available at www.sciencedirect.com

ScienceDirect

www.compseconline.com/publications/prodinf.htm

Information Security Technical Report

Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey

Asaf Shabtai, Robert Moskovitch, Yuval Elovici, Chanan Glezer*

Deutsche Telekom Laboratories at Ben-Gurion University, Ben-Gurion University, Be'er Sheva 84105, Israel

ABSTRACT

This research synthesizes a taxonomy for classifying detection methods of new malicious code by Machine Learning (ML) methods based on static features extracted from executables. The taxonomy is then operationalized to classify research on this topic and pinpoint critical open research issues in light of emerging threats. The article addresses various facets of the detection challenge, including: file representation and feature selection methods, classification algorithms, weighting ensembles, as well as the imbalance problem, active learning, and chronological evaluation. From the survey we conclude that a framework for detecting new malicious code in executable files can be designed to achieve very high accuracy while maintaining low false positives (i.e. misclassifying benign files as malicious). The framework should include training of multiple classifiers on various types of features (mainly OpCode and byte n-grams and Portable Executable Features), applying weighting algorithm on the classification results of the individual classifiers, as well as an active learning mechanism to maintain high detection accuracy. The training of classifiers should also consider the imbalance problem by generating classifiers that will perform accurately in a real-life situation where the percentage of malicious files among all files is estimated to be approximately 10%.

1. Introduction

Modern computer and communication infrastructures are highly susceptible to various types of attacks. A common way of launching these attacks is using malware (malicious software) such as worm, virus, Trojan or spyware (Kienzle and Elder, 2003; Heidari, 2004). Propagation of malware might result in havoc to private users, commercial companies and governments. As a case in point, a single instance of malware installed on a computer which belongs to a computer network, can result in the loss, or unauthorized utilization or modification, of large amounts of data and cause users to question the reliability of all the information on the network. High-speed digital communication channels enable malware to propagate and infect hosts very quickly and so it is essential to detect and eliminate new (unknown) malware in a prompt manner.

Anti-virus technology, a key player in tackling malware nowadays, is primarily based on two complementary approaches. *Signature-based* methods rely on the identification of unique strings in the binary code. Whenever a new type of malware is unleashed, anti-virus vendors need to catch an instance of the new malware, analyze it, create a new signature and update their clients. During the period between the appearance of a new (unknown) malware and the update of the signature-base of the anti-virus clients, millions of

* *Corresponding author.*
E-mail addresses: shabtaia@bgu.ac.il (A. Shabtai), robertmo@bgu.ac.il (R. Moskovitch), elovici@bgu.ac.il (Y. Elovici), chanan@bgu.ac.il (C. Glezer).

1363-4127/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.istr.2009.03.003

computers are vulnerable to the new malware. Therefore, while being very precise, this approach is useless against previously unobserved malware (Christodorescu and Jha, 2004; White, 1999). The second approach involves *heuristic-based* methods, which are based on rules determined by experts, which define a malicious behavior, or a benign behavior, in order to enable the detection of unknown malcodes (Jacob et al., 2008; Gryaznov, 1999). However, besides the fact that these methods can be bypassed by viruses, their main drawback is that, by definition, they can only detect the presence of a malcode after the infected program has been executed.

Generalizing the detection methods so it can detect unknown malware before its execution is therefore, crucial in facilitating effective defense. Recently, Machine Learning methods, such as classification algorithms were employed to automate and extend the idea of heuristic-based methods. In these methods, classifiers are applied to learn patterns in the binary code files in order to classify new (unknown) files. A classifier is a rule-set that is learnt from a given training set, which includes examples of both malicious and benign files. Our research corroborates earlier studies indicating that ML methods yield very accurate classification results and are thus effective for detecting new malware.

In this survey paper we first synthesize a taxonomy for classifying detection methods of new malicious code by Machine Learning (ML) methods utilizing static features extracted from executables. The taxonomy is then operationalized to classify research on this topic and pinpoint critical open research issues in the light of emerging threats.

2. Employing machine learning for detecting new malware – a taxonomy

Classifying unknown malware using Machine Learning (ML) instruments was inspired by the text categorization problem. In text categorization, a textual file is represented as bag of word, based on Salton's vector space model (Salton et al., 1975). After parsing the text and extracting the words it includes, a *vocabulary* of the entire word collection is constructed. The vocabulary is maintained as a vector of terms extracted from the entire set of documents. Each term in the vocabulary can be characterized by its frequency in the entire collection, often called *Document Frequency* (DF), which is later used for the term weighting. Each of the terms may appear zero or more times in a given document and at least once in one of the documents. For each document a vector of terms (matching the vocabulary) is created such that each cell in the vector represents the *Term Frequency* (TF) in the corresponding document. Equation (1) depicts the definition of a normalized TF with a range [0–1], so that each term's frequency is divided by the frequency of the most frequent term in the document. An extended representation is the *TF Inverse Document Frequency* (TFIDF), which combines the frequency of a term in the document (TF) and its frequency in the documents' collection, denoted by *Document Frequency* (DF), as shown in Equation (2), in which 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.

$$\mathrm{TF} = \frac{\text{term frequency}}{\max(\text{term frequency in document})} \tag{1}$$

$$\mathrm{TFIDF} = \mathrm{TF} \times \log\left(\frac{N}{\mathrm{DF}}\right) \tag{2}$$

The TF representation is actually the representation which was used in previous papers in the domain of malware classification, where counting words was replaced by other methods for representing executable files (i.e., byte-sequence n-grams). Nevertheless, in the textual domain it was shown that the TFIDF is a richer and more successful representation for retrieval and categorization purposes (Salton et al., 1975). Moreover, in the textual domain, stop-words, which appear more frequently (e.g. the, to, is) are often removed in order to mutually reduce dimensionality and increase text categorization accuracy. Such terms can be characterized by having high DF value.

The domain of malware classification is characterized by an enormous vocabulary size, especially when using byte- or OpCode-sequence n-grams, and it is thus too large to be processed straightforwardly by ML algorithms. In such cases, *Feature Selection* algorithms can be applied in order to select a subset of features which are the best for discriminating between various categories of executables' classification (i.e. malicious or benign).

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 first phase (Fig. 1a), 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 the pre-determined vocabulary (which can be an outcome of the setup Feature Selection process). The representative vectors of the files in the training set and the real classification of each file (benign/malicious) serve as input for a training algorithm (such as a Decision Tree or Artificial Neural Network). By processing these vectors, the algorithm generates a classifier. Next, during the testing phase (Fig. 1b), a testing-set collection of new benign and malicious files that did not appear in the training set, are classified by the classifier calibrated in the training phase. Each file in the test set is first parsed and the representative vector is extracted (using the same vocabulary used in the training phase). Based on this vector, the classifier classifies a 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.

For evaluation purposes, the following classical measures are usually employed. The *True Positive Rate* (TPR) measure is the number of *positive* instances (i.e. malware files) classified correctly, as shown in Equation (3). *False Positive Rate* (FPR) is the number of *negative* instances (i.e. benign files) misclassified (Equation (3)). The *Total Accuracy* measures the number of absolutely correctly classified instances, either positive or negative, divided by the entire number of instances shown in Equation (4).

