Security Informatics
a SpringerOpen Journal

RESEARCH Open Access

Detecting unknown malicious code by applying classification techniques on OpCode patterns

Asaf Shabtai[1,2*], Robert Moskovitch[1,2], Clint Feher[1,2], Shlomi Dolev[1,3] and Yuval Elovici[1,2]

* Correspondence: shabtaia@bgu.ac.il
[1]Deutsche Telekom Laboratories, Ben-Gurion University, Be'er Sheva, 84105, Israel
Full list of author information is available at the end of the article

Abstract

In previous studies classification algorithms were employed successfully for the detection of unknown malicious code. Most of these studies extracted features based on *byte n-gram* patterns in order to represent the inspected files. In this study we represent the inspected files using *OpCode n-gram* patterns which are extracted from the files after disassembly. The OpCode *n*-gram patterns are used as features for the classification process. The classification process main goal is to detect unknown malware within a set of suspected files which will later be included in antivirus software as signatures. A rigorous evaluation was performed using a test collection comprising of more than 30,000 files, in which various settings of OpCode *n*-gram patterns of various size representations and eight types of classifiers were evaluated. A typical problem of this domain is the imbalance problem in which the distribution of the classes in real life varies. We investigated the imbalance problem, referring to several real-life scenarios in which malicious files are expected to be about 10% of the total inspected files. Lastly, we present a chronological evaluation in which the frequent need for updating the training set was evaluated. Evaluation results indicate that the evaluated methodology achieves a level of accuracy higher than 96% (with TPR above 0.95 and FPR approximately 0.1), which slightly improves the results in previous studies that use byte *n*-gram representation. The chronological evaluation showed a clear trend in which the performance improves as the training set is more updated.

Keywords: Malicious Code Detection, OpCode, Data Mining, Classification

1. Introduction

Modern computer and communication infrastructures are highly susceptible to various types of attacks. A common method of launching these attacks is by means of *malicious software* (malware) such as worms, viruses, and Trojan horses, which, when spread, can cause severe damage to private users, commercial companies and governments. The recent growth in high-speed Internet connections enable malware to propagate and infect hosts very quickly, therefore it is essential to detect and eliminate new (unknown) malware in a prompt manner [1].

Anti-virus vendors are facing huge quantities (thousands) of suspicious files every day [2]. These files are collected from various sources including dedicated honeypots, third party providers and files reported by customers either automatically or explicitly. The large amount of files makes efficient and effective inspection of files particularly challenging. Our main goal in this study is to be able to filter out unknown malicious files from the files arriving to an anti-virus vendor every day. For that, we investigate the approach of representing malicious files by OpCode expressions as features in the classification task.

Several analysis techniques for detecting malware, which commonly distinguished between dynamic and static, have been proposed. In *dynamic analysis* (also known as behavioral analysis) the detection of malware consists of information that is collected from the operating system at runtime (i.e., during the execution of the program) such as system calls, network access and files and memory modifications [3-7]. This approach has several disadvantages. First, it is difficult to simulate the appropriate conditions in which the malicious functions of the program, such as the vulnerable application that the malware exploits, will be activated. Secondly, it is not clear what is the required period of time needed to observe the appearance of the malicious activity for each malware.

In *static analysis*, information about the program or its expected behavior consists of explicit and implicit observations in its binary/source code. The main advantage of static analysis is that it is able to detect a file without actually executing it and thereby providing rapid classification [8].

Static analysis solutions are primarily implemented using the *signature-based* method which relies on the identification of unique strings in the binary code [2]. While being very precise, signature-based methods are useless against unknown malicious code [9]. Thus, generalization of the detection methods is crucial in order to be able to detect unknown malware before its execution. Recently, classification algorithms were employed to automate and extend the idea of *heuristic-based* methods. In these methods the binary code of a file is represented, for example, using byte sequence (i.e., byte *n*-grams), and classifiers are used to learn patterns in the code in order to classify new (unknown) files as malicious or benign [1,10]. Recent studies, which we survey in the next section, have shown that by using byte *n*-grams to represent the binary file features, classifiers with very accurate classification results can be trained, yet there still remains room for improvement.

In this paper, which is an extended version of [11], we use a methodology for malware categorization by implementing concepts from the text categorization domain, as was presented by part of the authors in [12]. While most of the previous studies extracted features which are based on *byte n-grams* [12,13], in this study, we use *OpCode n-gram patterns*, generated by disassembling the inspected executable files, to represent the files. Unlike byte sequence, OpCode expressions, extracted from the executable file, are expected to provide a more meaningful representation of the code. In the analogy to text categorization, using letters or sequences of letters as features is analogous to using byte sequences, while using words or sequences of words is analogous to the OpCode sequences.

Another important aspect when using binary classifiers for the detection of unknown malicious code is the imbalance problem. The imbalance problem refers to scenarios in which the proportions of the classes are not equal. Previous studies presented evaluations based on test collections having similar proportions of malicious and benign files in the test collections. These proportions do not reflect real-life situations in which malicious code is significantly lower than 50% and therefore might report optimistic results. As a case in point, a recent McAfee survey [14] indicates that about 4%

of search results from the major search engines on the web contain malicious code. Additionally, Shin et al. [15] found that above 15% of the files in the KaZaA network contained malicious code.

We rigorously evaluate the framework that is suggested in this paper, using a test collection containing more than 30,000 files, in order to determine the optimal settings of the framework. Additionally, we investigate the imbalance problem and evaluate through various malicious-benign proportions, the best settings for a training set given a test set.

Another aspect in the maintenance of such a framework is the importance of updating the training set with new known malicious files. This is intuitively important, because the purpose of malicious files changes over time and accordingly the patterns within the code. Moreover, these malicious files are written in varying frameworks which result in differing patterns. However, it is not clear to what extent it is essential to retrain the classifier with the new files. For this purpose we designed a chronological experiment, based on a dataset including files from the years 2000 to 2007, trained each time on files untill year *k* and tested on the following years.

The rest of the paper is organized as follows. We begin in section 2 with a survey of previous relevant studies. Section 3 describes the methods we used, including concepts from text categorization, data preparation, and classifiers. In sections 4 and 5 we present the evaluation and the evaluation results. Lastly, section 6 discusses the results and future work.

2. Background

2.1 Detecting Unknown Malware using Byte N-Grams Patterns

Over the past decade, several studies have focused on the detection of unknown malware based on its binary code content. The authors of [16] were the first to introduce the idea of applying *Machine Learning* (*ML*) methods for the detection of different malwares based on their respective binary codes. Three different feature extraction (FE) approaches were employed: features extracted from the *Portable Executable* (*PE*) *section*, meaningful *plain-text strings* that are encoded in programs files, and *byte sequence features*.

Abou-Assaleh et al. [13] introduced a framework that uses the Common N-Gram (CNG) method and the *k*-nearest neighbor (KNN) classifier for the detection of malware. For each malicious and benign class a representative profile was constructed. A new executable file was compared with the profiles of malicious and benign classes, and was assigned to the most similar.

Kotler and Maloof [17] also used byte *n*-grams representation, however the vector of *n*-gram features was binary, presenting the presence or absence of a feature in the file and ignoring the frequency of feature appearances (in the file). In an extension of their previous study, Kolter and Maloof [18] classified malware into families (multiple classes) based on the functions in their respective payloads. In attempts to estimate their ability to detect malicious codes based on their issue dates, these techniques were trained on files issued before July 2003, and then tested on files issued from that point in time through August 2004.

Cai et al. [19] conducted several experiments in which they evaluated the combinations of seven feature selection methods, three classifiers, and byte *n*-gram size.

Recently, Moskovitch et al. [12] published the results of a study which used a test collection containing more than 30,000 files, in which the files were represented by byte *n*-grams. Additionally, an investigation of the imbalance problem, on which we elaborate later, was demonstrated. In this paper we present the results of an alternative representation of the executable files using OpCode *n*-gram patterns instead of using byte *n*-gram patterns.

2.2 Representing Executables using OpCodes

An *OpCode* (short for operational code) is the portion of a machine language instruction that specifies the operation to be performed. A complete machine language instruction contains an OpCode and, optionally, the specification of one or more operands. The operations of an OpCode may include arithmetic, data manipulation, logical operations, and program control.

The OpCodes, being the building blocks of machine language, have been used for statically analyzing application behavior and detecting malware. Karim et al. [20] addressed the tracking of malware evolution based on OpCode sequences and permutations. Data mining methods (Logistic Regression, Artificial Neural Networks and Decision Trees) are used in [21] to automatically identify critical instruction sequences that can distinguish between malicious and benign programs. The evaluation showed a high accuracy level of 98.4%. Bilar [22] examines the difference of statistical OpCode frequency distribution in malicious and non-malicious code. A total of 67 malware executables were compared with the aggregate statistics of 20 non-malicious samples. The results show that malicious software OpCode distributions differ significantly from non-malicious software and suggests that the method can be used to detect malicious code. The approach in [22] presents a single case in our methodology; in this paper we test several OpCode *n*-gram sizes while Bilar [22] used only 1-gram. Based on our experiments, using OpCode sequences improves the detection performance significantly. Santos et al. [23] used the OpCode *n*-grams (of size *n*=1,2) representation to ascribe malware instances to their families by measuring the similarity between files. This is, however, different from our goal in which we attempt to classify unknown suspicious files as malicious or benign in order to detect new malware.

Our approach also stems from the idea that there are families of malware such that two members of the same family share a common "engine." Moreover, there are malware generation utilities which use a common engine to create new malware instances; this engine may even be used to polymorph the threat as it propagates. When searching for such common engines among known malware, one must be aware that malware designers will attempt to hide such engines using a broad range of techniques. For example, these common engines may be located in varying locations inside the executables, and thus may be mapped to different addresses in memory or even perturbed slightly. To overcome such practices, we suggest disregarding any parameters of the OpCodes. We believe that disregarding the parameters would provide a more general representation of the files, which is expected to be more effective for purposes of classification into benign and malicious files.

