---

**Article:**
Milosevic, N., Dehghantanha, A. and Choo, K.-K.R. (2017) Machine learning aided Android malware classification. Computers & Electrical Engineering, 61. pp. 266-274. ISSN 0045-7906

https://doi.org/10.1016/j.compeleceng.2017.02.013

---

**manuscript No.**
(will be inserted by the editor)

# Machine learning aided malware classification of Android applications

**Nikola Milosevic · Ali Dehghantanha**

Received: date / Accepted: date

**Abstract** Malware have been used as a means for conducting cyber attacks for decades. Wide adoption of smartphones, which store lots of private and confidential information, made them an important target for malware developers. Android as the dominant mobile operating system has always been an interesting platform for malware developers and lots of Android malware species are infecting vulnerable users every day which make manual malware investigation an impossible mission. Leveraging machine learning techniques for malware forensics would assist cyber forensic investigators in their fight against malicious programs. In this paper, we present two machine learning aided approaches for static analysis of the mobile applications: one based on permissions , while the other based on source code analysis that utilizes a bag of words representation model. Our source code based classification achieved F-score of 95.1%, while the approach that used permission names only performed with F-measure of 89%. Our approach provides a method for automated static code analysis and malware detection with high accuracy and reduces smartphone malware analysis time.

**Keywords** Static malware analysis · Android · Machine learning

## 1 Introduction

According to Ericsson, in 2014., there were 2.6 billion of smartphone users and it is expected to reach 6.1 billion by 2020 (Boxall 2015). The smartphones

Nikola Milosevic
School of Computer Science,
University of Manchester
E-mail: nikola.milosevic@manchester.ac.uk

Ali Dehghantanha
School of Computing, Science and Engineering
University of Salford
E-mail: a.dehghantanha@salford.ac.uk

enabled people to have a device with a good processing power, internet access and functions of traditional mobile phones in one device, small enough to fit in a pocket. However, the wealth of private information which is stored on those devices made them an interesting target for malicious users and cyber criminals. Mobile devices are used as a means for accessing social networks, banking information, and online payment platforms which made them an important target for hackers. On the other hand, mobile devices have significant processing power that attackers can use for DDoS attacks or mining cryptocurrency (Pan et al. 2014; Felt et al. 2011). Kaspersky lab (Interpol and Kasperski lab 2014) reported that one in five mobile users was targeted by at least one type of cyber-attack over the period of one year. One in ten users was infected with malware designed to steal their money at least once. Trojans designed to send SMSs were the most widespread malicious programs in the reporting period. Over the 10 month period from August 2013 through March 2014, the number of attacks per month was up nearly tenfold, from 69,000 in August 2013 to 644,000 in March 2014. Intel Security estimated losses from cyber crime to be between 375-575 billion dollars per year and noted the trend of cyber criminals targeting mobile platforms (McAfee, Centre for Strategic & International Studies 2014). Mobile devices (smartphones and tablets) are perceived as IT security's "weakest link", followed by laptops and social media applications (Cyberedge 2014). With the rise of mobile smart devices, the number of malware and attacks on them also increased (Shaerpour et al. 2013). Smartphones have been used as a subject, objects, and tools in many cyber-crimes (Mohtasebi and Dehghantanha 2013).

Malware have been used as a means of crime in many previous cyber-attacks (Damshenas et al. 2013). The cyber attacker who was able to install malware on some system can install or delete programs, modify files, download sensitive information and use them to impersonate the user of the infected devices, upload files, monitor user's action and keystrokes, capture user's screen, use camera and retrieve photos or videos, use the infected system as a source of DDoS attack (Skoudis and Zeltser 2004). Many reported malware were targeting mobile devices (Dagon et al. 2004). Almost all different mobile operating systems were targeted by malicious programs (Damshenas et al. 2013). However, targeting operating systems with higher market share give more incentives to the malware developers. At the time of writing, a majority of mobile users use Android phones 82% (Kitagawa et al. 2015). The bigger market share of Android operating system along with Google's flexible publishing policy on the Android official application market – Google play – made Android users a popular target for malware developers. Several Trojan horses and other types of malware have been released on the Google Play (Reina et al. 2013; Viennot et al. 2014). Security firm G-Data (GData 2015) revealed that during the first quarter of 2015., nearly 5000 unique Android malware files were created each day. Android permission-based security model proved to provide very weak protection (Bai et al. 2010; Di Cerbo et al. 2010; Ongtang et al. 2012; Shin et al. 2009, 2010a,b) as most of users just grant apps requested permissions (Imgraben et al. 2014). Moreover, a lesser number of users tend to install anti-

virus and anti-malware tools on their mobile phones (Imgraben et al. 2014) which mandates a strong need for more efficient Android malware analysis tools (Shaerpour et al. 2013).

Malware analysis and software that performs malware analysis are crucial in preventing malware infection (Daryabar et al. 2013). There are two general approaches for malware analysis namely dynamic malware analysis and static malware analysis (Dezfouli et al. 2013). Static analysis is based on reviewing and inspecting of source code and binaries in order to find suspicious patterns. On the other hand, dynamic analysis (behavioral-based analysis) involves executing the analyzed software in an isolated environment, while monitoring and tracing behavior (Christodorescu and Jha 2006; Christodorescu et al. 2008; Lee and Mody 2006; Rieck et al. 2008; Schmidt et al. 2009a; Shabtai et al. 2010). Some of the early approaches to mobile malware detection were detecting anomalies in battery consumption, which could be caused by malware activities (Buennemeyer et al. 2008; Jacoby et al. 2004; Kim et al. 2008, 2011)). Operating system events, such as API calls, I/O requests, resource locks and battery consumption could be valuable for dynamic based malware detection (Schmidt et al. 2009a,b, 2010; Bläsing et al. 2010). TaintDroid is a malware detection system that monitors application's behavior and detects anomalies (Enck et al. 2014). Canfora et al. (2015) created a system that monitors six Android Dalvik op-codes and based on the monitored op-code frequencies distinguishes malicious applications from the benign ones. In order to avoid degrading of mobile device's performance, solutions based on distributed computing and collaborative analysis for both static and dynamic malware analysis were proposed (Cheng et al. 2007; Schmidt et al. 2009a; Shamili et al. 2010). M0Droid is analyzing system calls of Android applications on the server and creating signatures. Signatures are later sent to the devices so they can warn the users about threats (Damshenas et al. 2015).

