Malicious Document Detection using Machine Learning: A Survey

[1] Kum Chandani Patel, [2] Prashant Modi

[1]Student, [2]Assistant Professor

[1] Department of Computer Engineering ,

[1] Government Engineering College, Modasa, India.

Abstract : Initial penetration is one of the first steps of an Advanced Persistent Threat (APT) attack, and it is considered one of the most significant means of initiating cyber-attacks aimed at organizations. In this paper, we first provide a brief overview on malware as well as the anti-malware industry, and present the industrial needs on malware detection. We then survey intelligent malware detection methods, the process of detection is usually divided into two stages: *feature extraction* and *classification/clustering*. The performance of such intelligent malware detection approaches critically depend on the extracted features and the methods for classification/clustering. We provide a comprehensive investigation on both the feature extraction and the classification/clustering techniques. We also discuss the additional issues and the challenges of malware detection using data mining techniques and finally forecast the trends of malware development

IndexTerms **- File Formats, Malicious Document, Malware Detection, Machine Learning.**

I. Introduction

Surfing the Internet, using social media and sharing information back and forth, is not as safe as it used to be. In line with the exponential growth in communicated data we, as the users, are more dependent on the Internet than ever. With the increased Internet usage, states, companies and private users have to secure themselves even better, in order to protect vulnerable information. During the last couple of years, more attention has also been put on Cyber Security. Since 2009, cyber-attacks against organizations have increased, and 91% of all organizations were hit by cyber-attacks in 2013. With the rapid development of the Internet, malware became one of the major cyber threats nowadays. Any software performing malicious actions, including information stealing, espionage, etc. can be referred to as malware. Kaspersky Labs define malware as “a type of computer program designed to infect a legitimate user's computer and inflict harm on it in multiple ways.”

While the diversity of malware is increasing, anti-virus scanners cannot fulfill the needs of protection, resulting in millions of hosts being attacked. According to Kaspersky Labs (2016), 6 563 145 different hosts were attacked, and 4 000 000 unique malware objects were detected in 2015. In turn, Juniper Research predicts the cost of data breaches to increase to $2.1 trillion globally by 2019. In addition to that, there is a decrease in the skill level that is required for malware development, due to the high availability of attacking tools on the Internet nowadays. High availability of anti-detection techniques, as well as ability to buy malware on the black market result in the opportunity to become an attacker for anyone, not depending on the skill level. Current studies show that more and more attacks are being issued by script-kiddies or are automated.

Therefore, malware protection of computer systems is one of the most important cybersecurity tasks for single users and businesses, since even a single attack can result in compromised data and sufficient losses. Massive losses and frequent attacks dictate the need for accurate and timely detection methods. Current static and dynamic methods do not provide efficient detection, especially when dealing with zero-day attacks. For this reason, machine learning-based techniques can be used.

This paper discusses the main points and concerns of machine learning-based malware detection, the following benefits to attackers:

1. It is easier to lure users into opening documents than into launching executable programs.
2. A steady stream of new vulnerabilities has been observed in the recent years in document viewers due to their high complexity caused, in turn, by the complexity of document formats.
3. Flexibility and versatility of document formats offer ample opportunities for obfuscation of embedded malicious content.

The same features also hinder the identification of malicious documents and increase the computational burden on the detection tools. The favorite formats used by attackers are PDF (targeting Adobe Reader), Flash (targeting Adobe Flash Player), and Microsoft Office files. Here it can be seen that both the total amount of malware and new malware have an exponential growth.

The goal is to determine the best feature representation method and how the features should be extracted, the most accurate algorithm that can distinguish the malware families with the lowest error.

1. What is Malware?

Malware is referred to by numerous names. Examples include malicious software, malicious code (MC) and malcode.

Numerous definitions have been offered to describe malware. For instance, Christodorescu and Jha describe a malware instance as a program whose objective is malevolent. McGraw and Morrisett define malicious code as "any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system." For the purposes of this survey, we adopt the description given by Vasudevan and Yerraballi in, which describes malware as "a generic term that encompasses viruses, trojans, spywares and other intrusive code."

The canonical examples of malware include viruses, worms, and Trojan horses. This designation warrants a slightly more detailed discussion on these malware.

Virus: A computer virus is code that replicates by inserting itself into other programs. A program that a virus has inserted itself into is infected, and is referred to as the virus's host. An important caveat is that viruses, in order to function, require their hosts, that is, a virus needs an existing host program in order to cause harm.