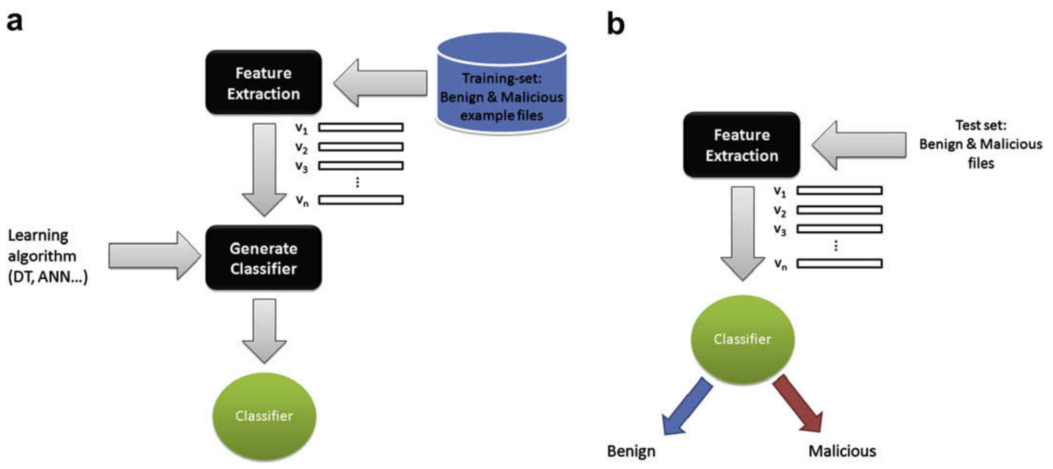


Fig. 1 – Training and testing phases of Machine Learning classifiers. (a) Training; (b) Testing.

$$\text{TPR} = \text{Sensitivity} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}|}; \quad \text{FPR} = \text{Specificity} = \frac{|\text{FP}|}{|\text{FP}| + |\text{TN}|} \tag{3}$$

$$\text{Total Accuracy} = \frac{|\text{TP}| + |\text{TN}|}{|\text{TP}| + |\text{FP}| + |\text{TN}| + |\text{FN}|} \tag{4}$$

Following the aforementioned theoretical background, related work in the area of applying ML techniques for detecting new malware can be classified according to the following three dimensions (Fig. 2):

File representation method deals with the patterns used for representing executable files. In other words, the *type* of features (i.e., n-grams, executable header's attributes) extracted from files. In this paper we focus only on static (morphological) features and not on dynamic (behavioral) features.

In ML applications throughout numerous application domains the large number of features, many of which not contributing to the classification accuracy (or even diminishing it), presents a huge problem. Thus, reducing 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 may exceed millions of features, far more than can be processed by any classification algorithm within a reasonable period of time. Additionally, it is important to identify those terms that appear in most of the files, in order to avoid vectors that contain many zeros. *Feature selection* methods aim to reduce the dimensionality and enhance compactness of the feature vector representing the files.

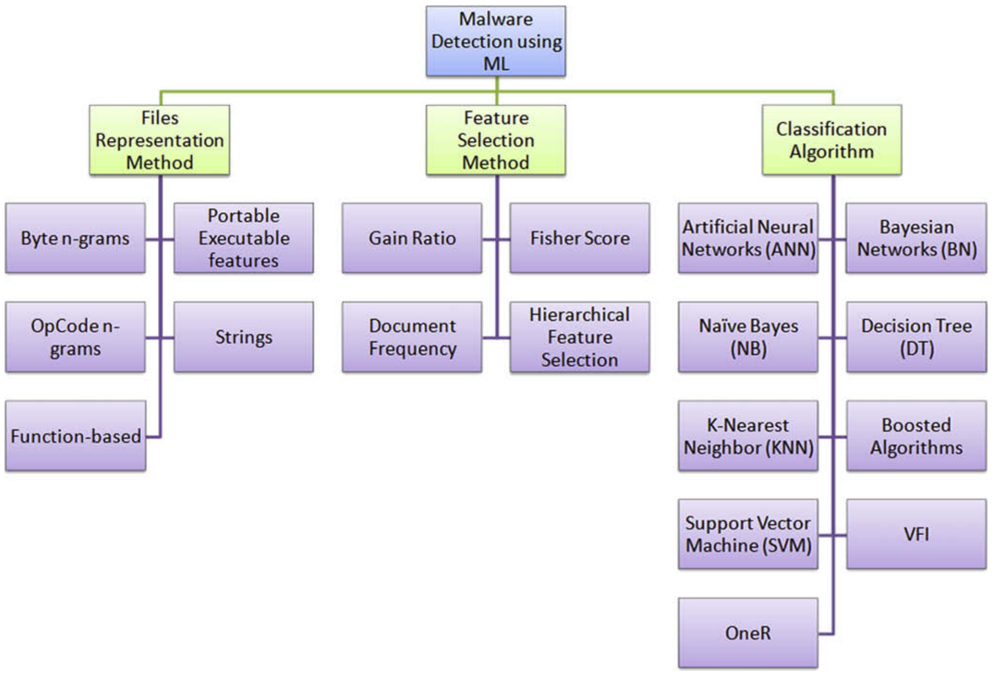


Fig. 2 – A taxonomy for malware detection using machine learning classifiers.

Classification algorithms are used for inducing a classifier. In the following sections we describe in detail the leading methods related to the aforementioned dimensions in the taxonomy and survey related papers.

3. Executable files representation methods

The first step in employing an ML approach for malware detection is to determine the representation of executable files. Several representation patterns of executable files were mentioned in the literature.

Schultz et al. (2001) were the first to introduce the idea of applying Machine Learning (ML) methods for the detection of different types of malware based on their respective binary codes. They used three different feature extraction (FE) approaches: Byte-sequence n-grams, Portable Executable (PE), and String features.

Byte n-gram features are sequences of *n* bytes extracted from an executable file, and thus words in the text categorization problem are congruent with byte n-grams in the malware detection problem. Although this type of feature does not provide meaningful information, it was shown to yield high accuracy in detecting new malware. Abou-Assaleh et al. (2004) used byte n-gram features extracted from the binary code of the file where the *L* most frequent n-grams of each class in the training set are chosen and represent the profile of the class. A new instance is associated with a class with the closest profile using K-Nearest Neighbors (KNN) algorithm. The experiment conducted on a small set of 65 malicious and benign files achieved 98% accuracy. Byte n-grams were also evaluated in Moskovitch et al. (2008d).

Portable Executable features are extracted from certain parts of EXE files stored in Win32 PE binaries (EXE or DLL). These are meaningful features that might indicate that the file was created or infected to perform malicious activity. Among features to be extracted are:

- Data extracted from the PE Header that describes *physical structure of a PE binary* (e.g., creation/modification time, machine type, file size)
- *Optional PE Header information* describing the logical structure of a PE binary (e.g., linker version, section alignment, code size, debug flags)
- *Import Section*: which DLLs were imported and which functions from which imported DLLs were used
- *Exports Section*: which functions were exported (if the file being examined is a DLL)
- *Resource Directory*: resources used by a given file (e.g., dialogs, cursors)
- *Version Information* (e.g., internal and external name of a file, version number)

String features are based on plain-text strings that are encoded in program files (such as "windows", "kernel", "reloc" etc) and possibly can also be used to represent files similarly to the text categorization problem. String features are used in Schultz et al. (2001) and provided the best performance, with 97.11% accuracy, when compared to using PE features and byte n-grams.

OpCode n-grams are used for representing executable files through streamlining an executable into a series of OpCodes (Dolev and Tzachar, 2008). 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. Being an essential element of machine language commands, OpCodes have been used for statically analyzing application and for detecting malware. Karim et al. (2005) addressed the tracking of malware evolution based on OpCode sequences and permutations. Data mining methods (logistic regression, neural networks and decision tree) are used in Siddiqui et al. (2008) to automatically identify critical instruction sequences that can classify programs as malicious or benign (with an accuracy of 98.4%). Bilar (2007) examined the OpCode frequency distribution difference between malicious and benign code. A total of 67 malware executables were compared with the aggregate statistics of 20 benign samples. The results indicate that malware OpCode distribution (especially rare OpCodes) significantly differs from that of benign software, suggesting that the method can be used to detect malicious code. Moskovitch et al. (2008a) provide an extensive evaluation using a test collection comprised of more than 30,000 files. Different settings of OpCode-sequence n-gram representations and five types of classifiers yielded an accuracy of up to 99%. Nevertheless, the OpCode approach is not always feasible because some executable files cannot be disassembled properly.