It became hard to cope with the current stream of new malware on mobile platforms. The mobile malware detection techniques are still relatively immature and subject of ongoing research (Enck et al. 2014). Classical malware detection techniques are either based on signature detection, which is ineffective against encrypted, metamorphed or polymorphed malware, or on tracking the behavior of the malicious applications, which are ineffective against malware with novel behavior. Static malware analysis techniques can be used to combat these issues however, they are mostly relying on manual human analysis which limits speed and scalability of investigation (Daryabar et al. 2011). To automate static analysis process, the transformation of a source code to calculus for communicating systems (CSS) statements and using formal methods for checking software behaviour was proposed for preventing update and ransomwere attacks (Mercaldo et al. 2016a,b). However, even this approach requires human analyst to describe the unwanted behaviour by using formal logic, which could be still time consuming.

Machine learning techniques can be used to automate static malware analysis process. These techniques enable machines to use intelligent approach, learn usual attack patterns and update their knowledge as would the human

do (Nath and Mehtre 2014). Applications of machine learning in cyber security is relatively new for example in intrusion detection, malware detection, data leak detection, which in previous studies gave promising results (Gavriluţ et al. 2009; Rieck et al. 2011; Nath and Mehtre 2014). Machine learning techniques have been used in malware analysis as well. For example, Nataraj et al. (2011) represented malware in grayscale images and utilized pattern recognition approaches used in image processing in order to detect malware. Also other approaches using standard machine learning algorithms such as perceptron, SVM, locality sensitive hashing and decision trees were utilized (Gavriluţ et al. 2009; Rieck et al. 2011; Nath and Mehtre 2014). Afonso et al. (2015) extracted data from android applications about network access, process execution, string manipulation, file manipulation and information reading and then applied several machine learning algorithms such as SVM, Bayesian networks, decision trees, random forest and Naive Bayes in order to classify malware. Yerima et al. (2015) extracted 100 features based on API calls, permissions, intents and command related keywords for each of the application and applied machine learning based on Eigen space analysis in order to detect malware on android devices. Sahs and Khan used permissions and control flow graphs of android application obtained using Androguard and created SVM based machine learning model that is able to classify android malware (Sahs and Khan 2012).

While dynamic android malware analysis is properly addressed, there have not been many works on static malware analysis based on machine learning. In this paper we focus on various types of static analysis (manifest analysis, code analysis) approaches based on machine learning in order to detect malicious android applications.

The contributions of this paper are twofold. The first contribution is a machine learning model for Android malware detection based on applications permissions. This approach is lightweight, so it can be applied to a wide range of mobile devices. It is also not too computationally expensive, so it the machine learning classifier can be a part of the mobile application installed on the device. The second contribution of this work is a new approach to perform code analysis using machine learning. This complex approach provides higher accuracy and is capable of revealing detailed application behavior. So far, static code analysis of malware was a task mainly performed by people. However, this approach shows that some aspects of static code analysis, such as detecting malicious behavior of the code, are possible to automate using machine learning.

The structure of this paper is as follow. In the next section, the methodology of this research is explained. Afterwards, research results are presented followed by discussion of the results. Finally, the paper is concluded and several future works are suggested.

## 2 Methodology

In this paper, we utilized two static malware analysis approaches: in the first one we analyzed permissions the application is requesting while in the second we analyzed the whole source code of the application. In order to automate analysis process, we experimented with two machine learning approaches, namely classification and clustering.

Classification is used for identifying to which of a set of category or subpopulation a new observation belongs. It uses supervised machine learning approach in which the model is created out of existing, labeled observation examples (Michie et al. 1994). Since software can be classified into malware and goodware, the task of malware detection can be modeled as a classification problem. Clustering is a technique for unsupervised machine learning that is able to make clusters of similar entities. Clustering algorithms are useful when there is only a small portion of dataset labeled. Based on the labeled examples, it is possible to infer the class of the clustered data in the same clusters as the labeled data. The practical implication is that labels obtained in unsupervised learning, during the clustering can be later used to retrain classification model with more data. This approach is a semi-supervised learning (Basu et al. 2002; Bilenko et al. 2004).

We created four experiments, using both machine learning approaches on two analysis methods: permission-based clustering, permission-based classification, source code based clustering, and source code based classification. For training and testing of our machine learning models, we utilized M0Droid dataset, which contains 200 malicious and 200 benign android apps (Damshenas et al. 2013).