2.3 The Imbalance Problem

The class imbalance problem was first introduced to the ML research community a little over a decade ago [24]. Typically, the class imbalance problem occurs when there are significantly more instances from one class relative to other classes. In such cases the classifier tends to misclassify the instances of the less represented classes. More and more researchers realized that the performance of their classifiers may be sub-optimal due to the fact that the datasets are not balanced. This problem is even more relevant in fields where the natural datasets are highly imbalanced in the first place [25], as in the problem we describe.

Over the years, the ML community has addressed the issue of class imbalance following two general strategies. The first strategy, which is classifier-independent, consists of balancing the original data-set by using different kinds of undersampling or oversampling approaches. In particular, researchers have experimented with random (e.g., [26]), directed (e.g., [24,26]), and artificial sampling [27]. The second strategy involves modifying the classifiers in order to adapt them to the data-sets. In particular, these approaches search for methods for incorporating misclassification costs into the classification process and assigning higher misclassification costs to the minority class so as to compensate for its small size. This was done for a variety of classifiers such as Artificial Neural Networks [28], Random Forests [29], and SVM [30].

In our case, the data is imbalanced in real-life conditions and reflected by the test-set in our experiments, therefore, we would like to understand the optimal construction of a training-set for achieving the best performance in real-life conditions. Similarly to the work of [31], in our research we also consider the question of what is the appropriate proportion of examples of each class (benign and malicious) for learning if only a limited number of training instances can be used altogether. Their work considers the case of Decision Trees induction on 26 different data-sets. We, on the other hand, focus on the single problem of interest here–malware detection–but consider eight different classifiers.

Another relevant issue to the research which emanates from the class imbalance problem concerns the choice of an evaluation metric. When faced with unequal class sizes, classification accuracy is often an inappropriate measure of performance. Indeed, in such circumstances, a trivial classifier that predicts every case as the majority class could achieve very high accuracy levels in extremely skewed domains. Several proposals have been made to address this issue including the decomposition of accuracy into its basic components [25], the use of ROC analysis [32] or the G-Mean [33]. In this paper we chose to decompose accuracy into basic components in addition to the use of the G-mean. This approach is conceptually simpler than using ROC analysis and sheds sufficient light on our results. The details of the evaluation measures we used will be given in Section 5.1.

3. Methods

The goal of our work was to explore methods of using data mining techniques in order to create accurate detectors for new (unseen) binaries. The overall process of classifying unknown files as either benign or malicious using ML methods is divided into two subsequent phases: training and testing. In the *training* phase, a training-set of benign and malicious files is provided to the system. Each file is then parsed and a vector representing each file is extracted based on a pre-determined vocabulary (which can be an outcome of setup *feature selection* process). The representative vectors of the files in the training set and their real (known) classification are the input for a learning algorithm (such as a Decision Tree or Artificial Neural Network algorithms). By processing these vectors, the learning algorithm trains a classifier. Next, during the testing phase, a test-set collection of new benign and malicious files which did not appear in the training-set are classified by the classifier that was generated in the training phase. Each file in the test-set is first parsed and the representative vector is extracted using the same vocabulary as in the training phase. Based on this vector, the classifier will classify the file as either benign or malicious. In the testing phase the performance of the generated classifier is evaluated by extracting standard accuracy measures for classifiers. Thus, it is necessary to know the real class of the files in the test-set in order to compare their real class with the class that was derived by the classifier.

3.2 Dataset Creation

We created a dataset of malicious and benign executables for the Windows operating system, the system most commonly used and attacked today. This malicious and benign file collection was previously used in [12]. We acquired 7,688 malicious files from the VX Heaven website [34]. To identify the files, we used the Kaspersky anti-virus. Benign files, including executable and DLL (Dynamic Linked Library) files, were gathered from machines running the Windows XP operating system on our campus. The benign set contained 22,735 files. The Kaspersky anti-virus program was used to verify that these files did not contain any malicious code.

Some of the files in our collection were either compressed or packed. These files could not be disassembled by disassembler software and therefore, after converting the files into OpCode representation we ended up with 5,677 malicious and 20,416 benign files (total of 26,093 files).

Code obfuscation is a prominent technique used by hackers in order to avoid detection by security mechanisms (e.g., anti-viruses and intrusion detection systems) [35]. These techniques are also applied on benign software for copyrights protection purposes. Packing and compressing files can be achieved by using off-the-shelf packers such as Armadillo, UPX and Themida. In such cases, static analysis methods might fail to correctly classify a packed malware [36]. Several solutions to the challenge of packed code were suggested (e.g., Ether [36], McBoost [37], PolyUnpack [38]). These methods were proposed for automatic unpacking of packed files by applying either static or dynamic analysis. Evaluation performed in these studies showed that unpacking files before being classified increase the classification accuracy [37,38]. Our proposed method can use such an approach in order to overcome packed files. In addition, we would like to point out that classifying benign files is also useful and can reduce the load of inspecting suspicious (or unknown) files. Also, the large number of malware files in our dataset that could be dissembled indicates that in order to appear benign and to pass security mechanisms (that are configured to block content that is encrypted\obfuscated and cannot be inspected), these techniques are not always used by hackers.

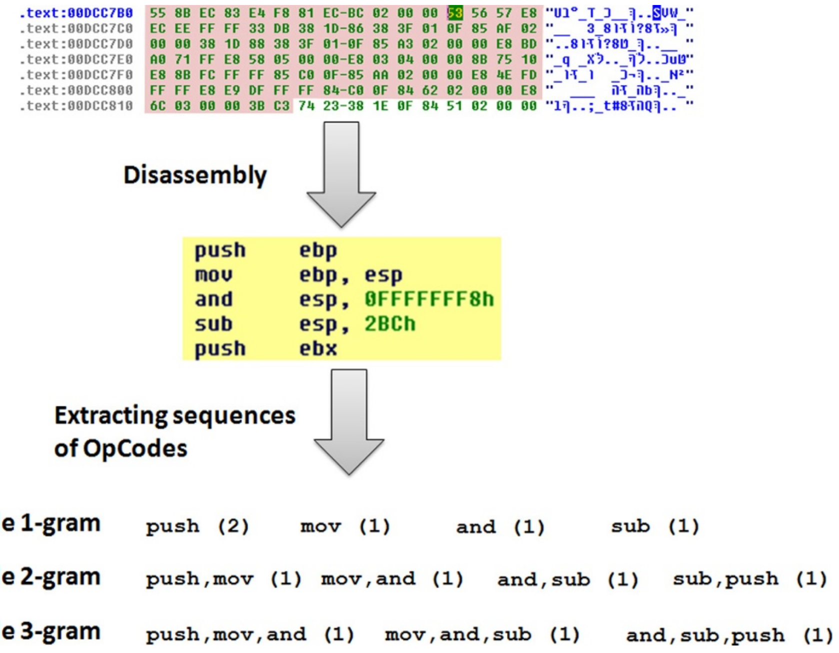
3.3 Data Preparation and Feature Selection

To classify the files we had to convert them into a vectorial representation. We had two representations, the known one, often called byte *n*-grams, which consists of byte sequences of characters extracted from the binary code [12], and the second OpCode *n*-grams represented by sequences of OpCodes. Using a disassembler software, we extracted a sequence of OpCodes from each file representing execution flow of machine operations. Subsequently, several OpCode *n*-gram lengths were considered where each *n*-gram was composed of *n* sequential OpCodes. This process is presented in Figure 1.

The process of streamlining an executable starts with *disassembling* it. The disassembly process consists of translating the machine code instructions stored in the executable to a more human-readable language, namely, *Assembly* language. The next and final step in streamlining the executable is achieved by extracting the sequence of OpCodes generated during the disassembly process. The extracting of sequences is in the same logical order in which the OpCodes appear in the executable, disregarding the extra information available (e.g., memory location, registers, etc.)

Although such a process seems trivial, malware writers often try to prevent the successful application of the disassembly process to prevent experts from analyzing their malwares. In this study we used IDA-Pro, the most advanced commercial disassembly program available today. IDA-Pro implements sophisticated techniques which enabled us to disassemble most of our malware collection successfully (approximately 74% of the malware files).

The size of vocabularies (number of distinct *n*-grams) extracted for the OpCode *n*-grams representation were of 515, 39,011, 443,730, 1,769,641, 5,033,722 and 11,948,491, for 1-gram, 2-gram, 3-gram, 4-gram, 5-gram and 6-gram, respectively. Later, the *normalized term frequency* (TF) and TF *inverse document frequency* (TFIDF) representations were calculated for each OpCode *n*-grams patterns in each file. The TF and TFIDF are well known measures in the *text categorization* field [39]. In our domain, each *n*-gram is analogous to a word (or a term) in a text document. The *normalized* TF is calculated by dividing the frequency of the term in the document by the



```
.text:00DCC7B0 55 8B EC 83 E4 F8 81 EC-BC 02 00 00 53 56 57 E8
.text:00DCC7C0 EC EE FF FF 33 DB 38 1D-86 38 3F 01 0F 85 AF 02
.text:00DCC7D0 00 00 38 1D 88 38 3F 01-0F 85 A3 02 00 00 E8 BD
.text:00DCC7E0 A0 71 FF E8 58 05 00 00-E8 03 04 00 00 8B 75 10
.text:00DCC7F0 E8 8B FC FF FF 85 C0 0F-85 AA 02 00 00 E8 4E FD
.text:00DCC800 FF FF E8 E9 DF FF FF 84-C0 0F 84 62 02 00 00 E8
.text:00DCC810 6C 03 00 00 3B C3 74 23-38 1E 0F 84 51 02 00 00
```

```
push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
sub     esp, 2BCh
push    ebx
```

```
OpCode 1-gram   push (2)          mov (1)          and (1)          sub (1)
OpCode 2-gram   push,mov (1)      mov,and (1)      and,sub (1)      sub,push (1)
OpCode 3-gram   push,mov,and (1)  mov,and,sub (1)  and,sub,push (1)
```

Figure 1 Converting byte representation into OpCode *n*-grams patterns.

frequency of the most frequent term in a document. The TFIDF combines the frequency of a term in the document (TF) and its frequency in the whole document collection, denoted by *document frequency* (DF). The term's (normalized) TF value is multiplied by the $IDF = log\ (N/DF)$, where N is the number of documents in the entire file collection and DF is the number of files in which it appears.