Function-based feature extraction (Menahem, 2008; Menahem et al., 2008) extracts functions that reside in the binary representation of a file and uses them to produce various attributes representing the file, such as: the total number of functions detected, size of the longest detected function, size of the shortest detected function, average size of detected functions, standard deviation of the size of detected functions, number of functions divided into groups by its length in bytes (16, 24, 40, 64, 90, 128 and 256 bytes) and more.

4. Feature selection methods

Various feature selection methods, including Document Frequency, Fisher Score, Gain Ratio and Hierarchical Feature Selection have been commonly used for reducing dimensionality of file representations. Feature selection methods operate according to the *filters approach* (Mitchell, 1997). 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 allows us to compare the performances of the different classification algorithms.

The *Document Frequency* (DF) measure is the amount of files in which the term appeared in. *Gain Ratio* (GR), originally presented by Quinlan in the context of *Decision Trees* (Mitchell, 1997), was designed to overcome a bias in the *Information-Gain* (IG) measure which quantifies the expected reduction of entropy caused by partitioning the examples according to a chosen feature. Given entropy $E(S)$ as a measure of the impurity in a collection of items, it is possible to quantify the effectiveness of a feature in classifying the training data. Equation (4) presents the formula of the entropy of a set of items S, based on C subsets of S (for example, classes of the items), represented by S_c. *Information-Gain* (IG) measures the expected reduction of entropy caused by portioning the examples according to attribute A, in which V is the set of possible values of A, as shown in Equation (3). These equations refer to discrete values; however, it is possible to extend them to continuous attributes.

$$\mathrm{IG}\,(S,A) = E(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} \cdot E(S_v) \tag{5}$$

$$E(S) = \sum_{c \in C} -\frac{|S_c|}{|S|} \cdot \log_2 \frac{|S_c|}{|S|} \tag{6}$$

Kolter and Maloof (2006) and Zhang et al. (2007) used Information-Gain measure. The IG measure favors features with a high variety of values over those with only a few. GR overcomes this problem by considering how the feature splits the data (Equations (5) and (6)), where S_i are d subsets of examples resulting from portioning S by the *d-valued* feature A. SI computes the entropy of S after portioning the examples according to attribute A

$$\mathrm{GR}\,(S,A) = \frac{\mathrm{IG}\,(S,A)}{\mathrm{SI}\,(S,A)} \tag{7}$$

$$\mathrm{SI}\,(S,A) = -\sum_{i=1}^{d} \frac{|S_i|}{|S|} \cdot \log_2 \frac{|S_i|}{|S|} \tag{8}$$

The *Fisher Score* ranking technique (Golub et al., 1999) calculates the difference, described in terms of mean and standard deviation, between the positive and negative examples relative to a certain feature. Equation (7) defines the Fisher score, in which R_i is the rank of feature i, describing the proportion of the subtraction of the mean of the feature i values in the positive examples (p) and the negative examples (n), and the sum of their standard deviation. The bigger the R_i, the bigger the difference between the values of positive and negative examples relative to feature i; thus, this feature is more *important* for separating the positive and negative examples. This technique is described in detail in Golub et al., (1999).

$$R_i = \frac{|\mu_{i,p} - \mu_{i,n}|}{\sigma_{i,p} + \sigma_{i,n}} \tag{9}$$

Moskovitch et al. (2008d,a) compared, as part of the overall settings in the training phase, the aforementioned methods of feature selection. Each feature selection method was used to select the top 50, 100, 200 and 300 most highly-ranked features. The results of the evaluation in Moskovitch et al. (2008d), which used the byte n-gram features, showed that the Fisher Score was very accurate when used with just 50 features and maintained this high performance level as the number of features increased. When more than 200 features were added, the accuracy of GR increased significantly and the accuracy of DF decreased.

Moskovitch et al. (2008a), which employed the OpCode n-gram features, showed that the Document Frequency (DF) was very accurate for most of the top features, while the Fisher Score performed very well, especially when fewer features were used (top 50 and top 100).

Henchiri and Japkowicz (2006) presented a hierarchical feature selection approach which selects the n-gram features that appear at rates above a specified threshold in a specific virus family, as well as in more than a minimal amount of virus classes (families). Kolter and Maloof (2006) extracted n-grams from the executables. Most relevant n-gram attributes were chosen using the InfoGain ratio.

5. Classification algorithms

The representative vectors of the files in the training set are the input for a learning algorithm. Examples for such inducing algorithms are: Decision Trees (DT), Artificial Neural Networks (ANN), Naïve Bayes (NB), Support Vector Machine (SVM) etc. By processing these vectors, the learning algorithm generates a classifier. At the testing phase, a testing-set collection of new benign and malicious files (files that did not appear in the training set) are being 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 (by using the same vocabulary that was used in the training phase). Based on this vector, the classifier will classify the file to the benign or malicious class.

An *Artificial Neural Network* (ANN) (Bishop, 1995) is an information processing paradigm inspired by the way biological nervous systems, such as the brain, process information. The key element is the structure of the information processing system, which is a network composed of a large number of highly interconnected neurons working together in order to approximate a specific function. An ANN is configured for a specific application, such as pattern recognition or data classification, through a *learning process* during which the individual weights of different neuron inputs are updated by a *training algorithm*, such as back-propagation. The weights are updated according to the examples the network receives, which reduces the *error function*. Equation (10) presents the output computation of a two-layered ANN, where x is the input vector, v_i is a weight in the output neuron, g is the activation function, w_{ij} is the weight of a hidden neuron and $b_{i,o}$ is a bias.

$$f(x) = g\Big[\sum_i v_i g\Big(\sum_j w_{ij} x_j + b_i + b_o\Big)\Big] \tag{10}$$

Decision Tree (DT) learners (Quinlan, 1993) are a well-established family of learning algorithms. Classifiers are represented as trees whose internal nodes are tests of individual features and whose leaves are classification decisions (classes). Typically, a greedy heuristic search method is used to find a small decision tree, which is induced from the data set by splitting the variables based on the expected Information Gain. Modern implementations include pruning, which avoids the problem of over-fitting. An important characteristic of Decision Trees is the explicit form of their knowledge, which can be easily represented as rules.

The *Naïve-Bayes* (NB) classifier (John and Langley, 1995) is based on the *Bayes theorem*, which in the context of classification states that the posterior probability of a class is proportional to its prior probability, as well as to the conditional likelihood of the features, given this class. If no independent assumptions are made, a Bayesian algorithm must estimate conditional probabilities for an exponential number of feature combinations. Naïve Bayes simplifies this process by assuming that features are *conditionally independent*, given the class, and requires that only a linear number of parameters be estimated. The prior probability of each class and the probability of each feature, given each class is easily estimated from the training data and used to determine the posterior probability of each class, given a set of features. Empirically, Naïve Bayes has been shown to accurately classify data across a variety of problem domains (Domingos and Pazzani, 1997).

Bayesian networks (BN), which are also called belief networks, are a form of probabilistic graphical model that represents a set of variables and their causal influences (Pearl, 1987). It is a directed acyclic graph of nodes and arcs representing causal dependencies among variables. The strength of the dependencies is expressed by conditional probability distributions. An arc is drawn from a parent node to a child node and each node is associated with the conditional probabilities based on its parent's variables. Unlike Naïve Bayes the variables are not assumed to be independent.