### 2.1 Permission-based analysis

In this approach, we analyze permissions that the android applications are using and we build a machine learning model that uses permission names as features. Android security model is based on apps permissions. Every application has to acquire different privileges to access a variety of phone features. During the installation, a user is notified about the permissions requested by applications and has the option to either allow access or to cancel installation of the application. However, malicious applications usually require certain permissions i.e. in order to access and exfiltrate sensitive information from the SD card, a malicious app would require access to both SD card and the internet. Our approach is modeling the combinations of Android permissions that malicious applications are using. We propose an approach to use the appearance of specific permissions as features for machine learning algorithm. In this approach, we first extracted permissions from our dataset and create a model. For training we used Weka toolkit (Hall et al. 2009) and tried several machine learning algorithms, including SVM, Naive Bayes, C4.5 Decision trees and JRIP. The advantage of permission-based analysis is that it is computa-

tionally inexpensive and it can be integrated on the mobile device. We used modified Weka 3.6.6 library for Android (Institute for Pervasive Computing 2015) to make Android application that utilizes this model. Our model, built using support vector machines with sequential minimal optimization, became a part of permission scanner in OWASP Seraphimdroid application (Milosevic 2015a,b).

We also applied several clustering techniques in order to examine and compare the performance of unsupervised learning algorithms with the supervised ones. Training, testing and evaluation were as well done using Weka toolkit. We applied Farthest First, Simple K-means and Expectation maximization (EM) algorithms provided by Weka.

### 2.2 Source code based analysis

The second approach is based on static analysis of the application's source code. Our assumption was that malicious codes are using a combination of services, methods and API calls in a way that is not usual for benign applications. Machine learning algorithms are able to learn combinations of methods, API and system calls that are common for malware and distinguish them from the patterns that are usual for benign applications. In this approach, android apps are first decompiled and then using decompiled code and text mining classification approach based on a bag of words to train a model. Decompiling Android applications to conduct static analysis involves several steps. It is first necessary to extract Dalvik Executable file (dex file) from the Android application package (APK file) by unzipping Android application package. The second step is to transform Dalvik Executable file to Java archive using dex2jar tool (Pan 2014). Afterwards, we extract .class files from the Java archive and utilize Procyon Java decompiler (version 0.5.29) to decompile .class files and create .java files. Then we merge all Java source code files of the same application into one large source file for further processing.

Since Java and natural language text do have a certain amount of similarity, we applied the technique that is used in classification of natural language processing called a bag of words. In this technique, the text, or code in our case, is represented as a bag or set of words, disregarding grammar or word order. The model is taking into account all the appearing words (Joachims 1998; McCallum et al. 1998). Our approach considered whole code (including import statements, method calls, arguments, instructions, etc.). The source code obtained in previous step was tokenized into unigrams that are used as bag of words. We used several machine learning algorithms for classifications namely C4.5 decision trees (in Weka toolkit called J48), Naive Bayes, support vector machines with sequential minimal optimization, random forests, JRIP, logistic regression and AdaBoostM1 with SVM base. We performed our training, testing and evaluation using Weka toolkit. For source code analysis we also applied ensemble learning with combinations of three and five algorithms and majority voting decision system. Again, the number of algorithms were

chosen, so system is able to unambiguously choose the output class based on majority of votes.

We also experimented with clustering on the source code. Clustering algorithms we used include Farthest First, Simple K-means and Expectation maximization (EM). Flow diagram of the process is presented in Figure 2.

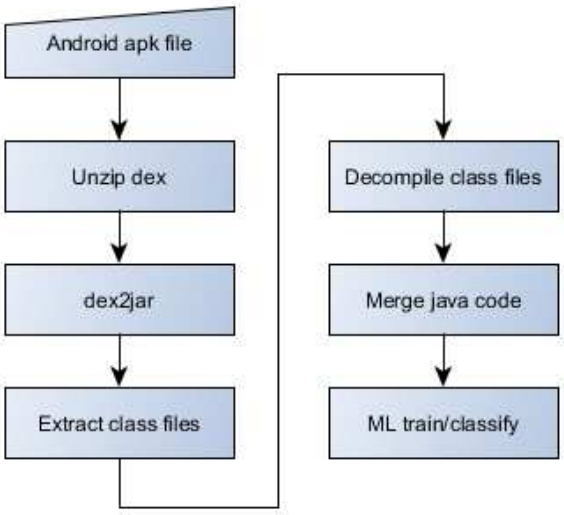


**Fig. 1** Workflow of android file decompiling and machine learning based malware detection methodology

## 2.3 Ensemble learning

In order to improve the performance of classical machine learning algorithms, we performed tests using ensemble learning with voting on both approaches: permission-based and source code based analysis approach. Ensemble methods are using multiple classification algorithms to obtain better performance than it could be obtained from any single of the constituent algorithms. The final prediction is chosen as the label that was predicted by the majority of classifiers (Kuncheva 2004; Kittler et al. 1998). We have experimented ensembles that contained combinations of three and five algorithms. We have chosen number of algorithms as an odd number, where majority voting can unambiguously choose the class. For classification algorithms, we used SVM, C4.5, decision trees, random tree, random forests, JRIP and linear regression.

## 3 Evaluation and Discussion

We evaluated the performance of our approaches using 10-fold cross validation (Kohavi et al. 1995). In 10-fold cross validation, the original sample is randomply partitioned into 10 equal sized sub-samples. A single sub-sample is retained for the testing, while 9 is used for training. The process is repeated 10 times, each time using different sub-sample for testing. The results can be then averaged to produce single estimation. The advantage of this method is that all samples are used for validation exactly once (Bengio and Grandvalet 2004; Rodriguez et al. 2010). For all algorithms, the 10-fold cross validation was executed in the same manner. As metrics for the evaluation of the algorithms, we used precision, recall and F-measure. These metrics are well adopted for classification task and widely used in text mining and machine learning communities (Makhoul et al. 1999; Flach 2003; Boiy and Moens 2009; Milne and Witten 2008; Kubat et al. 1998; Lewis and Gale 1994). Classified items can be true positive (TP - items correctly labeled as belonging to the class), false positive (FP  items incorrectly labeled as belonging to the certain class), false negative (FN  items incorrectly labeled as not belonging to the certain class) and true negative (TN  items correctly labelled as not belonging to certain class).