The TF representation is actually the representation which was used in previous papers in the domain of malicious code classification [13,16,17], where counting words was replaced by byte *n*-grams extracted from the executable files. However, in the textual domain, it was shown that the *TFIDF* is a richer and more successful representation for the retrieval and categorization purposes [39] and thus we expected that using the *TFIDF* weighting would lead to better performance than the *TF*.

In ML applications, the large number of features (many of which do not contribute to the accuracy and may even decrease it) in many domains presents a significant problem. In our study, the reduction of the number of features is crucial and must be performed while maintaining a high level of accuracy. This is due to the fact that the vocabulary size may exceed millions of features; far more than can be processed by any feature selection tool within a reasonable period of time. Additionally, it is important to identify the terms that appear in most of the files in order to avoid vectors that contain many zeros. Thus, we first extracted the 1,000 features (i.e., OpCode *n*-grams patterns) with the highest *Document Frequency* values and on which three feature selection methods were later applied.

The three feature selection methods operate according to the *filters approach* [40]. In a filters approach method, a measure is used to quantify the correlation of each feature to the class (malicious or benign) and estimate its expected contribution to the classification task. The feature measure that is used by the feature selection method is independent of any classification algorithm, thus allowing us to compare the performances of the different classification algorithms. We used the *Document Frequency* measure *DF* (the amount of files in which the term appeared), *Gain Ratio* (GR) [40] and *Fisher Score* (FS) [41]. Based on each feature selection measure we selected the top 50, 100, 200 and 300 features.

Using the selected features, we evaluated eight commonly used classification algorithms: Support Vector Machine (SVM) [42], Logistic Regression (LR) [43], Random Forest (RF) [44], Artificial Neural Networks (ANN) [45], Decision Trees (DT) [46], Naïve Bayes (NB) [47], and their boosted versions, BDT and BNB [48]. We used the WEKA implementation of these methods [49].

4 Evaluation

4.1 Research Questions

We set out to evaluate the use of OpCodes patterns for the purpose of unknown malicious code detection through three main experiments. When designing these experiments our objective was to investigate the usage of OpCode for unknown malcode detection while considering various strategies and settings of the framework. We summarize the research goals in six questions:

1. Which *term-representation* is better: *TF* or *TFIDF*?

2. Which OpCode *n-gram* size is the best: *1, 2, 3, 4, 5* or *6*? or a combination of OpCode *n*-gram sizes?
3. Which *top-selection* (number of features) is the best: *50*, *100*, *200* or *300* and which *features selection* method: *DF*, *FS* and *GR* is superior?
4. Which *classifier* is the best: SVM, LR, RF, ANN, DT, BDT, NB or BNB?
5. What is the best *Malicious File Percentage* (*MFP*) in the training set for varying MFP in the test set?
6. How often should a classifier be trained with recent malicious files in order to improve the detection accuracy of new malicious files?

To answer the above questions we first performed a wide set of experiments to identify the best term representation, *n*-gram size, top-selection and feature selection method. After determining the optimal settings when using the OpCode representation, we compared the achieved accuracy to the byte *n*-gram representation used in [12]. In the second experiment we investigated the imbalance problem to determine the optimal settings of the training set for each classifier in varying "real-life" conditions. Finally, in the third experiment, we performed a chronological evaluation to determine how well a classifier, which was trained on past examples, can detect new malicious file and to investigate the importance and need in updating the training set frequently.

For evaluation purposes, we used the *True Positive Rate (TPR)* measure, which is the number of *positive* instances classified correctly, *False Positive Rate (FPR)*, which is the number of *negative* instances misclassified, and the *Total Accuracy*, which measures the number of absolutely correctly classified instances, either positive or negative, divided by the entire number of instances. For the imbalance analysis, where the accuracy measure can sometimes be misleading, we also computed the G-Means measure. This measure, which is often used in imbalance dataset evaluation studies, is a metric that combines both the sensitivity and specificity by calculating their geometric mean.

5 Experiments and Results

5.1 Experiment 1 - evaluate OpCode *n*-gram representations settings

In the first experiment we aimed to answer the first four research questions presented in section 4.1. In accordance to these questions, we wanted to identify the best settings of the classification framework which is determined by a combination of: (1) the term-representation (TF or TFIDF); (2) the OpCode *n*-gram size (1, 2, 3, 4, 5 or 6); (3) the top-selection of features (50, 100, 200 or 300); (4) the feature selection method (DF, FS or GR); and (5) the classifier (SVM, LR, RF, ANN, DT, BDT, NB or BNB). We designed a wide and comprehensive set of evaluation runs, including all the combinations of the optional settings for each of the aspects, amounting to 1,152 runs in a 5-fold cross validation format for all eight classifiers. The files in the test-set were not in the training set, presenting unknown files to the classifier. In this experiment, the *Malicious File Percentage* (MFP) in the training and test sets was set according to the natural proportions in the file-set at approximately 22%.

5.1.1 Feature representation vs. n-grams

We first wanted to find the best terms representation (i.e., TF or TFIDF). Figure 2 presents the mean TPR, FPR, accuracy and G-Mean of the combinations of the term

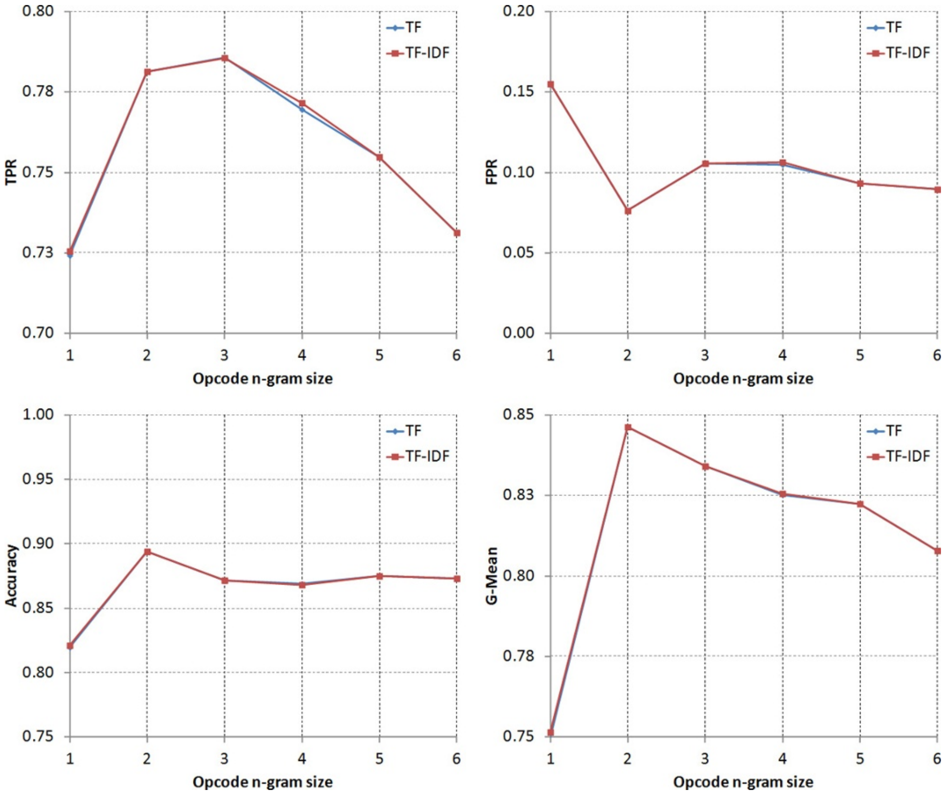


Figure 2 The mean TPR, FPR, accuracy and G-Mean for each term representation (TF and TFIDF) as a function of the OpCode *n*-gram size. While the mean TPRs, FPRs, accuracies and G-Means of the TF and TFIDF were quite identical, the mean accuracy and G-Mean of the 2-gram outperforms all the other *n*-grams with the lowest FPR.

representation and *n*-grams size. The mean TPRs, FPRs, accuracies and G-Means of the TF and the TFIDF were quite identical, which is good because maintaining the TFIDF requires additional computational efforts each time a malcode or benign files are added to the collection. This can be explained by the fact that for each *n*-gram size, the top 1,000 OpCode *n*-grams, having the highest Document Frequency (DF) value, were selected. This was done in order to avoid problems related to sparse data (i.e., vectors that contain many zeros). Consequently, the selected OpCode *n*-grams appear in both sets and therefore eliminate the IDF factor in the TF-IDF measure.

Following this observation we opted to use the TF representation for the rest of our experiments.

Interestingly, the best *n*-gram size of OpCodes was the 2-gram with the highest accuracy and G-Mean values and the lowest FPR (and with TPR similar but slightly lower from the 3-gram). This signifies that the sequence of two OpCodes is more representative than single OpCodes, however, longer grams decreased the accuracy. This observation can be explained by the fact that longer OpCode *n*-grams indicates larger vocabulary (since there are more combinations of the *n*-grams), yet on the other hand, a large number of *n*-grams results in fewer appearances in many files, thus creating sparse vectors. As a case in point, we extracted 443,730 3-grams and 1,769,641 4-grams. In such cases, where many of the vectors are sparse, the detection accuracy will be decreased.