Schultz et al. (2001) applied four classifiers – a signature-based method (anti-virus), Ripper, a rule-based learner, Naïve Bayes, and Multi-Naïve Bayes. This study found that all the ML methods were more accurate than the signature-based algorithm. The ML methods were more than twice as accurate, with the out-performing method being Naïve Bayes, using strings, or Multi-Naïve Bayes using byte sequences.

Support Vector Machines (SVMs) (Joachims, 1998) are a binary classifier which attempts to find a linear hyper-plane separating given examples into the two given classes. Later, an extension that enables handling multi-class classification was developed. SVM is widely known for its capacity to handle a large amount of features, such as text, as was shown by Joachims (1998).

Boosting is a method for combining multiple classifiers. Adaboost was introduced by Freund and Schapire (1999) and among its many variants is the Adaboost.M1 that is implemented in WEKA (Witten and Frank, 2005). Given a set of examples and a base classifier, it generates a set of hypotheses combined by weighted majority voting. Learning is achieved in iterations. In each iteration, a new set of instances is selected by favoring misclassified instances of previous iterations. This is done using an iteratively updated distribution that includes a probability for each instance to be selected in the next iteration.

The *K-Nearest Neighbor* (KNN) algorithm (Kibler, 1991) is among the simplest of all ML algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors. k is a positive integer, typically small. If $k = 1$, then the object is simply assigned to the class of its nearest neighbor. In binary (two-class) classification problems, it is helpful to choose k to be an odd number as this prevents tied votes. The same method can be used for regression, by simply assigning the property value for the object to be the average of the values of its k furthest neighbors. It can be useful to weigh the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the remote ones. The neighbors are taken from a set of objects for which the correct classification (or, in the case of regression, the value of the property) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. In order to identify neighbors, the objects are represented by position vectors in a multidimensional feature space. It is customary to use the Euclidean distance, though other distance measures, such as the Manhattan distance could in principle be used instead.

Kolter and Maloof (2004) evaluated data sets using several inducers: IBK (KNN), a similarity based classifier called the TFIDF classifier, Naïve Bayes, Support Vector Machine (SVM) and Decision Tree (J48). The last three of these were also boosted. Two main experiments were conducted on two different data sets, a small collection and a large collection. The small collection included 476 malicious and 561 benign executables and the larger collection included 1651 malicious and 1971 benign executables. In both experiments, the four best-performing classifiers were Boosted J48, SVM, boosted SVM and IBK. Boosted J48 out-performed the others. The authors indicated that the results of their byte-sequence n-gram study were better than those presented by Schultz et al. (2001). Since costs of misclassification error are usually unknown, the methods were evaluated using receiver operating characteristic (ROC) analysis. When classifying new instances into two optional classes 'benign' and 'malicious', Boosted Decision Trees out-performed other methods with an area under the ROC curve of 0.996. When classifying new instances into multi-class (such as virus, backdoor etc.), the area under the ROC curve reaches an average of 0.9.

Henchiri and Japkowicz (2006) applied several classifiers: ID3, J48 Naïve Bayes, SVM and SMO to the data set used by Schultz et al. (2001) and obtained results that were better than those obtained through traditional feature selection, as presented in Schultz et al. (2001), which focused mainly on 5-grams. However, it is not clear whether these results are reflective more of the feature selection method or of the number of features that were used.

In Abou-Assaleh et al. (2004) a new executable file was compared with the profiles of malicious and benign classes, and was assigned to the most similar. Two different data sets were used: the I-worm collection, which consisted of 292 Windows Internet worms and the win32 collection, which consisted of 493 Windows viruses. The best results were achieved by using 3–6 byte-sequence n-grams and a profile based on 500–5000 features. Schultz et al. (2001) conducted experiments using 4301 programs in Win32 PE format (3301 malicious programs and 1000 benign programs) and the results show that all ML methods outperform the signature-based classifier.

In Elovici et al. (2007), the Receiver Operating Characteristics (ROC) was calculated for several individual plug-in classifiers and a standard weighing mechanism that combines the individual plug-in decision into a final single decision based on the simple voting algorithm. The individual classifiers were induced using Artificial Neural Networks, Decision Trees and Bayesian Networks trained on byte-sequence 5-grams and Portable Executable features. The integrated classifier was shown to have the best true positive rate with a minimum false positive rate and thus superior over each individual ML technique.

Moskovitch et al. (2008d) compared the following classifiers on various representation settings of byte-sequence n-grams: ANN, DT, NB, Boosted Decision Trees (BDT), Boosted Naïve Bayes (BNB) and three types of SVMs. Each representation setting was determined by the size of n-gram (3, 4, 5 or 6), the term representation method (TF or TFIDF), the feature selection method (Document Frequency, Gain Ratio or Fisher Score), and the number of selected top features (50, 100, 200 or 300). The classifiers were compared based on the best representation settings, which employed the top 300 Fisher score-selected 5-gram terms represented by TF. Results indicated that the BDT, DT and ANN out-performed the NB, BDT, BNB and SVM classifiers by exhibiting lower false positive rates. The poor performance of the Naïve Bayes can be explained by the independence assumption of the NB classifier.

Moskovitch et al. (2008a) compared the Decision Tree, ANN, Naïve Bayes, Boosted DT and Boosted NB on OpCode-sequence n-gram features. Like in Moskovitch et al. (2008d), several representations of OpCode n-gram settings were considered. Results from evaluating the classifiers on the best setting, which employed the top 300 2-grams, represented by their TF values and selected by the DF measure, indicated that BDT, DT and ANN out-performed the NB, BDT, BNB classifiers in maintaining lower levels of false positives.

Random Forest, Hyper-Pipes, VFI, OneR and PART classifiers were evaluated by Menahem (2008) and Menahem et al. (2008). This research focused on ensembles of classifiers, employing various types of classifiers from different families in order to increase the variance which improves prediction accuracy. The concept of ensemble of multiple classifiers is described in Section 6.1.

A *Random Forest* (Kam, 1995) is a classifier that is comprised of a collection of decision tree predictors. Each individual tree is trained on a partial, independently sampled, set of instances selected from the complete training set. The predicted output class of a classified instance is the most frequent class output of the individual trees.

Hyper-Pipes classifier (Mitchell, 1997) records the range of values observed in the training data for each attribute and class. When classifying a new instance, the classifier checks which ranges contain the attribute values of the test instance, choosing the class with the largest number of correct ranges.

VFI (*Voting Feature Intervals*) construct intervals around each class by discretizing numeric attributes and using point intervals for nominal ones, record class counts for each interval on each attribute, and classify test instances by voting (Guvenir, 1997).

OneR classifier (Holte, 1993) produces simple rules based on one attribute only. It operates by generating a separate rule for each individual feature. Each feature is discretized into bins and within each bin the percentage that each class (diagnosis) appears is calculated. Each bin is assigned to the class that has the highest percentage within that bin. After forming the rules, the single feature with the smallest error rate during training is selected. The class for any test case is assigned based on the bin into which its value for the selected feature falls.

PART is a partial decision tree algorithm. It is a developed version of C4.5 and RIPPER algorithms and its main characteristic is that it does not need to perform global optimization like C4.5 and RIPPER to produce the appropriate rules (Mitchell, 1997).

6. Additional issues

6.1. *Weighting algorithms (ensembles)*

By its nature, the measured performance (e.g., accuracy) of ML classifiers is affected by two main factors: (1) the inducer algorithm used to generate the classifier (e.g., Artificial Neural Network, Decision Tree, Naïve Bayes); and (2) the type of features used to represent the instances. Thus, various types of classifiers have different "inductive biases". When generating multiple classifiers using different inducers and features (extracted from the same set of instances), each classifier will perform differently. It might be the case that a certain classifier will outperform others when classifying instances into a subset of classes (for example, a classifier that classifies an executable file as benign or Worm with high accuracy but with low accuracy when classifying into other malicious classes such as Trojan and virus).