Given the number of true positives and false negatives, recall is calculated using following formula:

$$Recall = \frac{TP}{(TP + FN)}$$

The recall is sometimes referred as sensitivity or the true positive rate. Given the number of true positive and false positive classified items, precision is calculated as:

$$Precision = \frac{TP}{(TP + FP)}$$

Precision is sometimes referred as positive predictive rate. The measure that combines precision and recall is called F-measure.

$$F = \frac{(1 + \beta^2) * Recall * Precision}{\beta^2 * Precision + Recall)}$$

The variable $\beta$ indicates the relative value of precision. A value of $\beta = 1$, which is usually used, indicates the equal value of recall and precision, whereas lower values indicate more emphasis on precision and higher values indicate more emphasis on recall (Hersh 2005).

### 3.1 Evaluation of permission-based classification

The evaluation of machine learning classification algorithm on permission-based classification is presented in Table 1.

| Algorithm | Precision | Recall | F-Score |
|---|---|---|---|
| C4.5 decision trees | 0.827 | 0.827 | 0.827 |
| Random forest | 0.871 | 0.866 | 0.865 |
| Bayes Networks | 0.747 | 0.747 | 0.747 |
| SVM with SMO | 0.879 | 0.879 | 0.879 |
| JRip | 0.821 | 0.819 | 0.819 |
| Logistic regression | 0.823 | 0.822 | 0.821 |

**Table 1** Evaluation results of permission-based classification using single machine learning algorithms

As it can be observed from the table, support vector machines with sequential minimal optimization performed the best with F-measure of 0.879. This algorithm also correctly classified 87.9% of test instances in 10-fold cross validation. The algorithm is efficient in terms of speed. Only 0.04 seconds were needed to train model and instances are classified very fast, so this approach is suitable for live classification of the applications. We have integrated this model for classification based on permissions with SVM in OWASP Seraphimdroid Android application (Milosevic 2015a) that can be obtained from Google Play store. This proves that modern smartphones have enough computational power to perform machine learning classification based on the permissions.

On the other hand, Bayesian algorithms such as Naive Bayes and Bayesian networks performed the worst. This may be due to the small dataset, containing only 387 instances. Bayesian algorithms usually require much more data than SVM in order to train the model with high accuracy (Amari and Wu 1999). With more data, even SVM model may slightly improve (Wu and Dietterich 2004).

SVM algorithm performs better on statistical t-test with confidence interval of 0.05 than Naive Bayes, Bayesian Network, JRip and Logistic regression, however, it is statistically not significantly better than decision trees and random forests.

In Table 2., we present the results of ensemble learning using majority voting. We experimented with the ensembles of three algorithms in order to see which algorithms contribute to the best results in ensembles. Three algorithms that performed the best were SVM with SMO, Logistic regression and Random forest performing with an F-measure of 0.891 and classifying 89.1% correctly. This is only a slight improvement compared to SVM algorithm alone and t-test showed it is not significantly better with the confidence interval of 0.05.

On the other hand, ensemble algorithms performed much slower. Obviously, they need more time to apply multiple, in our case three or five machine learning algorithms and then to post-process results. Since the significance test showed that the performance of the ensemble learning algorithm is not significantly better than single machine learning algorithm, there is no real sense of using these algorithms in production.

Both results from the single classifier and from ensemble method present a promising performance that can be used in anti-malware systems and that will perform sufficiently well. It will also be able to recognize unseen malware

that were not yet released since it does not rely on signatures, but rather on learned dangerous permission combinations. Previously has been shown that machine learning algorithms perform well achieve high detection rates, even on completely new, previously unseen malware (Kolter and Maloof 2006).

| Algorithm | Precision | Recall | F-Score |
|---|---|---|---|
| Random tree+Random forest+C4.5 | 0.878 | 0.876 | 0.876 |
| Random tree+Random forest+SVM with SMO | 0.885 | 0.884 | 0.884 |
| SVM with SMO+Logistic regression+Random forest | 0.892 | 0.891 | 0.891 |
| Bayes Nets+SVM with SMO+Logistic Regression | 0.879 | 0.876 | 0.876 |
| C4.5+ Random forests+ Random tree+ SVM with SMO+Logistic regressing | 0.895 | 0.894 | 0.894 |

**Table 2** Evaluation results of permission-based classification using ensemble learning

However, this approach also do have some limitations. For permission-based approach we reported F-measure of 87.9% for single machine learning algorithms. This means there are chances that some malware are not classified as such and some benign applications are classified as malicious. In our case, 340 applications were correctly classified, while 47 were incorrectly classified. Using ensemble learning the number of misclassified instances dropped to 42. However, it is believed that anti-malware solutions are only capable of detecting 19% of zero-day malware while in a month their detection rate is just increased to 61% (Cyveillance 2010). We believe that permission-based classification approach could aid malware detection, especially on Android devices. Our reported performance is higher than the one reported by Cyveillence. Also, permission-based analysis using machine learning is computationally inexpensive and can be a part of a mobile application installed on the device. OWASP Seraphimdroid application is able to scan and classify all the installed applications (83 apps on the test device) on Nexus 5 device under 8 seconds.