5.1.2 Feature Selections and Top Selections

To identify the best feature selection method and the top number of features, we calculated the mean TPR, FPR, accuracy and G-Mean of each method, as shown in Figure 3. Generally, DF outperformed on all sizes of top features, while FS performed very well, especially when fewer top features were used (top 50 and top 100). Moreover, the DF and FS performance was more stable for varying numbers of top feature in terms accuracy and G-Mean. The DF is a simple feature selection method which favors features which appear in most of the files. This can be explained by its criterion, which has an advantage for fewer features. In other methods, the lack of appearances in many files might create zeroed vectors and might consequently lead to a lower accuracy level.

5.1.3 Classifiers

Figure 4 depicts the mean TPR, FPR, accuracy and G-Mean for each classifier as a function of the OpCode *n*-gram size using the TF representation. The performance of both the Naïve Bayes and the Boosted Naïve Bayes was the worst for all the *n*-gram sizes, having the lowest mean TPR, accuracy and G-Mean, and highest mean FPR. The remaining the classifiers performed very well, having the Random Forest, Boosted Decision Trees and Decision Trees outperforming. The mean accuracy, TPR and G-Mean of the 2-gram outperforms all the other *n*-grams with the lowest mean FPR for all classifiers, but not significantly.

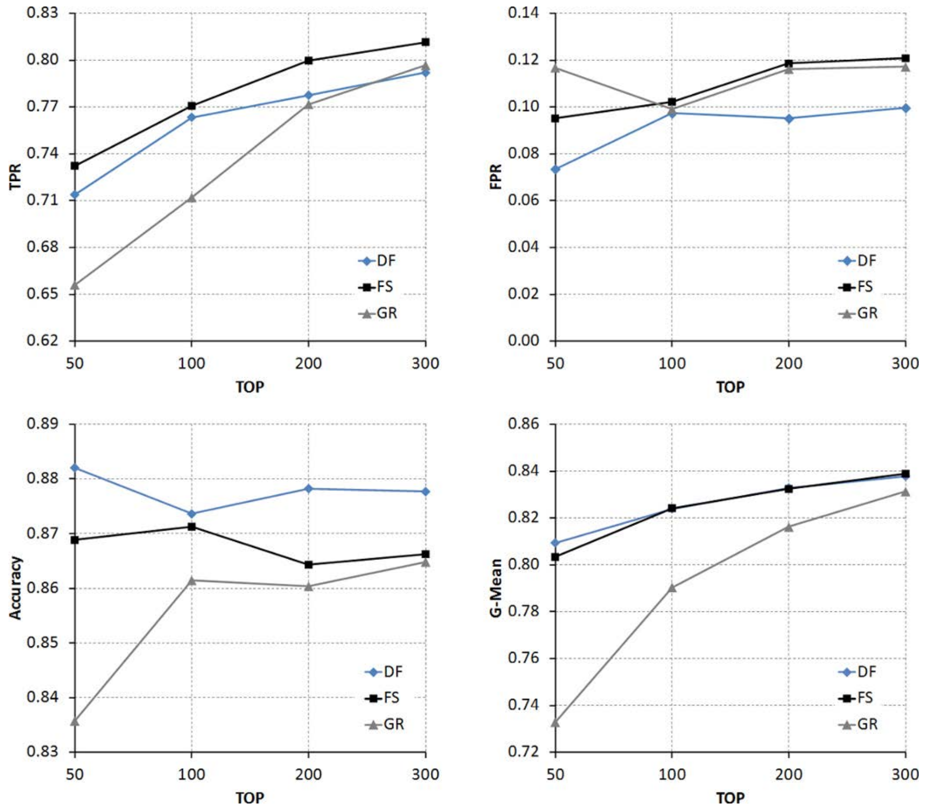


Figure 3 The mean TPR, FPR, accuracy and G-Mean of the evaluated feature selection methods (Document Frequency, Fisher Score, Gain Ratio) as a function of the number of top features (50, 100, 200 and 300). DF was accurate for all sizes of top features. FS performed very well, especially when fewer features were used (top 50 and top 100).

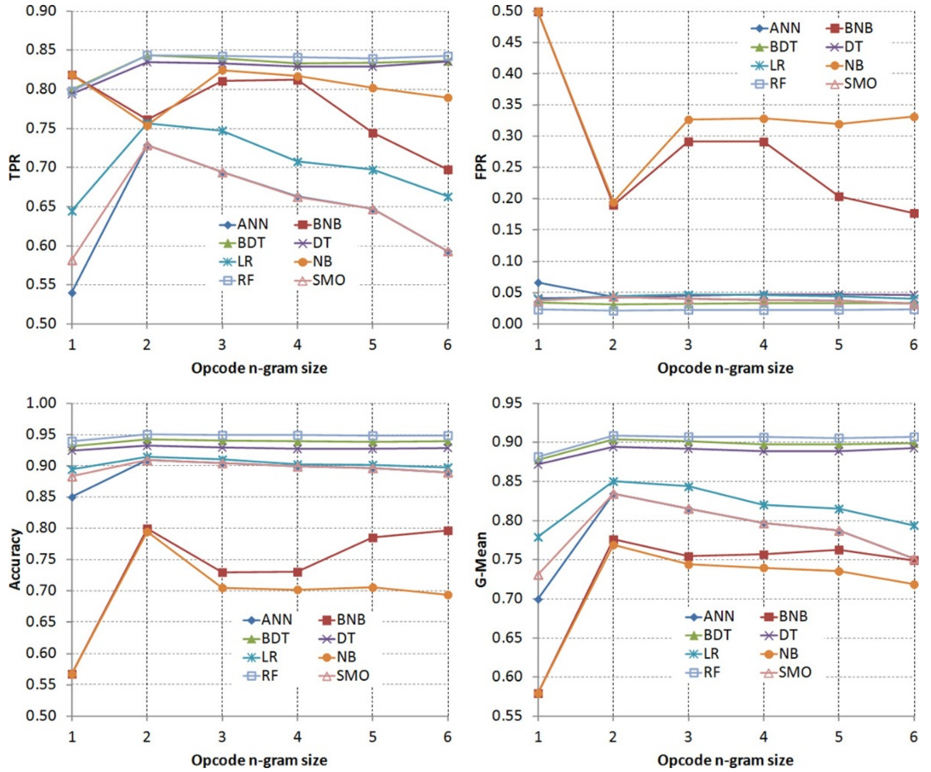


Figure 4 The mean TPR, FPR, accuracy and G-Mean for each classifier (using TF representation) as a function of the OpCode *n*-gram size. The performance of both the Naïve Bayes and the Boosted Naïve Bayes was worst for all *n*-gram sizes having the lowest mean TPR, accuracy and G-Mean and highest mean FPR. The mean accuracy, TPR and G-Mean of the 2-gram outperforms all the other *n*-grams with the lowest mean FPR for all classifiers.

Classifiers differ in performance within different domains and the best fitted classifier can often be identified by experimentation. From the results we conclude that for this problem domain, complex classifiers, such as the ensemble Random Forest algorithm [44] which induces many decision trees and then combines the results of all trees, and the boosted decision tree [48] generate a more accurate classifier.

Additionally, in order to compare the classifiers' performance, we selected the settings which had the highest mean accuracy level over all the classifiers. In order to find the best settings for all the classifiers, we calculated the mean FPR, TPR, accuracy and G-Mean for each setting that is defined by the: (1) *n*-gram size; (2) feature representation; (3) feature selection method; and (4) the number of top features. Table 1 depicts the top five settings with the highest mean accuracy level (averaged over all the

Table 1 The top five settings with the highest mean accuracy over all the classifiers.

n-gram size	Representation	Feature selection	Top features	FPR	TPR	Accuracy	G-Mean
2	TF	DF	300	0.045	0.744	0.911	0.840
2	TFIDF	DF	300	0.045	0.744	0.911	0.840
2	TF	DF	100	0.053	0.754	0.907	0.845
2	TFIDF	DF	100	0.053	0.754	0.907	0.845
2	TF	DF	200	0.047	0.729	0.906	0.830

classifiers). *The outperforming setting was the: 2-gram, TF, using 300 features selected by the DF measure.*

The results of each classifier when using the best mean settings (i.e., -gram, TF, using 300 features selected by the DF measure), including the accuracy, TPR, FPR and G-Mean are presented in Table 2. In addition, the optimal setting of each classifier is presented, as well as the resulted accuracy for the optimal setting, and the difference compared to the accuracy achieved with the best averaged setting. The comparisons show that for all classifiers, excluding the NB and BNB, the best averaged setting yields similar performance.

The graphs in Figure 5 depict the TPR, FPR, accuracy and G-Mean of each classifier when comparing the best averaged settings (2-gram, TF representation, using 300 features selected by the DF measure) with the classifier's optimal settings. The graphs show that the Random Forest and Boosted Decision Tree yielded the highest accuracy and lowest FPR. Naïve Bayes and Boosted Naïve Bayes performed poorly and thus we omitted them from the following experiments.

In the following two experiments we used the best six classifiers (RF, DT, BDT, LR, ANN, SVM) when trained on the best averaged settings (2-gram, TF representation, 300 top features selected by the DF measure).

5.1.4 Varying OpCode n-grams sizes

In this analysis we set out to answer the second part of research question 2 and to understand whether a combination of different sizes of OpCode *n*-grams, as features in the classification task, may result in better detection performance. For this we used three OpCode *n*-grams sets on which the three feature selection methods were applied with four top-selections (50, 100, 200 or 300):

- ***Constant n-gram size*** This option refers to the 6 OpCode *n*-grams sets that were used in the previous experiments, in which the *n*-grams in each set are of the same size (1, 2, 3, 4, 5 and 6).

