Informed Software Installation through License Agreement Categorization

Anton Borg, Martin Boldt, Niklas Lavesson

Information Security for South Africa

2011 Johannesburg

# Informed Software Installation through License Agreement Categorization

Anton Borg, Martin Boldt and Niklas Lavesson
Blekinge Institute of Technology
School of Computing
SE-371 79 Karlskrona, Sweden
{anton.borg, martin.boldt, niklas.lavesson}@bth.se

***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.**

***Keywords*-Parameter tuning; EULA analysis; Spyware; automated detection**

## I. 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.

### A. 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].

### B. Outline

In Section II-A we present the background, definitions and related work. Section II 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 IV. The experimental results are presented in Section V and discussed further in Section VI. The paper is then concluded with some suggestions for future work in Section VII.

## II. Background and Related Work

### A. 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

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

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.

### B. *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, $c : I \to C$, 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.

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

## III. 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 III-A. The malicious applications, counting 1,530 applications, have been provided by Lavasoft[4] and the legitimate applications, counting 5,511 applications, have been downloaded

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

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

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

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 IV, 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 III-A. 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 III-A, 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.

### A. 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 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

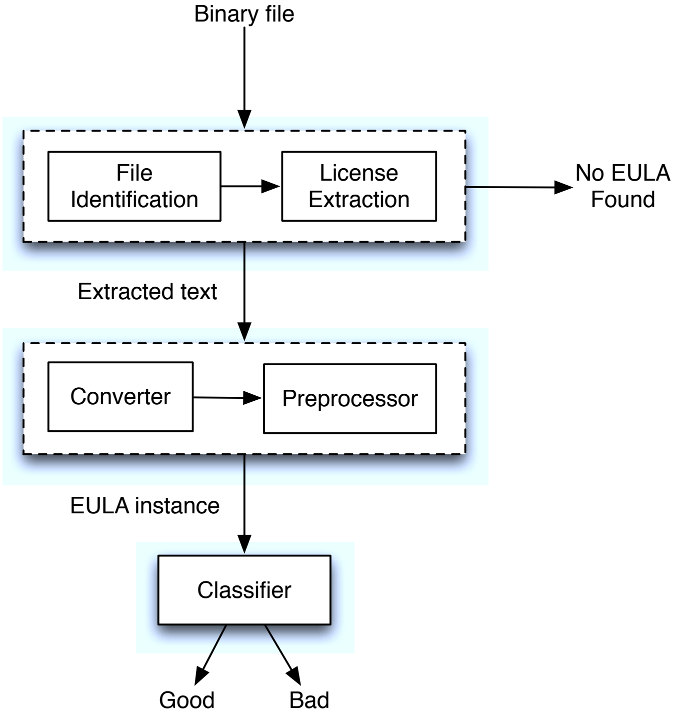

Fig. 1. A conceptual view of EULA extraction and classification.

**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 III-A.

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

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

[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/

supports a large number of filetypes and are thus suitable for our system.

The transformation stage is divided into two substages, conversion 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 III-B. 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.

### B. 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 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 $\chi^2$ 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. $\chi^2$ [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.

## IV. 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 IV-C). 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.

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

TABLE I
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] Nave Bayes Multinominal

### A. *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 I.

### B. *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.

*1) 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 I, is 1385 and based on this we get that the default value for our data set is 11.

*2) 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%.

### C. *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 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].

## V. RESULTS

### A. *Experiment 1*

As can be seen in Figure 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

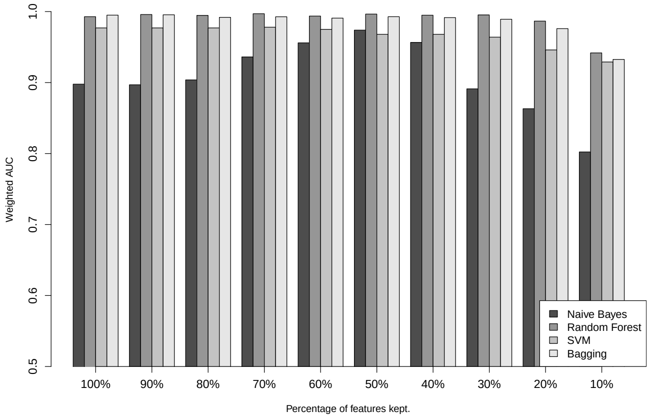

Fig. 2. AUC for Random Forest, Bagging, Naive-bayes Multinomial and SVM on data sets with different amount of kept attributes.

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 I.

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.

### B. 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.

*1) Random Forest:* Table II(a) and II(b) shows the result of the parameter tuning done on the algorithms. Looking at Table II(a) 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 II(b), 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.

*2) Bagging:* The effects of tuning the Bagging algorithm is presented in Table II(c) and II(d). The results in Table II(c) 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 II
RESULTS FOR EXPERIMENT 2

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

| Trees | Weighted AUC<br>mean (STD) | FNR<br>mean (STD) | TPR<br>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<br>mean (STD) | FNR<br>mean (STD) | TPR<br>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) |

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

| Bag size | Weighted AUC<br>mean (STD) | FNR<br>mean (STD) | TPR<br>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) |

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

| Iterations | Weighted AUC<br>mean (STD) | FNR<br>mean (STD) | TPR<br>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.

Table II(d) 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 II(c) indicates that decreasing the bag size can result in decreased performance.

## VI. 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 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.

### *A. 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.

### *B. 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.

### *C. 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 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 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.

## VII. 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.

## 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*, Montral, 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.