### 3.2 Evaluation of permission-based clustering

Table 3 presents the results of our permission-based clustering approach. As it can be seen from the table, the results are not as good as classification results. The best algorithm is incorrectly clustering more than 35% of instances. permission-based classification incorrectly classifier around 10.5% of instances.Clustering is grouping similar items together, without any knowledge of how the grouping should be performed. In this sense, clustering is different from supervised learning, where training set is defined in a way to show how to perform classification. In clustering, there are no labels and no training set. The set of elements is clustered into the certain number of groups usually based on the elements' similarity. In our case, applications will be grouped into the groups of applications that are using a similar set of permissions. However, applications that do use the similar set of permission as some malware do not need to be malicious and vice versa.

In our case of permission-based analysis, clustering showed higher error rate than classification and it can be hardly used in malware detection.

| Algorithm | Correctly clustered instances | Incorrectly clustered instances |
|---|---|---|
| SimpleKMeans | 229 (59.17%) | 158.0 (40.83%) |
| FarthestFirst | 199 (51.42%) | 188.0 (48.58%) |
| EM | 250 (64.6%) | 137.0 (35.4%) |

**Table 3** Evaluation results of permission-based clustering

### 3.3 Evaluation of source code based classification

Out of 400 applications in our data set, we were unable to decompile 32 apps (10 benign and 22 malicious). This might be due to code encryption and obfuscation or instability of our Java decompiler. However, the remaining 368 source files are sufficient to train a good model for the purpose of this research.

The evaluation of classification that is analyzing source code of the mobile application is presented in Table 4.

| Algorithm | Precision | Recall | F-Score |
|---|---|---|---|
| C4.5 decision trees | 0.886 | 0.886 | 0.886 |
| Random forest | 0.937 | 0.935 | 0.935 |
| Naive Bayes | 0.825 | 0.821 | 0.820 |
| Bayesian networks | 0.825 | 0.821 | 0.819 |
| SVM with SMO | 0.952 | 0.951 | 0.951 |
| JRip | 0.916 | 0.916 | 0.916 |
| Logistic regression | 0.935 | 0.935 | 0.935 |

**Table 4** Evaluation results of source code based classification using single machine learning algorithm

As it can be seen from the table, over 95% of instances are correctly classified by using Support Vector Machines algorithm. The high accuracy of source code based classification approach reveals that the machine can learn from the source code about application's behavior. Even though bag of word model disregards grammar and word order in text, in our case source code, it is possible to train successful machine learning model that is able to determine which applications are malicious and which are not. Other machine learning algorithms such as Random forests, logistic regression and JRip also performed with F-score over 90%. This indicates that, as hypothesized, source code can provide sufficient amount of information for successful machine learning classification algorithm. Also, with the machine learning based source code analysis, it is possible to analyze whether some android package (apk) file is malicious or not in less than 10 seconds, which is far behind human analyst capabilities.

| Algorithm | Precision | Recall | F-Score |
|---|---|---|---|
| C4.5 decision tree+random tree+random forests | 0.950 | 0.948 | 0.948 |
| Logistic regression+C4.5+SVM with SMO | 0.947 | 0.946 | 0.946 |
| Random tree+Random Forest+SVM with SMO | 0.825 | 0.821 | 0.820 |
| SVM with SMO+Logistic regression+Random forest | 0.942 | 0.940 | 0.940 |
| SVM with SMO+Logistic regression+ AdaBoostM1 with SVM base | 0.952 | 0.951 | 0.951 |
| Logistic regression+JRip+Random Forests+ C4.5+SVM with SMO | 0.950 | 0.948 | 0.948 |
| SVM with SMO+Logistic regression+ Simple Logistic regression+AdaBoostM1 with SVM base | 0.958 | 0.957 | 0.956 |

**Table 5** Evaluation results of source code based classification using ensemble learning

In Table 5, we present the results of ensemble learning methods. Ensemble learning with voting gave a slight improvement compared to the best results by using single machine learning algorithm (Ensemble learning best F-measure - 0.956; SVM alone F-measure 0.951) by combining SVM with SMO, logistic regression, LogitBoost with simple regression functions as base learners (simple logistic regression) and AdaBoostM1 with SVM as a base. Some of the ensembles (i.e. C4.5 decision tree+random tree+random forests or SVM with SMO+Logistic regression+Random forest) performed worse than SVM with SMO. Since the F-measure of C4.5 decision trees alone was 0.886, it was negatively affecting ensembles with algorithms that performed better than C4.5. Even in ensembles that contained SVM may have misclassified some instances if the majority of algorithms voted for the wrong class. The combination of algorithms in one case (SVM with SMO+Logistic regression+Simple Logistic regression+AdaBoostM1 with SVM base) had a positive impact on the classification performance. However, the improvement of 0.5% in F-measure was not statistically significant.

Our source code analysis approach requires source codes to be decompiled which could be an issue since decompiling can hardly be done on the device because of the operating system restriction and needed computational power. Also, many malware developers are packing or obfuscating their codes to make it even more difficult to be reversed or decompiled. However, from decompiled codes, it is possible to classify successfully unseen malware in 95.1% of cases with single machine learning algorithm.

### 3.4 Evaluation of source based clustering

Table 6 present results of source code clustering. These results are more promising than the results obtained from permission-based clustering since the best performance for correctly clustered instances rose from 64.6% to 82.3%. The increase in performance is due to the fact that source code provides a greater amount of details based on which clustering can be done. However, there are still 17.6% incorrectly clustered instances. Since clustering is unsupervised machine learning algorithm, it creates clusters based on code similarity, which is

not necessary a good indication of code's behavior in terms of maliciousness. The way clustering is mapping instances without supervision is the main reason for worse performance than classification. The results for non-supervised learning are acceptable for creating larger labeled data sets (Raskutti et al. 2002). Classification performed 14% better (SVM), which indicates that clustering should not be used for detecting malware, but rather only for expanding small data sets if necessary.

| Algorithm | Correctly clustered instances | Incorrectly clustered instances |
|---|---|---|
| SimpleKMeans | 303 (82.3%) | 65 (17.66%)) |
| FarthestFirst | 296 (80.44%) | 72 (19.56%) |
| EM | 300 (81.53%) | 68 (18.47%) |

**Table 6** Evaluation results of permission-based clustering

## 4 Conclusion and Future Works

In this paper, we presented two machine learning (classification and clustering) aided approaches based on app permissions and source code analysis to detect malware on Android devices. The major advantage of these methods is that the use of machine learning enables them to detect unseen malware families with very high precision and recall.