- ***Top 1,800 over all n-gram sizes*** In this set, *all* OpCode *n*-grams, *of all sizes*, were sorted according to their DF value. Then, the first 1,800 *n*-grams with the top DF score were selected. Feature selection was applied on the collection of 1,800 *n*-grams patterns.

- ***Top 300 for each n-gram size*** In this set, for each OpCode *n*-gram size (1- to 6-gram), the first 300 *n*-grams with the top DF score were selected (i.e., total of 1,800 *n*-grams). Feature selection was applied on the collection of 1,800 *n*-grams patterns.

Table 2 The accuracy, FPR, TPR and G-Mean of each classifier when using the best mean settings (i .e., 2-gram, TF representation, top 300 features selected by the DF measure).

Classifier	Best averaged settings				Classifier optimal settings		Difference in accuracy
	Accuracy	TPR	FPR	G-Means	Optimal settings	Accuracy	
RF	95.146	0.843	0.020	0.909	6-gram; TF; GR; top300	95.375	0.229
BDT	94.436	0.850	0.031	0.908	2-gram; TF; FS; top300	94.649	0.213
DT	93.009	0.835	0.045	0.893	2-gram; TF; FS; top300	93.741	0.732
LR	92.965	0.819	0.041	0.886	2-gram; TF; DF; top300	92.965	0.000
ANN	92.136	0.785	0.043	0.867	3-gram; TF; DF; top300	92.199	0.063
SVM	92.136	0.785	0.043	0.867	3-gram; TF; DF; top300	92.199	0.063
NB	84.537	0.517	0.068	0.694	5-gram; TF; FS; top50	85.802	1.265
BNB	84.537	0.517	0.068	0.694	2-gram; TF; DF; top50	86.683	2.146

The table also depicts the optimal settings of each classifier and the difference in accuracy with the averaged settings.

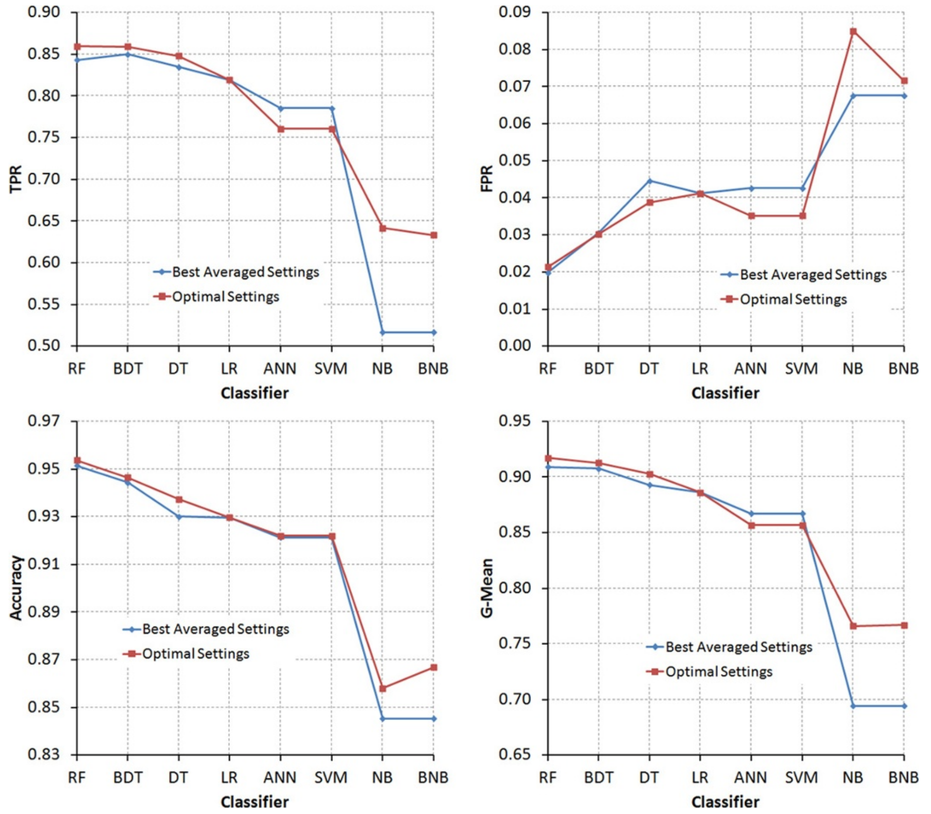


Figure 5 **TPR, FPR, Accuracy and G-Mean of each classifier when comparing the best averaged settings (i.e., 2-gram, TF representation, 300 features selected by the DF measure) and the classifier's optimal settings**.

The distribution of *n*-grams sizes for the two *n*-grams sets that consist of varying n-gram sizes is presented in Table 3. From the table we can see, as expected, that the DF feature selection method favors short *n*-grams which appear in a larger number of files. In addition, we can see that in most cases, FS and GR tend to select *n*-grams of size 2, 3 and 4 which we conclude to be more informative and with a tendency to discriminate better between the malicious and benign classes in the classification task.

In Figure 6 we present the mean TPR, FPR, accuracy and G-Mean of each classifier when using the best mean settings obtained for each of the three OpCode *n*-grams patterns sets:

- ***Constant n-gram size*** 2-gram, TF representation, 300 features selected by the DF measure (as presented in section 5.1.3) - denoted by [2gram;TF;Top300;DF].
- ***Top 1,800 over all n-gram sizes*** TF representation, 300 features selected by the GR measure - denoted by [Top1800All;TF;Top300;GR].
- ***Top 300 for each n-gram size*** TF representation, 300 features selected by the GR measure - denoted by [Top300Each; TF;Top300;GR].

The results show that using various sizes of OpCode *n*-grams patterns does not improve the detection performance and in fact for most classifiers, the performance accuracy was deteriorated. We therefore use the constant *n*-gram size sets for the next experiments.

Table 3 Distribution of *n*-gram sizes, chosen by each feature selection method, for the two *n*-grams sets that consist of varying *n*-grams sizes.

		Top 1800 over all *n*-grams						Top 300 for each *n*-gram size					
Feature selection	**Top features**	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**
DF	50	0.3	**0.42**	0.24	0.04	0	0	**1**	0	0	0	0	0
DF	100	0.17	**0.39**	0.28	0.13	0.03	0	**1**	0	0	0	0	0
DF	200	0.11	0.27	**0.36**	0.21	0.06	0.01	**1**	0	0	0	0	0
DF	300	0.08	0.25	**0.34**	0.23	0.09	0.01	**1**	0	0	0	0	0
FS	50	0.06	0.18	**0.38**	0.16	0.1	0.12	**0.54**	0.08	0.16	0.1	0.06	0.06
FS	100	0.03	0.17	**0.36**	0.26	0.12	0.06	**0.43**	0.12	0.19	0.1	0.11	0.05
FS	200	0.02	0.13	**0.33**	0.27	0.18	0.08	**0.35**	0.14	0.2	0.14	0.12	0.08
FS	300	0.02	0.1	**0.3**	**0.3**	0.21	0.08	**0.3**	0.15	0.2	0.15	0.11	0.08
GR	50	0	0.16	**0.36**	0.22	0.22	0.04	0.06	0.2	**0.36**	0.14	0.1	0.14
GR	100	0	0.12	**0.33**	0.29	0.17	0.09	0.03	0.19	**0.36**	0.23	0.11	0.08
GR	200	0.01	0.13	**0.31**	**0.32**	0.16	0.09	0.04	0.15	**0.31**	0.22	0.17	0.12
GR	300	0.01	0.17	0.3	0.28	0.15	0.08	0.04	0.14	0.28	0.24	0.18	0.13

5.2 Experiment 2 - The imbalance problem

In our second experiment, we addressed our 5th research question in order to find the best Malicious File Percentage (MFP) among the training-set files for varying MFP in the test-set files, and more specifically, for low MFP in the test-set (10-15%), which resembles a real-life scenario. We created five levels of Malicious Files Percentage (MFP) in the training set (5, 10, 15, 30, and 50%). For example, when referring to 15%, we assert that 15% of the files in the training set were malicious and 85% were benign. The test-set represents the real-life situation while the training set represents the set-up of the classifier, which is controlled. We had the same MFP levels for the test-sets as well. Thus, we ran all the product combinations of five training sets and five test-sets for a total of 25 runs for each classifier. The dataset was divided into two parts. Each time the training set was chosen from one part and the test set was chosen from the other part, thus forming a 2-fold cross validation-like evaluation to render the results more significant.

Training-Set Malware Percentage

Figure 7 presents the mean accuracy, FPR, TPR, and G-Mean (i.e., averaged over all the MFP levels in the test-sets) of each classifier and for each training MFP level. It is shown that all classifiers, excluding ANN, had a similar trend and perform better when using MFP of 15% - 30% in the training set, while Random Forest and Boosted Decision Tree outperformed all other classifiers exceeding 94.5% accuracy and 87.1% TPR, while keeping the FPR bellow 4%. The ANN performance was generally low and dropped significantly for 5%, 15% and 50% MFP in the training set. Additionally, it is shown that the FPR grows for all classifiers with the increasing of the MFP in the training set. This can be explained by the fact that for training sets with higher MFP most of the test sets are have a lower MFP, which in turn results in higher FPR. This in fact emphasizes the imbalance problem.