The outcome of this observation is that one should use multiple classifiers that will classify a new instance predicting its real class (like a panel of experts), and then mesh the results, using an intelligent integration mechanism, into one coherent classification. Hopefully, this "super-classifier" will outperform each of the individual classifiers and classify new instances with higher accuracy by learning the strength and weakness of each classifier.

In order to make the ensemble more effective, there should be some sort of diversity between the classifiers (Kuncheva, 2005). In a Multi-Inducer strategy, diversity is obtained by using different types of inducers. Each inducer contains an explicit or implicit bias that leads it to prefer certain generalizations over others. Ideally, this multi-inducer strategy would always perform as well as the best of its components. Even more ambitiously, there is hope that this combination of paradigms might produce synergic effects, leading to levels of accuracy that neither atomic approach by itself would be able to achieve.

The main idea of an ensemble methodology is to combine a set of classifiers in order to obtain a better composite, global classifier. An ensemble methodology imitates our second nature to seek several opinions before making any crucial decision. We weigh the individual opinions, and combine them to reach a final decision. Recent studies evaluated the improvement in performance (in terms of accuracy, false positive rate and area under the ROC graph) when using ensemble algorithms to combine the individual classifiers.

Several researchers examined the applicability of ensemble methods in detecting malicious code. Zhang et al. (2007) classifies new malicious code based on *n*-gram features extracted from binary code of the file. First, *n*-gram based features are extracted and the best *n*-grams are chosen by applying the Information-Gain feature selection method. Then, a probabilistic neural network (PNN) is used to construct several classifiers for detection. Finally, the individual decisions of the PNN classifiers are combined by using the Dempster–Shafer theory (Dempster, 1967) to create the combining rules. The method was evaluated on a set of malicious executables downloaded from the website VX Heavens (http://www.vx.netlux.org) and clean programs gathered from a Windows 2000 server machine: a total of 423 benign programs and 450 malicious executable files (150 Viruses, 150 Trojans and 150 Worms) all in Windows PE format. The results indicated a better ROC curve for the ensemble of PNNs when compared to the individual best PNN of each of the three malware classes.

The idea of *Distribution Summation* combining method is to sum up the conditional probability vector obtained from each classifier (Clark and Boswell, 1991). The selected class is chosen according to the highest value in the total vector. Mathematically, it can be written as:

$$\text{Class}(x) = \underset{c_i \in \text{dom}(y)}{\arg\max} \sum_k \hat{P}_{M_k}(y = c_i|x) \quad (11)$$

Elovici et al. (2007) uses the Distribution Summation in order to combine the results of five different classifiers generated by training ANN, BN and DT inducers on byte n-grams and PE features. The ROC was calculated for the individual plug-in classifiers and for the Distribution Summation weighing schema. The integrated classifier was shown to have the best true positive rate with a minimum false positive rate and thus was shown to be superior over each individual ML technique.

Menahem (2008) and Menahem et al. (2008) performed a comprehensive and exhaustive evaluation of seven ensemble methods using five different base inducers (C4.5 Decision Tree, Naïve Bayes, KNN, VFI and OneR) on five malware data sets. The main goal was to find the best ensemble method for detecting malware in terms of accuracy, AUC and classification speed. The following combining algorithms were tested: Majority Voting, Distribution Summation, Naïve-Bayes Combination, Bayesian Combination, Performance Weighting, Stacking and Troika.

In the *Majority Voting* combining scheme (Kittler et al., 1998), a classification of an unlabeled instance is performed according to the class that obtains the highest number of votes (the most frequent vote). This method is also known as the Plurality Vote (PV) or the basic ensemble method (BEM). This approach has frequently been used as a combining scheme for comparing to newly proposed methods. Mathematically it can be written as:

$$\text{class}(x) = \underset{c_i \in \text{dom}(y)}{\arg\max} \left(\sum_k g(y_k(x), c_i) \right) \quad (12)$$

where $y_k(x)$ is the classification of the kth classifier and $g(y,c)$ is an indicator function defined as:

$$g(y, c) = \begin{cases} 1, y = c \\ 0, y \neq c \end{cases} \quad (13)$$

Note that in case of a probabilistic classifier, the crisp classification $y_k(x)$ is usually obtained as follows:

$$y_k(x) = \underset{c_i \in \text{dom}(y)}{\arg\max} \hat{P}_{M_k}(y = c_i|x) \quad (14)$$

where M_k denotes classifier k and denotes the probability of y obtaining the value c given an instance x.

In *Performance Weighting*, the weight of each classifier is set proportional to its accuracy performance on a validation set (Opitz and Shavlik, 1996):

$$\alpha_i = \frac{(1 - E_i)}{\sum_{j=1}^{t} (1 - E_i)} \quad (15)$$

where E_i is a normalization factor which is based on the performance evaluation of classifier i on a validation set.

In the *Bayesian Combination* method, the weight associated with each classifier is the posterior probability of the classifier given the training set (Buntine, 1990).

$$\text{Class}(x) = \underset{c_i \in \text{dom}(y)}{\arg\max} \sum_k P(M_k|S) \cdot \hat{P}_{M_k}(y = c_i|x) \quad (16)$$

where $P(M_k|S)$ denotes the probability that the classifier M_k is correct given the training set S. The estimation of $P(M_k|S)$ depends on the classifier's representation. To estimate this value for Decision Trees the reader is referred to Buntine (1990).

Using Bayes' rule, one can extend the *Naïve-Bayes* idea for combining various classifiers (John and Langley, 1995):

$$\text{Class}(x) = \underset{\substack{c_j \in \text{dom}(y) \\ \hat{P}(y = c_j) > 0}}{\arg\max} \hat{P}(y = c_j) \cdot \prod_{k=1} \frac{\hat{P}_{M_k}(y = c_j|x)}{\hat{P}(y = c_j)} \quad (17)$$

Stacking is a technique used to achieve the highest generalization accuracy (Wolpert, 1992). By using a meta-learner, this method tries to induce which classifiers are reliable and which are not. Stacking is usually employed to combine models built by different inducers. The idea is to create a meta-data set containing a tuple for each tuple in the original data set. However, instead of using the original input attributes, it uses the predicted classifications by the classifiers as the input attributes. The target attribute remains as in the original training set. A test instance is first classified by each of the base classifiers. These classifications are fed into a meta-level training set from which a meta-classifier is produced. This classifier (Meta-classifier) combines the different predictions into a final one. It is recommended that the original data set should be partitioned into two subsets. The first subset is reserved to form the meta-data set and the second subset is used to build the base-level classifiers. Consequently the meta-classifier predications reflect the true performance of base-level learning algorithms. Stacking performance can be improved by using output probabilities for every class label from the base-level classifiers. It has been shown that with stacking the ensemble performs (at best) comparably to selecting the best classifier from the ensemble by cross-validation (Dzeroski and Zenko, 2004).

Troika (Menahem, 2008; Menahem et al., 2008) was designed to address Stacking problems, namely, the poor performance on multi-class problems (Seewald, 2003). Troika's general purpose ensemble scheme may be used to combine any type of classifiers which were trained on any subgroup of possible classes of a problem's domain. In other words, it is possible with Troika to combine models (classifiers) that were trained on, and therefore may later predict, non-congruent data sets in terms of instances' classes. Troika uses three layers of classifier combination rather than one, as

Table 1 – Summary of related work on malware detection using ML based on static features.