Current commercial signature-based anti-malware solutions are incapable of detecting malicious software until the release of appropriate signatures that may take some time. Also, malware that are targeting certain organizations or individuals are not well spread and anti-malware companies might miss them (Pan et al. 2011). However, static analysis with the help of machine learning could help in identifying new, zero-day malware with relatively high precision and recall.

In this paper, we showed that certain aspects of static analysis software, especially for the fact whether or not software is malicious, can be achieved by machine learning algorithm quite successfully. The permission-based method was able to classify malware from goodware in 89% of cases while source code analysis classification performance was over 95%. The performance of 95.1%, produced by SVM, and 95.6%, produced by ensemble learning method, is comparable with the current state-of-the art in the field, performing slightly better than approach by Canfora et al. (2015) and similar to approach by Afonso et al. (2015). Both of these high performing approaches from 2015. used dynamic analysis of the android applications in combination with machine learning. Our approach is, to the best of our knowledge, the only automated static malware analysis method for android applications that uses machine learning. Clustering and unsupervised learning methods are worse for predicting whether the application is malicious or not, since they base their learning on similarities between different instances.

In the future, we will examine the combination of the proposed permission and source code analysis and how it will affect results. Also, bigger labeled balanced data set and online learning can be helpful to improve results. Another improvement may come from combining static and dynamic software analysis in which multiple machine learning classifier would be applied to analyze both source code and dynamic features of application in run-time.

## References

V.M. Afonso, M.F. de Amorim, A.R.A. Grégio, G.B. Junquera, P.L. de Geus, Identifying android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques **11**(1), 9–17 (2015)

S.-i. Amari, S. Wu, Improving support vector machine classifiers by modifying kernel functions. Neural Networks **12**(6), 783–789 (1999)

G. Bai, L. Gu, T. Feng, Y. Guo, X. Chen, Context-Aware Usage Control for Android., in *SecureComm*, vol. 10, Springer, 2010, pp. 326–343. Springer

S. Basu, A. Banerjee, R. Mooney, Semi-supervised clustering by seeding, in *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*, Citeseer, 2002, pp. 19–26. Citeseer

Y. Bengio, Y. Grandvalet, No unbiased estimator of the variance of k-fold cross-validation. Journal of Machine Learning Research **5**(Sep), 1089–1105 (2004)

M. Bilenko, S. Basu, R.J. Mooney, Integrating constraints and metric learning in semi-supervised clustering, in *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004, pp. 81–88. ACM

T. Bläsing, L. Batyuk, A.-D. Schmidt, S.A. Camtepe, S. Albayrak, An android application sandbox system for suspicious software detection, in *Malicious and unwanted software (MALWARE), 2010 5th international conference on*, IEEE, 2010, pp. 55–62. IEEE

E. Boiy, M.-F. Moens, A machine learning approach to sentiment analysis in multilingual web texts. Information retrieval **12**(5), 526–558 (2009)

A. Boxall, 2015, The number of smartphone users in the world is expected to reach a giant 6.1 billion by 2020. http://www.digitaltrends.com/mobile/smartphone-users-number-6-1-billion-by-2020/

T.K. Buennemeyer, T.M. Nelson, L.M. Clagett, J.P. Dunning, R.C. Marchany, J.G. Tront, Mobile device profiling and intrusion detection using smart batteries, in *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, IEEE, 2008, pp. 296–296. IEEE

G. Canfora, F. Mercaldo, C.A. Visaggio, Mobile malware detection using op-code frequency histograms, in *Proceedings of International Conference on Security and Cryptography (SECRYPT)*, 2015

J. Cheng, S.H. Wong, H. Yang, S. Lu, Smartsiren: virus detection and alert for smartphones, in *Proceedings of the 5th international conference on Mobile systems, applications and services*, ACM, 2007, pp. 258–271. ACM

M. Christodorescu, S. Jha, Static analysis of executables to detect malicious patterns, Technical report, DTIC Document, 2006

M. Christodorescu, S. Jha, C. Kruegel, Mining specifications of malicious behavior, in *Proceedings of the 1st India software engineering conference*, ACM, 2008, pp. 5–14. ACM

Cyberedge, CYBERTHREAT DEFENSE REPORT, Technical report, 2014

Cyveillance, 2010, Cyveillance Testing Finds AV Vendors Detect on Average Less Than 19% of Malware Attacks. http:www.businesswire.comnewshome20100804005348enCyveillance-Testing-Finds-AV-Vendors-Detect-Average

D. Dagon, T. Martin, T. Starner, Mobile phones as computing devices: The viruses are coming! Pervasive Computing, IEEE **3**(4), 11–15 (2004)

M. Damshenas, A. Dehghantanha, 2013, M0Droid. http:m0droid.netai.netmodroid

M. Damshenas, A. Dehghantanha, R. Mahmoud, A survey on malware propagation, analysis, and detection. International Journal of Cyber-Security and Digital Forensics (IJCSDF) **2**(4), 10–29 (2013)