10% Malware Percentage in the Test-Set

we consider the 10% MFP level in the test-set to be a reasonable real-life scenario, as mentioned in the introduction. Figure 8 presents the mean accuracy, FPR, TPR and G-Mean for a 2-fold cross validation experiment for each MFP in the training set and

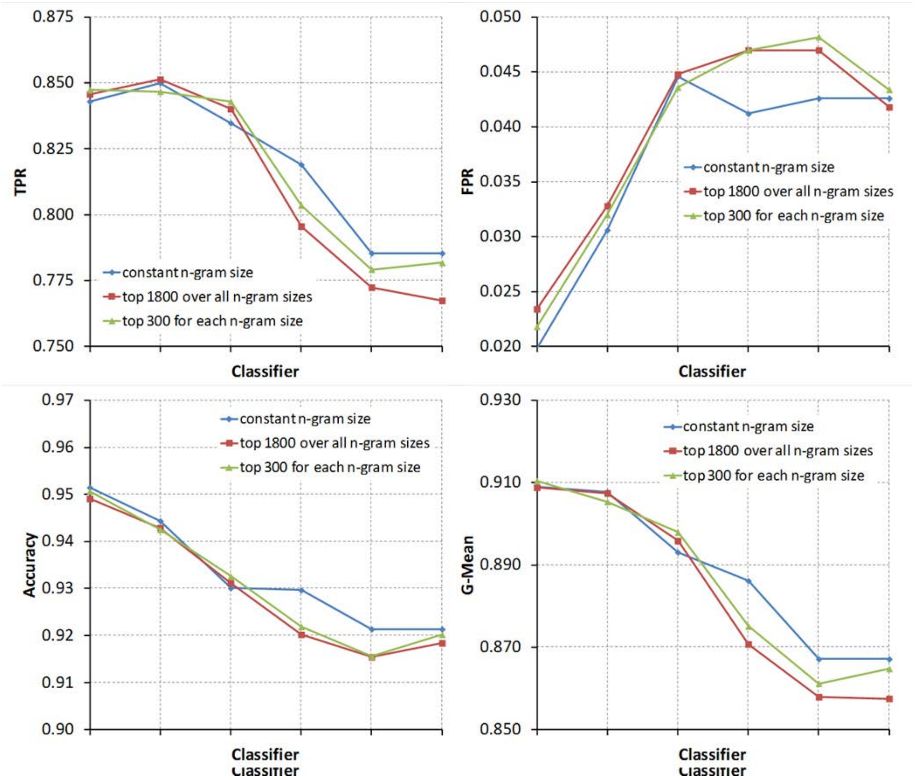


Figure 6 Mean TPR, FPR, accuracy and G-Mean of each classifier when using the best mean settings obtained for each of the three *n*-grams sets: [2gram;TF;Top300;DF], [Top1800All;TF; Top300;GR] and [Top300Each;TF;Top300;GR].

with a fixed level of 10% MFP in the test-set. These results are quite similar in their magnitude to the results in Figure 7, although here the performance level was higher. For the RF and BDT, the highest performance level was in 10% and 15% of MFP in the training set, which is more similar to the MFP in the test-set.

Relations among MFPs in Training and Test-sets

Further to our results from the training-set point of view (Figures 7 and 8), we present a detailed description of the accuracy, TPR and FPR for the MFP levels in the two sets in a 3-dimensional presentation for each classifier (the graphs of the two best classifiers, RF and BDT, are presented in Figure 9; the graphs of the rest of the classifiers are provided in Additional file 1). Interestingly, a stable state is observed in the accuracy measure for any MFP level. In addition, we can see that for a given MFP in the *training set*, the TPR and the FPR of the classifiers are stable for any MFP level in the test set. This observation, which emphasizes the imbalance problem, signifies that in order to achieve a desired TPR and FPR, only the training set can be considered and selecting the proper MFP in the training set will ensure the desired TPR and FPR for any MFP in the test set.

When comparing these results with the results of the byte *n*-grams patterns experiments in [12] we notice that in terms of accuracy, the byte *n*-grams classifiers are more sensitive to varying MFP levels in the training and test-sets. In particular, the DT and BDT classifiers behaved optimally when the MFP levels in the training-set and test-set were similar. This observation may indicate an advantage of the OpCode *n*-grams representation as being less sensitive to the levels of MFP in the two sets, or

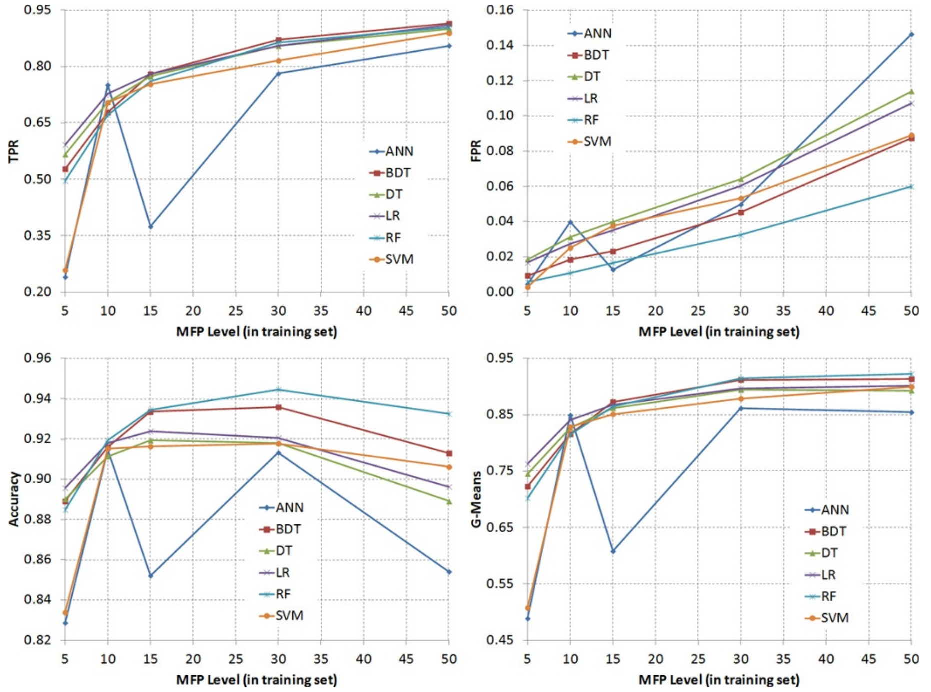


Figure 7 Mean accuracy, FPR, TRP and G-Mean (over all the MFP levels in the test-sets) for each MFP in the training set. RF and BDT out-performed across the varying MFPs.

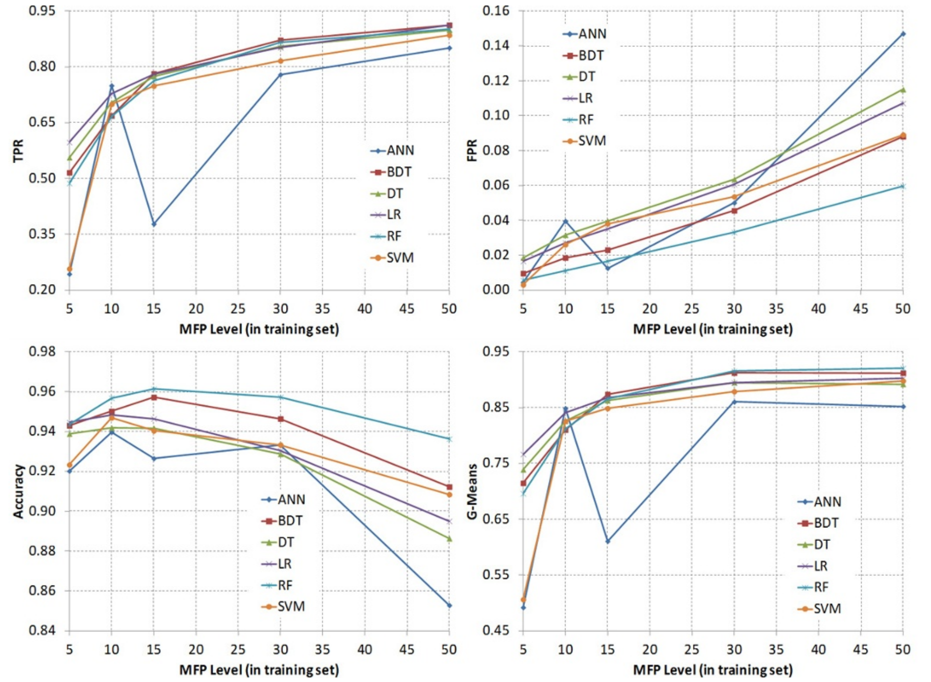
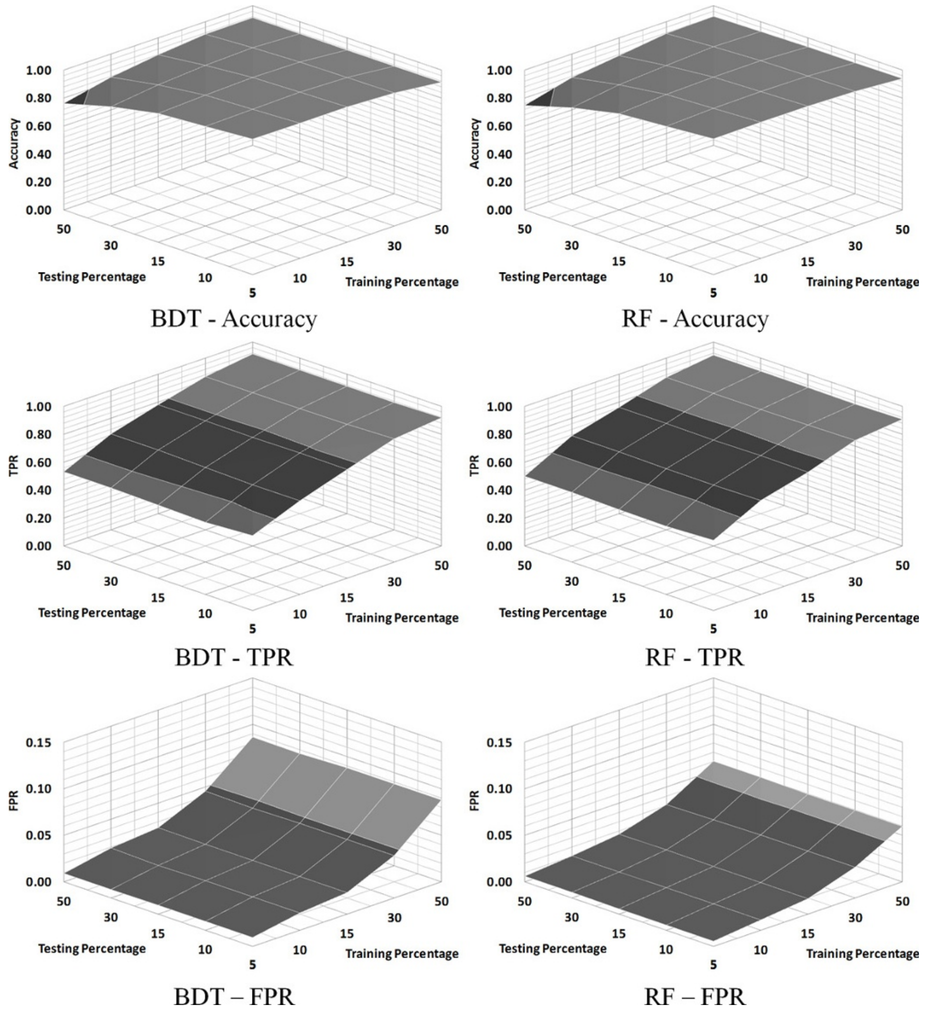


Figure 8 The mean accuracy, FPR, TPR and G-Mean for 10% MFP in the test-set, for each MFP in the training set.