Citation	Representation method	Feature selection method	Classifiers	Additional issues	Description
Schultz et al. (2001)	PE, Strings, byte n-grams		RIPPER, Naïve Bayes, Multi-Naïve Bayes		ML classifiers compared with signature-based detection. The study found that all of the ML methods were more accurate than the signature-based algorithm
Abou-Assaleh et al. (2004)	Byte n-grams		K-Nearest Neighbors		The experiment conducted on a small set of 65 malicious and benign files achieved 98% accuracy
Moskovitch et al. (2008d)	Byte n-grams	Document Frequency, Fisher Score, Gain Ratio	Artificial Neural Networks, Decision Trees, Naïve Bayes, Boosted Decision Trees, Boosted Naïve Bayes, Support Vector Machines	Imbalance Problem	- Large collection of 30,000 malicious and benign files were used - The Fisher Score feature selection out-performed the other methods - The ANN and DT achieved high mean accuracies, exceeding 94%, with low levels of false alarms - When setting a level of 10% malicious file percentage in the test set, as a real-life situation, a high level of accuracy (above 95%) was achieved when less than 33% of the files in the training set were malicious
Moskovitch et al. (2008a)	OpCode n-grams	Document Frequency, Fisher Score, Gain Ratio	Artificial Neural Networks, Decision Trees, Naïve Bayes, Boosted Decision Trees, Boosted Naïve Bayes	Imbalance Problem	- Large collection of 30,000 malicious and benign files was used. - The 2-gram OpCodes out-performed the others, and the Fisher Score and the DF feature selection were better than the Gain Ratio. - BDT, DT and ANN out-performed, while maintaining low levels of false positives. - When setting a level of 10% malicious files percentage in the test set, illustrating a real-life situation a high level of accuracy (above 99%) was achieved when 15% of the files in the training set were malicious
Moskovitch et al. (2008c)	Byte n-grams	Document Frequency, Fisher Score, Gain Ratio	Support Vector Machines	Active Learning	The evaluation of the classifier before the acquisition and after showed an improvement in accuracy.
Moskovitch et al. (2008b)	Byte n-grams	Fisher Score	Decision Trees	Chronological Evaluation	- 16% MFP in the training set, which correspond to the test set, yield a higher level of accuracy - More updated training set yields higher level of accuracy
Kolter and Maloof (2004)	Byte n-grams		TFIDF classifier, Naïve Bayes, Support Vector Machines, Decision Trees, Boosted Decision Trees, Boosted Naïve Bayes, Boosted SVM		- Collection of 1971 benign and 1651 malicious executables. - The four best-performing classifiers were Boosted J48, SVM, boosted SVM and IBK. Boosted J48 out-performed the others. - The authors indicated that results of their byte-sequence n-gram study were better than those presented by Schultz et al. (2001)

Kolter and Maloof (2006)	Byte n-grams	InfoGain	K-Nearest Neighbors, Naïve Bayes, Support VectorMachines, Decision Trees, Boosted Decision Trees, Boosted Naïve Bayes, Boosted SVM	Chronological Evaluation	The results indicate that importance of maintaining the training set up-to-date by acquisition of new executables in order to cope with unknown new executables
Henchiri and Japkowicz (2006)	Byte n-grams	Hierarchical feature selection	Decision Trees, Naïve Bayes, Support Vector Machines		Applied the classifiers on the data set used by Schultz et al. (2001) and obtained results that were better than those obtained through traditional feature selection
Zhang et al. (2007)	Byte n-grams	InfoGain	Probabilistic Neural Network	Weighting method: Dempster–Shafer	- 423 benign executables and 450 malicious executable files. - Based on the area under ROC curve performance criteria the ensemble PNN outperforms the single PNN in all cases
Elovici et al. (2007)	PE, byte n-grams	Fisher Score	Bayesian Networks, Artificial Neural Networks, Decision Trees	Weighting method: Distribution Summation	The simple Distribution Summation method improved the overall accuracy of each individual classifier
Menahem (2008) and Menahem et al. (2008)	PE, byte, n-grams, Function-based features	Document Frequency, Fisher Score, Gain Ratio	Decision Trees, K-Nearest Neighbors, VFI, OneR, Naïve Bayes	Weighting method: Voting, Distribution Summation, Naïve-Bayes Combination, Bayesian Combination, Performance Weighting, Stacking, Troika	- Troika's AUC and accuracy are very high. - When using a multi-class data set, its AUC improves in about 8%, producing a model with lower FPR and a nice capability of identifying the malware's specific class

with Stacking. The result is a better performing ensemble scheme, usually intended for multi-class problems, where there are enough instances (Menahem, 2008; Menahem et al., 2008). In Menahem (2008) and Menahem et al. (2008), Troika was shown to perform on average better, than the best classifier selected using cross-validation.

According to Menahem (2008) and Menahem et al. (2008), the benefit of using Troika as an ensemble method in the domain of unknown malware detection is twofold; first, its AUC and accuracy are very high. Second, when using a multi-class data set, its AUC improves in about 8%, producing a model with lower FPR and a nice capability of identifying the malware's specific class. If Execution time is a big concern, then Bayesian Combination will be the best choice, being second only to Troika in terms of accuracy and AUC.

6.2. *Imbalance problem*

The imbalance problem refers to a case where the proportion of instance classes in the training set is not reflective of their abundance at large. Most of the aforementioned studies presented evaluations based on test collections, having identical proportion of malicious versus benign files in the test collections (50–50%). These proportions do not reflect real-life situations, in which malicious code is commonly significantly less than 50% and might report optimistic results. As a case in point, a recent survey[1] by McAfee indicates that about 4% of search results from the major search engines on the web contain malicious code. Additionally, Shin et al. (2006) found in excess of 15% of the files in the KaZaA network contained malicious code.

The class imbalance problem was introduced to the Machine Learning research community a little over a decade ago (Kubat and Matwin, 1997; Fawcett and Provost, 1997; Ling and Li, 1998). Typically it 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 suboptimal due to the fact that the data sets are not balanced. This problem is even more important in fields where the natural data sets are highly imbalanced in the first place (Chawla et al., 2004), like in the domain of previously unencountered malware detection.

Over the years, the ML community has addressed the issue of class imbalances by employing two generic strategies. The first one, *classifier-independent*, consists of balancing the original data set, using different kinds of under-sampling or over-sampling approaches. In particular, researchers have experimented with random sampling, directed sampling and artificial sampling. According to the random sampling approach, instances from the training set are either duplicated or eliminated at random (e.g., Japkowicz and Stephen, 2002). In the directed sampling approach, specific instances are targeted for under-sampling or over-sampling with the idea of strengthening the most relevant data and weakening the least relevant ones (e.g., Japkowicz and Stephen, 2002; Kubat and Matwin, 1997). In artificial sampling, the smaller class is over-sampled with artificially generated data designed to augment the minority class without creating the risk of over-fitting (Chawla et al., 2002).

The second approach, involves modifying classifiers in order to adapt them to the data sets. In particular, this approach attempts to incorporate misclassification costs into the classification process and assign higher misclassification costs to the minority class so as to compensate for its small size. This was done for a variety of different classifiers such as Neural Networks, Random Forests, and SVM.

Nevertheless, in the malware detection domain data is not imbalanced in the training set, but rather in real-life conditions, as reflected by the test set. Thus, the main goal is to understand the optimal construction of a training set to achieve the best performance in real-life conditions.