M. Damshenas, A. Dehghantanha, K.-K.R. Choo, R. Mahmud, M0droid: An android behavioral-based malware detection model. Journal of Information Privacy and Security **11**(3), 141–157 (2015)

F. Daryabar, A. Dehghantanha, N.I. Udzir, Investigation of bypassing malware defences and malware detections, in *Information Assurance and Security (IAS), 2011 7th International Conference on*, IEEE, 2011, pp. 173–178. IEEE

F. Daryabar, A. Dehghantanha, N.I. Udzir, M. Sani, S.b. Shamsuddin, F. Norouzizadeh, et al., Analysis of known and unknown malware bypassing techniques. International Journal of Information Processing & Management **4**(6) (2013)

F.N. Dezfouli, A. Dehghantanha, R. Mahmod, N.F.B.M. Sani, S.B. Shamsuddin, F. Daryabar, A survey on malware analysis and detection techniques. International Journal of Advancements in Computing Technology **5**(14), 42–51 (2013)

F. Di Cerbo, A. Girardello, F. Michahelles, S. Voronkova, Detection of malicious applications on android os. Computational Forensics, 138–149 (2010)

W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, A.N. Sheth, Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) **32**(2), 5 (2014)

A.P. Felt, M. Finifter, E. Chin, S. Hanna, D. Wagner, A survey of mobile malware in the wild, in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ACM, 2011, pp. 3–14. ACM

P.A. Flach, The geometry of ROC space: understanding machine learning metrics through ROC isometrics, in *ICML*, 2003, pp. 194–201

D. Gavriluţ, M. Cimpoeşu, D. Anton, L. Ciortuz, Malware detection using machine learning, in *Computer Science and Information Technology, 2009. IMCSIT'09. International Multiconference on*, IEEE, 2009, pp. 735–741. IEEE

GData, 2015, G DATA Mobile Malware Report

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update. ACM SIGKDD explorations newsletter **11**(1), 10–18 (2009)

W. Hersh, Evaluation of biomedical text-mining systems: lessons learned from information retrieval. Briefings in bioinformatics **6**(4), 344–356 (2005)

J. Imgraben, A. Engelbrecht, K.-K.R. Choo, Always connected, but are smart mobile users getting more security savvy? a survey of smart mobile device users. Behaviour & Information Technology **33**(12), 1347–1360 (2014)

Institute for Pervasive Computing, 2015, Pervasive Computing Infrastructure. https://www.pervasive.jku.at/Teaching/lvaInfo.php?key=346&do=uebungen

Interpol and Kasperski lab, Mobile Cyber Threats, Technical report, 2014

G.A. Jacoby, R. Marchany, N.J. Davis IV, Battery-based intrusion detection a first line of defense, in *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, IEEE, 2004, pp. 272–279. IEEE

T. Joachims, Text categorization with support vector machines: Learning with many relevant features, in *European conference on machine learning*, Springer, 1998, pp. 137–142. Springer

H. Kim, K.G. Shin, P. Pillai, Modelz: monitoring, detection, and analysis of energy-greedy anomalies in mobile handsets. Mobile Computing, IEEE Transactions on **10**(7), 968–981 (2011)

H. Kim, J. Smith, K.G. Shin, Detecting energy-greedy anomalies and mobile malware variants, in *Proceedings of the 6th international conference on Mobile systems, applications, and services*, ACM, 2008, pp. 239–252. ACM

M. Kitagawa, A. Gupta, R. Cozza, I. Durand, D. Glenn, K. Maita, L. Tay, T. Tsai, R. Atwal, M. Escherich, E. He, A. Jump, B. Lakehal, C. Lu, T.H. Nguyen, A. Sato, V. Tripathi, A. Zimmermann, W. Lutman, Market Share: Final PCs, Ultramobiles and Mobile Phones, All Countries, 2Q15 Update, Technical report, 2015

J. Kittler, M. Hatef, R.P. Duin, J. Matas, On combining classifiers. Pattern Analysis and Machine Intelligence, IEEE Transactions on **20**(3), 226–239 (1998)

