M. Hall, E. Frank, G. Holmes, and B. Pfahringer, "The Weka data mining software: An update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[19] M. Simeon and R. Hilderman, "Categorical proportional difference: A feature selection method for text categorization," *Seventh Australasian Data Mining Conference*, pp. 201–208, 2008.

[20] F. Sebastiani, "Classification of text, automatic," *The Encyclopedia of Language and Linguistics*, pp. 457–462, Jan. 2006.

[21] I. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques," Jan. 2005.

[22] T. Fawcett, *ROC graphs: Notes and practical considerations for data mining representation*. HP Laboratories, 2003.

ABSTRACT

Background: Computer users often need to distinguish between good and bad instances of software and e-mail messages without the aid of experts. This decision process is further complicated as the perception of spam and spyware varies between individuals. As a consequence, users can benefit from using a decision support system to make informed decisions concerning whether an instance is good or bad.

Objective: This thesis investigates approaches for estimating the utility of e-mail and software. These approaches can be used in a personalized decision support system. The research investigates the performance and accuracy of the approaches.

Method: The scope of the research is limited to the legal grey-zone of software and e-mail messages. Experimental data have been collected from academia and industry. The research methods used in this thesis are simulation and experimentation. The processing of user input, along with malicious user input, in a reputation system for software were investigated using simulations. The preprocessing optimization of end user license agreement classification was investigated using experimentation. The impact of social interaction data in regards to personalized e-mail classification was also investigated using experimentation.

Results: Three approaches were investigated that could be adapted for a decision support system. The results of the investigated reputation system suggested that the system is capable, on average, of producing a rating ±1 from an objects correct rating. The results of the preprocessing optimization of end user license agreement classification suggested negligible impact. The results of using social interaction information in e-mail classification suggested that accurate spam detectors can be generated from the low-dimensional social data model alone, however, spam detectors generated from combinations of the traditional and social models were more accurate.

Conclusions: The results of the presented approaches suggest that it is possible to provide decision support for detecting software that might be of low utility to users. The labeling of instances of software and e-mail messages that are in a legal grey-zone can assist users in avoiding an instance of low utility, e.g. spam and spyware. A limitation in the approaches is that isolated implementations will yield unsatisfactory results in a real world setting. A combination of the approaches, e.g. to determine the utility of software, could yield improved results.

2012:07

ISSN 1650-2140
ISBN: 978-91-7295-236-2