Moskovitch et al. (2008d) and Moskovitch et al. (2008a) examined 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. Moskovitch et al. (2008d) performed an extensive empirical evaluation with a test set of more than 30,000 files to investigate the imbalance problem using byte-sequence n-grams. Five levels of Malicious Files Percentage (MFP) in the training set (16.7, 33.4, 50, 66.7 and 83.4%) were used to train classifiers. 17 levels of MFP (5, 7.5, 10, 12.5, 15, 20, 30, 40, 50, 60, 70, 80, 85, 87.5, 90, 92.5 and 95%) were set in the test set to represent various benign/malicious file ratios during the detection. The evaluation results suggest that varying classification algorithms react differently to the various benign/malicious files ratio. For 10% MFP in the test set, representing real-life conditions, the highest performance was achieved with less than 33.3% MFP in the training set, and in specific classifiers an accuracy above 95% was achieved.

Moskovitch et al. (2008a) performed a similar test to investigate the relationship between the MFP in the test set; however, they used OpCode-sequence n-grams. The results indicate that classifiers trained on OpCode n-grams are relatively indifferent to changes in the MFP level of the test set. Also, when setting a level of 10% MFP in the test set, illustrating a real-life situation, a high level of accuracy (above 99%) was achieved when 15% of the files in the training set were malicious. Contrasting these findings with the experiments employing byte-sequence n-grams (Moskovitch et al., 2008d), it is evident that the OpCode-sequence n-grams outperformed the byte-sequence n-grams when using similar classifiers.

6.3. *Active learning (AL)*

A major challenge facing the anti-virus community is the dynamic nature of malware and the need for constant and rapid acquisition of new malware samples in order to be aligned with emerging threats. The acquisition of new malicious files is essential for updating the classifiers (re-training). This is often done by using honey-pots or by continuous scanning of network traffic in an attempt to detect a new malware. However, the main challenge in both options is to scan all collected files efficiently, especially when scanning

[1] McAfee Study Finds 4% of Search Results Malicious, By Frederick Lane, June 4, 2007 [http://www.newsfactor.com/story.xhtml?story_id=010000CEUEQO].

high-throughput networks operating at multi-GB/s rates. In addition manual inspection of potentially malicious files is time-consuming, but in some ‘‘hard’’ cases need to be performed by human experts.

Moskovitch et al. (2008c) introduced the task of efficient acquisition of unknown malicious files in a stream of executable files. They proposed using Active Learning (AL) as a selective method acquiring the most important files in a stream and improving a classifier’s performance. In this method the active learner identifies new examples which are expected to be unknown and present a ranked list of the most informative examples, which are probably very different from what is already coded in the classifiers.

This approach can be applied at a network communication node (router) of a Network Service Provider (NSP) to increase the probability of acquiring new malicious files. Studies in several domains successfully applied AL in order to reduce the effort of labeling examples. Unlike in random learning, in which a classifier is trained on a pool of labeled examples, the classifier actively indicates the specific examples that should be labeled, which are commonly the most informative examples for the training task. In their research, Moskovitch et al. (2008c) used three SVM algorithms (three different kernels) along with multiple settings on byte-sequence n-grams. Two selective AL sampling methods were considered: Simple Margin (Tong and Koller, 2000) and Error Reduction (Roy and McCallum, 2001). Evaluation of the classifier before and after the acquisition using the AL methods showed a significant improvement in accuracy which justifies the AL process. In general, both AL sampling methods performed very well, with the Simple Margin out-performing the Error Reduction.

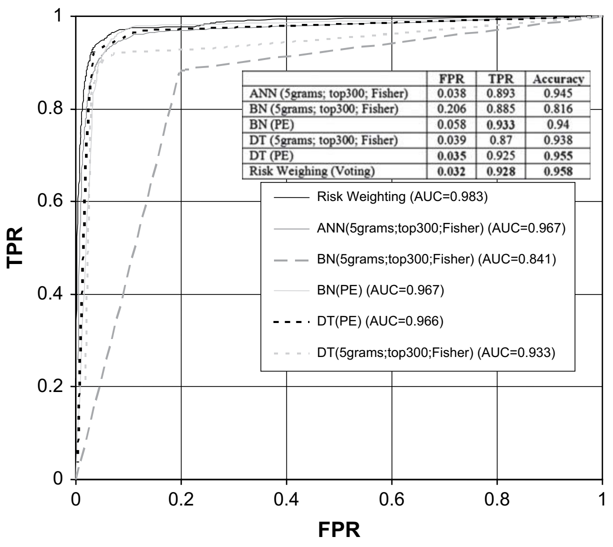
6.4. *Chronological evaluation*

Malware development and malware tool-kits evolve over time. Attackers try to find new creative methods to attack systems and to bypass defense mechanism.

The Chronological evaluation deals with the question of whether training a classifier using old executable files (both benign and malicious) can result in a classifier that will be able to detect newly released malware which is the real situation that the ML methods try to deal with. Most research did not take into account the release date of an executable and consequently the evolution of programs.

Kolter and Maloof (2006) reported an extension of their previous study (Kolter and Maloof, 2004). In attempts to estimate their ability to detect malicious codes based on their issue dates, classifiers were trained on files issued before July 2003, and then tested on 291 files issued from that point in time through August 2004. The results were, as expected, not as good as those of previous experiments. These results indicate the importance of maintaining an updated training set by acquisition of new executables, in order to cope with unknown new executables.

Moskovitch et al. (2008b) also evaluate the importance and need in updating the training set. The question was how important it is to update the repository of malicious and benign files and whether in specific years the files made a greater contribution to the accuracy, when introduced in the



	FPR	TPR	Accuracy
ANN (5grams; top300; Fisher)	0.038	0.893	0.945
BN (5grams; top300; Fisher)	0.206	0.885	0.816
BN (PE)	0.058	0.933	0.94
DT (5grams; top300; Fisher)	0.039	0.87	0.938
DT (PE)	0.035	0.925	0.955
Risk Weighing (Voting)	0.032	0.928	0.958

Fig. 3 – Evaluation results of five classifiers.

training set or in the test set. A large amount of collected tests were sub-divided according to the year in which the files were created. There were altogether seven training sets, in which each file contained files from 2000 to 2006 (according to the year). Each training set was evaluated separately on each following year till 2007. Two experimental setups were used, in which the training set had 16.6% and 50% Malicious File Percentage (MFP), while the MFP at the test was set to 10% representing real-life conditions. Results of the chronological evaluation, when using the ANN classifier, indicate that when having 16.6% MFP in the training set, which better corresponds to the percentage in the test set, a higher level of accuracy was achieved, and also a clear trend that the more updated the training set, the higher is the level of accuracy.

7. Conclusions

This article presented a taxonomy for classifying detection methods of new malicious code by Machine Learning (ML) methods based on static features extracted from executables. The taxonomy is now applied to classify up-to-date research on this topic (Table 1). The article addresses various facets of the detection challenge, including: file representation patterns, feature selection methods, classification algorithms, weighting ensembles, as well as the imbalance problem, active learning, and chronological evaluation.

We conclude that a framework for detecting new malicious code in executable files can be designed to achieve very high accuracy while maintaining low false positives (i.e. misclassifying benign files as malicious). The framework should include training of multiple classifiers on various types of features.

Following are some practical insights from our review. First, when setting-up a classifier for use in a real-life situation, OpCode and Portable Executables (PE) file representation patterns yield the highest level of accuracy. If extracting these features is not feasible, using the byte-sequence representation is the second best option.

Our findings indicate that applying weighting (ensemble) algorithms on the classification results of individual classifiers improves classification accuracy substantially. Experiments also showed that TFIDF representation has no added value over the TF, which is not the case in information retrieval applications. This is a very important finding, since using the TFIDF representation introduces some computational overhead in the maintenance of the training collection when it is updated. As a case in point, Fig. 3 depicts evaluation results of five classifiers when the file representation method is based on PE and byte-sequence n-grams; feature selection method is Fisher score (only for the n-gram); and classification algorithm is ANN, Bayesian networks and Decisions trees (Elovici et al., 2007).