R. Kohavi, et al., A study of cross-validation and bootstrap for accuracy estimation and

model selection, in *Ijcai*, vol. 14, 1995, pp. 1137–1145
J.Z. Kolter, M.A. Maloof, Learning to detect and classify malicious executables in the wild. The Journal of Machine Learning Research **7**, 2721–2744 (2006)
M. Kubat, R.C. Holte, S. Matwin, Machine learning for the detection of oil spills in satellite radar images. Machine learning **30**(2-3), 195–215 (1998)
L.I. Kuncheva, *Combining pattern classifiers: methods and algorithms* (John Wiley & Sons, New Jersey, United States, 2004)
T. Lee, J.J. Mody, Behavioral classification, in *EICAR Conference*, 2006, pp. 1–17
D.D. Lewis, W.A. Gale, A sequential algorithm for training text classifiers, in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, Springer-Verlag New York, Inc., 1994, pp. 3–12. Springer-Verlag New York, Inc.
J. Makhoul, F. Kubala, R. Schwartz, R. Weischedel, et al., Performance measures for information extraction, in *Proceedings of DARPA broadcast news workshop*, 1999, pp. 249–252
McAfee, Centre for Strategic & International Studies, Net Loses: Estimating the Global Cost of Cybercrime, Technical report, 2014
A. McCallum, K. Nigam, et al., A comparison of event models for naive bayes text classification, in *AAAI-98 workshop on learning for text categorization*, vol. 752, Citeseer, 1998, pp. 41–48. Citeseer
F. Mercaldo, V. Nardone, A. Santone, C.A. Visaggio, Download malware? no, thanks: how formal methods can block update attacks, in *Proceedings of the 4th FME Workshop on Formal Methods in Software Engineering*, ACM, 2016a, pp. 22–28. ACM
F. Mercaldo, V. Nardone, A. Santone, C.A. Visaggio, Ransomware Steals Your Phone. Formal Methods Rescue It, in *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, Springer, 2016b, pp. 212–221. Springer
D. Michie, D.J. Spiegelhalter, C.C. Taylor, Machine learning, neural and statistical classification (1994)
D. Milne, I.H. Witten, Learning to link with wikipedia, in *Proceedings of the 17th ACM conference on Information and knowledge management*, ACM, 2008, pp. 509–518. ACM
N. Milosevic, 2015a, OWASP Seraphimdroid GitHub page. https://github.com/nikolamilosevic86/owasp-seraphimdroid
N. Milosevic, 2015b, OWASP Seraphimdroid project page. https://www.owasp.org/index.php/OWASP_SeraphimDroid_Project
S. Mohtasebi, A. Dehghantanha, Towards a unified forensic investigation framework of smartphones. International Journal of Computer Theory and Engineering **5**(2), 351–355 (2013)
L. Nataraj, S. Karthikeyan, G. Jacob, B. Manjunath, Malware images: visualization and automatic classification, in *Proceedings of the 8th international symposium on visualization for cyber security*, ACM, 2011, p. 4. ACM
H.V. Nath, B.M. Mehtre, Static malware analysis using machine learning methods. Recent Trends in Computer Networks and Distributed Systems Security, 440–450 (2014)
M. Ongtang, S. McLaughlin, W. Enck, P. McDaniel, Semantically rich application-centric security in android. Security and Communication Networks **5**(6), 658–673 (2012)
B. Pan, 2014, dex2jar. https://github.com/pxb1988/dex2jar
J.J.Y. Pan, C.C. Fung, et al., Boutique malware–custom made attacks on e-business. In preceedings of The 9th International Conference on e-Business (iNCEB2010), 108–112 (2011)
X. Pan, Y. Zhongyang, Z. Xin, B. Mao, H. Huang, Defensor: Lightweight and Efficient Security-Enhanced Framework for Android, in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, IEEE, 2014, pp. 260–267. IEEE
B. Raskutti, H. Ferrá, A. Kowalczyk, Combining clustering and co-training to enhance text classification using unlabelled data, in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2002, pp. 620–625. ACM
A. Reina, A. Fattori, L. Cavallaro, A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. EuroSec, April (2013)

K. Rieck, T. Holz, C. Willems, P. Düssel, P. Laskov, Learning and classification of malware behavior, 108–125 (2008)

K. Rieck, P. Trinius, C. Willems, T. Holz, Automatic analysis of malware behavior using machine learning. Journal of Computer Security **19**(4), 639–668 (2011)

J.D. Rodriguez, A. Perez, J.A. Lozano, Sensitivity analysis of k-fold cross validation in prediction error estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence **32**(3), 569–575 (2010)

J. Sahs, L. Khan, A machine learning approach to android malware detection, in *Intelligence and Security Informatics Conference (EISIC), 2012 European*, IEEE, 2012, pp. 141–147. IEEE

A.-D. Schmidt, S.A. Camtepe, S. Albayrak, Static smartphone malware detection (2010)

A.-D. Schmidt, J.H. Clausen, A. Camtepe, S. Albayrak, Detecting symbian os malware through static function call analysis, in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, IEEE, 2009a, pp. 15–22. IEEE

A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K.A. Yüksel, S.A. Camtepe, S. Albayrak, Static analysis of executables for collaborative malware detection on android, in *Communications, 2009. ICC'09. IEEE International Conference on*, IEEE, 2009b, pp. 1–5. IEEE

A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, C. Glezer, Google android: A comprehensive security assessment. IEEE Security & Privacy, 35–44 (2010)

K. Shaerpour, A. Dehghantanha, R. Mahmod, Trends in android malware detection. The Journal of Digital Forensics, Security and Law: JDFSL **8**(3), 21 (2013)

A.S. Shamili, C. Bauckhage, T. Alpcan, Malware detection on mobile devices using distributed machine learning, in *Pattern Recognition (ICPR), 2010 20th International Conference on*, IEEE, 2010, pp. 4348–4351. IEEE

W. Shin, S. Kiyomoto, K. Fukushima, T. Tanaka, Towards formal analysis of the permission-based security model for android, in *Wireless and Mobile Communications, 2009. ICWMC'09. Fifth International Conference on*, IEEE, 2009, pp. 87–92. IEEE

W. Shin, S. Kiyomoto, K. Fukushima, T. Tanaka, A formal model to analyze the permission authorization and enforcement in the android framework, in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, IEEE, 2010a, pp. 944–951. IEEE

W. Shin, S. Kwak, S. Kiyomoto, K. Fukushima, T. Tanaka, A small but non-negligible flaw in the Android permission scheme, in *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, IEEE, 2010b, pp. 107–110. IEEE

E. Skoudis, L. Zeltser, *Malware: Fighting malicious code* (Prentice Hall Professional, New Jersey, USA, 2004)

N. Viennot, E. Garcia, J. Nieh, A measurement study of Google Play, in *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, ACM, 2014, pp. 221–233. ACM

P. Wu, T.G. Dietterich, Improving SVM accuracy by training on auxiliary data sources, in *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004, p. 110. ACM

S.Y. Yerima, S. Sezer, I. Muttik, Android malware detection: An eigenspace analysis approach, in *Science and Information Conference (SAI), 2015*, IEEE, 2015, pp. 1236–1242. IEEE