BDT - Accuracy

RF - Accuracy

BDT - TPR

RF - TPR

BDT – FPR

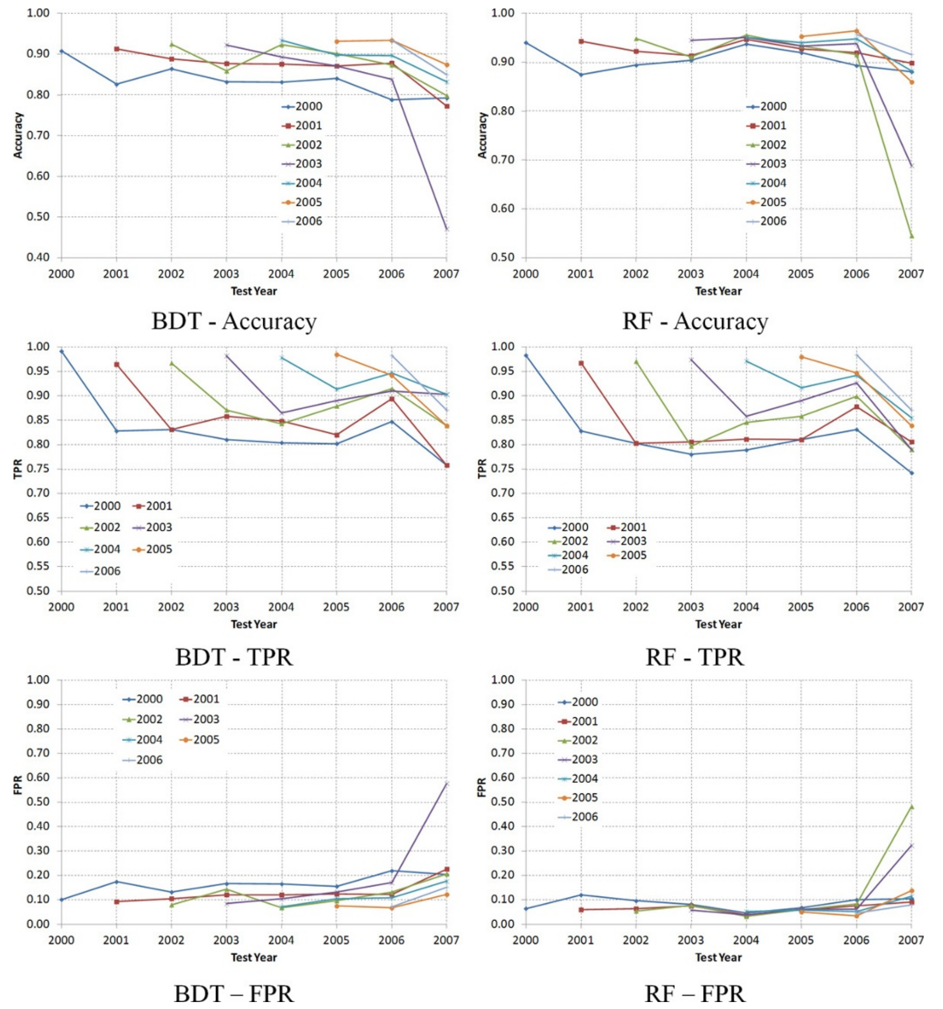
RF – FPR

Figure 9 The mean Accuracy, TPR and FPR for different MFP levels in the training and test sets for the two best classifiers BDT and RF (the graphs for the rest of the classifiers are provided in Additional file 1).

more specifically in the test sets which represent the changes of proportions in real life conditions.

5.3 Experiment 3 - Chronological Evaluation

In the third experiment, we addressed our 6th research question in order to understand the need in updating the training set. The question asks how important it is to update the repository of malicious and benign files and whether, for specific years, the files were more contributive to the accuracy when introduced in the training set or in the test set. In order to answer these questions we divided the entire test collection into years from 2000 to 2007, in which the files were created. We had 7 training sets, in which training set k included samples from the year 2000 till year 200[k] (where k = 0,1,2..,6). Each training set k was evaluated separately on each following year from 200[k+1] till 2007. Clearly, the files in the test were not present in the training set. Figure 10 presents the results with a 50% MFP in the training set and10% MFP in the testing set for the two best classifiers BDT and RF (the graphs for the rest of the classifiers are provided in Additional file 2). Out of the ANN classifier, all other classifiers observed similar



BDT - Accuracy

RF - Accuracy

BDT - TPR

RF - TPR

BDT – FPR

RF – FPR

Figure 10 The results (accuracy, TPR and FPR) for with a 50% MFP on the training set and 10% MFP on the test set for the two best classifiers BDT and RF (the rest of the classifiers are presented in Additional file 2).

behavior in which higher TPR and lower FPR were achieved when training on newer files. In fact, in all of the cases, the TPR was above 0.95 and FPR approximately 0.1 when training the models on a yearly basis. Finally, for all classifiers, when testing on 2007 examples, a significant decrease in the accuracy was observed; a fact that might indicate that new types of malware were released during 2007.

6. Discussion and Conclusions

In this study we used OpCode *n*-gram patterns generated by disassembling the inspected executable files to extract features from the inspected files. OpCode *n*-grams are used as features during the classification process with the aim of identifying unknown malicious code. We performed an extensive evaluation using a test collection comprising more than 30,000 files. The evaluation consisted of three experiments.

In the first experiment, we found that the TFIDF representation has no added value over the TF representation, which is not the case in many information retrieval applications. This is very important since using the TFIDF representation introduces additional computational challenges in the maintenance of the collection when it is updated. In order to reduce the number of OpCode *n*-gram features, which ranges from thousands to millions, we used the DF measure to select the top 1,000 features and tested three feature selection methods. The 2-gram OpCodes outperformed the others and the DF was the best feature selection method. We also evaluated the performance of classifiers when using a constant size of OpCode *n*-grams versus using varying sizes of *n*-grams. The result of this experiment showed no improvement when using OpCode *n*-grams of different sizes.

In the second experiment, we investigated the relationship between the Malicious File Percentage (MFP) in the test-set, which represents real-life scenario, and in the training set, which is used for training the classifier. In this experiment, we found that there are classifiers which are relatively non-reactive to changes in the MFP level of the test-set. In general, this indicates that in order to achieve a desired TPR and FPR, only the training set can be considered and selecting the proper MFP in the training set will ensure the desired TPR and FPR for any MFP in the test set.

In the third experiment we wanted to determine the importance of updating the training set over time. Thus, we divided the test collection into years and evaluated training sets of selected years on the next years. Evaluation results show that an update in the training set is needed. Using 10% malicious files in the training set showed a clear trend in which the performance improves when the training set is updated on a yearly basis.

Based on the reported experiments and results, we suggest that when setting up a classifier for real-life purposes, one should first use the OpCode representation and, if the disassemble of the file is not feasible, use the byte representation [12], which appears to be less accurate. In addition, one should consider the expected proportion of malicious files in the stream of data. Seeing as we assume that in most real-life scenarios low proportions of malicious files are present, training sets should be designed accordingly.

In future work we plan to experiment with cost-sensitive classification in which the costs of the two types of errors (i.e., missing a malicious file and false alarm) are not equal. We believe that the application of cost-sensitive classification depends on the goals to be achieved, and accordingly the cost of having a misclassification of each type. Having experience in using this approach in real life setting, we can give two general examples of such applications. The first example pertains to for anti-virus companies that need to analyze dozens of thousands of maliciously suspected (or unknown) files, including benign files, every day. In such an application the goal is to perform an initial filtering to reduce the amount of files to investigate manually. Thus, having a relatively high false-positive is reasonable in order to decrease the probability of missing an unknown malicious file. Another application is as an anti-virus. In this case we would like to decrease the probability of false-negative, which will result in quarantining, deleting, or blocking of a legitimate file. For both scenarios it is difficult to assign the costs for the two errors (note that each type of malware can be assigned with a different cost level based on the damage it causes) and therefore in this paper we focus on exploring and identifying the settings and classifiers that can classify the files as accurately as possible, leaving the cost-sensitive analysis for future work.

Additional material

Additional file 1: Relations among MFPs in training and test-sets: accuracy, TPR and FPR for the MFP levels in the two sets in a 3-dimensional presentation. Detailed description of the accuracy, TPR and FPR for the malicious file percentage levels in the two sets in a 3-dimensional presentation for each classifier.