Finally, training of classifiers should also consider the results presented for addressing the imbalance problem by generating classifiers that will perform accurately in real-life situations where the actual percentage of malicious files among all files is estimated to be approximately 10%. An Active Learning mechanism is recommended for rapid acquisition of new malware examples in order to maintain the training set up-to-date with emerging threats, and preserve high detection accuracy.

REFERENCES

Abou-Assaleh T, Cercone N, Keselj V, Sweidan R. N-gram based detection of new malicious code. In: Proceedings of the 28th annual international computer software and applications conference; 2004.

Bilar D. Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensic 2007;1:2.

Bishop C. Neural networks for pattern recognition. Oxford: Clarendon Press; 1995.

Buntine W. A theory of learning classification rules. Doctoral dissertation, Sydney: School of Computing Science, University of Technology; 1990.

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research (JAIR) 2002;16:321–57.

Chawla NV, Japkowicz N, Kotcz A. Editorial: special issue on learning from imbalanced data sets. SIGKDD Explorations Newsletter 2004;6(1):1–6.

Christodorescu M, Jha S. Testing malware detectors. ACM SIGSOFT Software Engineering Notes 2004;29(4):34–44.

Clark P, Boswell R. Rule induction with CN2: some recent improvements. In: Proc. of the European working session on learning; 1991. p. 151–63.

Dempster A. Upper and lower probabilities induced by multi-valued mapping. Annals of Mathematical Statistics 1967;2: 325–39.

Dolev S, Tzachar N. Malware signature builder and detection for executable code. Patent application; 2008.

Domingos P, Pazzani M. On the optimality of simple Bayesian classifier under zero-one loss. Machine Learning 1997;29:103–30.

Dzeroski S, Zenko B. Is combining classifiers with stacking better than selecting the best one? Machine Learning 2004;54(3):255–73.

Elovici Y, Shabtai A, Moskovitch R, Tahan G, Glezer C. Applying machine learning techniques for detection of malicious code in network traffic, KI; 2007. p. 44–50.

Fawcett TE, Provost F. Adaptive fraud detection. Data Mining and Knowledge Discovery 1997;1(3):291–316.

Freund Y, Schapire RE. A brief introduction to boosting. In: International joint conference on artificial intelligence; 1999.

Golub TR, Slonim DK, Tamayo P, Huard C, Gaasenbeek M, Mesirov JP, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science 1999;286:531–7.

Gryaznov D. Scanners of the year 2000: heuristics. In: Proceedings of the 5th international virus bulletin; 1999.

Guvenir GD. Classification by voting feature intervals. In: 9th European conference on machine learning; 1997. p. 85–92.

Heidari M. Malicious codes in depth. Available from: http://www.securitydocs.com/pdf/2742.pdf; 2004.

Henchiri O, Japkowicz N. A feature selection and evaluation scheme for computer virus detection. In: Proceedings of ICDM-2006, Hong Kong; 2006. p. 891–95.

Holte R. Very simple classification rules perform well on most commonly used datasets. Machine Learning 1993;11:63–91.

Japkowicz N, Stephen S. The class imbalance problem: a systematic study. Intelligent Data Analysis Journal 2002;6.

Joachims T. Making large-scale support vector machine learning practical. In: Schölkopf B, Burges C, A.S., editors. Advances in kernel methods: support vector machines. Cambridge, MA: MIT Press; 1998.

Jacob G, Debar H, Filiol E. Behavioral detection of malware: from a survey towards an established taxonomy. Journal in Computer Virology 2008;4:251–66.

John GH, Langley P. Estimating continuous distributions in Bayesian classifiers. In: Proc. of the conference on uncertainty in artificial intelligence; 1995. p. 338–45.

Kam HT. Random decision forest. In: Proc. of the 3rd int'l conf. on document analysis and recognition, Montreal, Canada, August 14–18; 1995. p. 278–82.

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.

Kibler DA. Instance-based learning algorithms. Machine Learning 1991:37–66.

Kienzle DM, Elder MC. Internet WORMS: past, present, and future: recent worms: a survey and trends. In: ACM workshop on rapid malcode (WORM03); 2003.

Kittler J, Hatef M, Duin R, Matas J. On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence 1998;20(3):226–39.

Kolter JZ, Maloof MA. Learning to detect malicious executables in the wild. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY: ACM Press; 2004. p. 470–8.

Kolter J, Maloof M. Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research 2006;7:2721–44.

Kubat M, Matwin S. Addressing the curse of imbalanced data sets: one-sided sampling. In: Proceedings of the 14th international conference on machine learning; 1997. p. 179–86.

Kuncheva LI. Diversity in multiple classifier systems (editorial). Information Fusion 2005;6(1):3–4.

Ling CX, Li C. Data mining for direct marketing: problems and solutions. In: Proceedings of the 4th ACM SIGKDD international conference on knowledge discovery and data mining; 1998. p. 73–79.

Menahem E. Troika – an improved stacking schema for classification tasks. M.Sc. thesis, Israel: Information System Engineering Dept., Ben-Gurion University of the Negev; 2008.

Menahem E, Shabtai, Rokach L, Elovici Y. Improving malware detection by applying multi-inducer ensemble. Computational Statistics and Data Analysis 2008.

Mitchell T. Machine learning. McGraw-Hill; 1997.

Moskovitch R, Feher, Tzachar N, Berger E, Gitelman M, Dolev S, et al. Unknown malcode detection using OPCODE representation. In: European conference on intelligence and security informatics 2008 (EuroISI08), Esbjerg, Denmark; 2008a.

Moskovitch R, Feher C, Elovici Y. Unknown malcode detection – a chronological evaluation. In: IEEE Intelligence And Security Informatics; 2008b.
Moskovitch R, Nissim N, Elovici Y. Acquisition of malicious code using active learning. In: PinKDD; 2008c.
Moskovitch R, Stopel D, Feher C, Nissim N, Elovici Y. Unknown malcode detection via text categorization and the imbalance problem. In: IEEE Intelligence and Security Informatics, Taiwan; 2008d.
Opitz D, Shavlik J. Generating accurate and diverse members of a neural network ensemble. Advances in Neural Information Processing Systems 1996;8:535–41.
Pearl J. Evidential reasoning using stochastic, simulation of causal models. Artificial Intelligence 1987;32(2):245–58.
Quinlan JR. C4.5: programs for machine learning. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc.; 1993.
Roy N, McCallum A. Toward optimal active learning through sampling estimation of error reduction. In: ICML; 2001.
Salton G, Wong A, Yang CS. A vector space model for automatic indexing. Communications of the ACM 1975;18:613–20.
Schultz M, Eskin E, Zadok E, Stolfo S. Data mining methods for detection of new malicious executables. In: Proc. of the IEEE symposium on security and privacy; 2001. p. 178–84.
Seewald AK. Towards understanding stacking – studies of a general ensemble learning scheme. PhD thesis, TU Wien; 2003.
Shin S, Jung J, Balakrishnan H. Malware prevalence in the KaZaA file-sharing network. In: Internet measurement conference (IMC), Brazil; 2006.
Siddiqui M, Wang MC, Lee J. Data mining methods for malware detection using instruction sequences. Artificial Intelligence and Applications 2008.
Tong S, Koller D. Support vector machine active learning with applications to text classification. Journal of Machine Learning Research 2000;2:45–66.
White S. Anatomy of a commercial-grade immune system. IBM Research Center; 1999.
Witten IH, Frank E. Data mining: practical machine learning tools and techniques. 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc.; 2005.
Wolpert DH. Stacked generalization. Neural Networks 1992;5:241–59.
Zhang B, Yin J, Hao J, Zhang D, Wang S. Malicious codes detection based on ensemble learning. Autonomic and trusted computing; 2007. p. 468–27.