Additional file 2: Chronological evaluation: accuracy, TPR and FPR with a 50% MFP in the training set and 10% MFP in the testing set for all classifiers. Detailed description of the accuracy, TPR and FPR with a 50% MFP in the training set and 10% MFP in the testing set for all classifiers.

Author details
[1]Deutsche Telekom Laboratories, Ben-Gurion University, Be'er Sheva, 84105, Israel [2]Department of Information Systems Engineering, Ben-Gurion University, Be'er Sheva, 84105, Israel [3]Department of Computer Science, Ben-Gurion University, Be'er Sheva, 84105, Israel

Authors' contributions

RM and AS conceived of the study, studied the research domain, participated in the design of the study, performed the analysis of the results, and drafted the manuscript. CF carried out the data collection and experiments. YE and SD participated in the design of the study and its coordination. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Received: 12 July 2011 Accepted: 27 February 2012 Published: 27 February 2012

References

1. Shabtai A, Moskovitch R, Elovici Y, Glezer C: Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Technical Report* 2009, **14(1)**:1-34.
2. Griffin K, Schneider S, Hu X, Chiueh T: Automatic generation of string signatures for malware detection. *12th International Symposium on Recent Advances in Intrusion Detection* Heidelberg: Springer; 2009, 101-120.
3. Rieck K, Holz T, Düssel P, Laskov P: Learning and classification of malware behavior. *Conference on Detection of Intrusions and Malware & Vulnerability Assessment* Heidelberg: Springer; 2008, 108-125.
4. Bailey M, Oberheide J, Andersen J, Mao ZM, Jahanian F, Nazario J: Automated classification and analysis of Internet malware. *12th International Symposium on Recent Advances in Intrusion Detection* Heidelberg: Springer; 2007, 178-197.
5. Lee W, Stolfo SJ: A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security* 2000, **3(4)**:227-261.
6. Moskovitch R, Elovici Y, Rokach L: Detection of unknown computer worms based on behavioral classification of the host. *Computational Statistics and Data Analysis* 2008, **52(9)**:4544-4566.
7. Jacob G, Debar H, Filiol E: Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology* 2008, **4**:251-266.
8. Shabtai A, Potashnik D, Fledel Y, Moskovitch R, Elovici E: Monitoring, analysis and filtering system for purifying network traffic of known and unknown malicious content. *Security and Communication Networks* 2010, DOI: 10.1002/sec.229.
9. Moser A, Kruegel C, Kirda E: Limits of static analysis for malware detection. *Annual Computer Security Applications Conference, IEEE Computer Society* 2007, 421-430.
10. Menahem E, Shabtai A, Rokach L, Elovici Y: Improving malware detection by applying multi-inducer ensemble. *Computational Statistics and Data Analysis* 2008, **53(4)**:1483-1494.
11. Moskovitch R, Feher C, Tzachar N, Berger E, Gitelman M, Dolev S, Elovici Y: Unknown malcode detection using OpCode representation. *European Conference on Intelligence and Security Informatics* Heidelberg: Springer; 2008, 204-215.
12. Moskovitch R, Stopel D, Feher C, Nissim N, Japkowicz N, Elovici Y: Unknown malcode detection and the imbalance problem. *Journal in Computer Virology* 2009, **5(4)**:295-308.
13. Abou-Assaleh T, Keselj V, Sweidan R: N-gram based detection of new malicious code. *Proc of the 28th Annual International Computer Software and Applications Conference, IEEE Computer Society* 2004, 41-42.
14. McAfee Study Finds 4% of Search Results Malicious. *Frederick Lane* 2007 [http://www.newsfactor.com/story.xhtml?story_id = 010000CEUEQO].
15. Shin S, Jung J, Balakrishnan H: Malware prevalence in the KaZaA file-sharing network. *Internet Measurement Conference(IMC), ACM Press* 2006, 333-338.
16. Schultz M, Eskin E, Zadok E, Stolfo S: Data mining methods for detection of new malicious executables. *Proc of the IEEE Symposium on Security and Privacy, IEEE Computer Society* 2001, 38.
17. Kolter JZ, Maloof MA: Learning to detect malicious executables in the wild. *Proc of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press* 2006, 470-478.
18. Kolter J, Maloof M: Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research* 2006, **7**:2721-2744.
19. Cai DM, Gokhale M, Theiler J: Comparison of feature selection and classification algorithms in identifying malicious executables. *Computational Statistics and Data Analysis* 2007, **51**:3156-3172.
20. Karim E, Walenstein A, Lakhotia A, Parida L: Malware phylogeny generation using permutations of code. *Journal in Computer Virology* 2005, **1(1-2)**:13-23.

21. Siddiqui M, Wang MC, Lee J: **Data mining methods for malware detection using instruction sequences.** *Artificial Intelligence and Applications* ACTA Press; 2008, 358-363.
22. Bilar D: **Opcodes as predictor for malware.** *International Journal Electronic Security and Digital Forensics* 2007, **1(2)**:156-168.
23. Santos I, Brezo F, Nieves J, Penya YK, Sanz B, Laorden C, Bringas PG: **Idea: Opcode-sequence-based malware detection.** *Proc 2nd International Symposium on Engineering Secure Software and Systems* 2010, 35-42.
24. Kubat M, Matwin S: **Addressing the curse of imbalanced data sets: one-sided sampling.** *Proc of the 14th International Conference on Machine Learning* 1997, 179-186.
25. Chawla NV, Japkowicz N, Kotcz A: **Editorial: Special issue on learning from imbalanced datasets.** *SIGKDD Explorations Newsletter* 2004, **6(1)**:1-6.
26. Japkowicz N, Stephen S: **The class imbalance problem: a systematic study.** *Intelligent Data Analysis Journal* 2002, **6(5)**:429-450.
27. Chawla NV, Bowyer KW, Kegelmeyer WP: **SMOTE: synthetic minority over-sampling technique.** *Journal of Artificial Intelligence Research (JAIR)* 2002, **16**:321-357.
28. Lawrence S, Burns I, Back AD, Tsoi AC, Giles CL: **Neural network classification and unequal prior class probabilities.** In *Tricks of the Trade, Lecture Notes in Computer Science State-of-the-Art Surveys.* Edited by: Orr G, Muller K-R, Cruana R. Springer Verlag; 1998:299-314.
29. Chen C, Liaw A, Breiman L: **Using random forest to learn unbalanced data.** *Technical Report 666* Statistics Department, University of California at Berkeley; 2004.
30. Morik K, Brockhausen P, Joachims T: **Combining statistical learning with a knowledge-based approach - a case study in intensive care monitoring.** *ICML, Morgan Kaufmann Publishers Inc* 1999, 268-277.
31. Weiss GM, Provost F: **Learning when training data are costly: the effect of class distribution on tree induction.** *Journal of Artificial Intelligence Research* 2003, **19**:315-354.
32. Provost F, Fawcett T: **Robust classification systems for imprecise environments.** *Machine Learning* 2001, **42(3)**:203-231.
33. Kubat M, Matwin S: **Machine learning for the detection of oil spills in satellite radar images.** *Machine Learning* 1998, **30**:195-215.
34. Heavens VX:[http://vx.netlux.org].
35. Linn C, Debray S: **Obfuscation of executable code to improve resistance to static disassembly.** *Proc of the 10th ACM conference on Computer and communications security* ACM Press; 2003, 290-299.
36. Dinaburg A, Royal P, Sharif MI, Lee W: **Ether: malware analysis via hardware virtualization extensions.** *ACM Conference on Computer and Communications Security, ACM Press* 2008, 51-62.
37. Perdisci R, Lanzi A, Lee W: **McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables.** *Annual Computer Security Applications Conference, IEEE Computer Society* 2008, 301-310.
38. Royal P, Halpin M, Dagon D, Edmonds R, Lee W: **PolyUnpack: automating the hidden-code extraction of unpack-executing malware.** *Annual Computer Security Applications Conference* IEEE Computer Society; 2006, 289-300.
39. Salton G, Wong A, Yang CS: **A vector space model for automatic indexing.** *Communications of the ACM* 1975, **18**:613-620.
40. Mitchell T: *Machine Learning* McGraw-Hill; 1997.
41. Golub T, Slonim DK, Tamayo P, Huard C, Gaasenbeek M, Mesirov JP, Coller H, Loh ML, Downing JR, Caligiuri MA, Bloomfield CD, Lander ES: **Molecular classification of cancer: class discovery and class prediction by gene expression monitoring.** *Science* 1999, **286**:531-537.
42. Joachims T: **Making large-scale support vector machine learning practical.** In *Advances in Kernel Methods.* Edited by: Scholkopf B, Burges C, Smola AJ. Cambridge, MA: MIT Press; 1999:169-184.
43. Neter J, Kutner MH, Nachtsheim CJ, Wasserman W: *Applied Linear Statistical Models* McGraw-Hill; 1996.
44. Kam HT: **Random Decision Forest.** *Proc of the 3rd International Conference on Document Analysis and Recognition* 1995, 278-282.
45. Bishop C: *Neural Networks for Pattern Recognition* Oxford: Clarendon Press; 1995.
46. Quinlan JR: *C4.5: Programs for Machine Learning* San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc; 1993.
47. Domingos P, Pazzani M: **On the optimality of simple Bayesian classifier under zero-one loss.** *Machine Learning* 1997, **29**:103-130.
48. Freund Y, Schapire RE: **A brief introduction to boosting.** *International Joint Conference on Artificial Intelligence* Morgan Kaufmann Publishers Inc; 1999, 1401-1406.
49. Witten IH, Frank E: *Data Mining: Practical Machine Learning Tools and Techniques.* 2 edition. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc; 2005.

doi:10.1186/2190-8532-1-1
Cite this article as: Shabtai *et al*.: **Detecting unknown malicious code by applying classification techniques on OpCode patterns.** *Security Informatics* 2012 **1**:1.