Worm: A computer worm replicates itself by executing its own code independent of any other program. The primary distinction between a virus and a worm is that a worm does not need a host to cause harm. Another distinction between viruses and worms is their propagation model. In general, viruses attempt to spread through programs/files on a single computer system. However, worms spread via network connections with the goal of infecting as many computer systems connected to the network as possible.

Trojan horse: A Trojan horse is malware embedded by its designer in an application or system. The application or system appears to perform some useful function but is performing some unauthorized action. Trojan horses are typically associated with accessing and sending unauthorized information from its host. Such Trojan horses can be classified as spyware as well. Malware embedded by its designer is not limited by this kind of malicious activity. The embedded malware could also be a time bomb.

Adware. The only purpose of this malware type is displaying advertisements on the computer. Often adware can be seen as a subclass of spyware and it will very unlikely lead to dramatic results.

Spyware. As it implies from the name, the malware that performs espionage can be referred to as spyware. Typical actions of spyware include tracking search history to send personalized advertisements, tracking activities to sell them to the third parties subsequently.

Rootkit. Its functionality enables the attacker to access the data with higher permissions than is allowed. For example, it can be used to give an unauthorized user administrative access. Rootkits always hide its existence and quite often are unnoticeable on the system, making the detection and therefore removal incredibly hard.

Backdoor. The backdoor is a type of malware that provides an additional secret "entrance" to the system for attackers. By itself, it does not cause any harm but provides attackers with broader attack surface. Because of this, backdoors are never used independently. Usually, they are preceding malware attacks of other types.

Keylogger. The idea behind this malware class is to log all the keys pressed by the user, and, therefore, store all data, including passwords, bank card numbers and other sensitive information.

Ransomware. This type of malware aims to encrypt all the data on the machine and ask a victim to transfer some money to get the decryption key. Usually, a machine infected by ransomware is "frozen" as the user cannot open any file, and the desktop picture is used to provide information on attacker's demands.

Remote Administration Tools (RAT). This malware type allows an attacker to gain access to the system and make possible modifications as if it was accessed physically. Intuitively, it can be described in the example of the TeamViewer, but with malicious intentions.

2.1 Who are the Users and Creators of Malware?

Malware writers/users go by a variety of names. Some of the most popular names are black hats, hackers, and crackers. The actual persons or organizations that take on the aforementioned names could be an external/internal threat, a foreign government, or an industrial spy.

There are essentially two phases in the lifecycle of software during which malware is inserted. These phases are referred to as the pre-release phase and the post release phase. An internal threat or insider is generally the only type of hacker capable of inserting malware into software before its release to the end-users. An insider is a trusted developer, typically within an organization, of some software to be deployed to its end users. All other persons or organizations that take on the hacker role insert malware during the post-release phase, which is when the software is available for its intended audience.

In creating new malware, black hats generally employ one or both of the following techniques: obfuscation and behavior addition/modification in order to circumvent malware detectors. Obfuscation attempts to hide the true intentions of malicious code without extending the behaviors exhibited by the malware. Behavior addition/ modification effectively creates new malware, although the essence of the malware may not have changed. The widespread use of the aforementioned techniques by malware coders along with those mentioned by researchers suggests that reused code is a major component in the development of new malware. This implication plays a critical role in some of the signature-based malware detection–sometimes referred to as misuse detection–methods .

2.2 The Malware Detector

All malware detection techniques can be divided into signature-based and behavior-based methods. Before going into these methods, it is essential to understand the basics of two malware analysis approaches: static and dynamic malware analysis. As it implies from the name, static analysis is performed "statically", i.e. without execution of the file. In contrast, dynamic analysis is conducted on the file while it is being executed for example in the virtual machine.

Static analysis can be viewed as "reading" the source code of the malware and trying to infer the behavioral properties of the file. Static analysis can include various techniques :

1. **File Format Inspection:** file metadata can provide useful information. For example, Windows PE (portable executable) files can provide much information on compile time,imported and exported functions, etc.
2. **String Extraction**: this refers to the examination of the software output (e.g. status or error messages) and inferring information about the malware operation.
3. **Fingerprinting**: this includes cryptographic hash computation, finding the environmental artifacts, such as hardcoded username, filename, registry strings.
4. **AV scanning:** if the inspected file is a well-known malware, most likely all anti-virus scanners will be able to detect it. Although it might seem irrelevant, this way of detection is often used by AV vendors or sandboxes to "confirm" their results.
5. **Disassembly**: this refers to reversing the machine code to assembly language and inferring the software logic and intentions. This is the most common and reliable method of static analysis.

Static analysis often relies on certain tools. Beyond the simple analysis, they can provide information on protection techniques used by malware. The main advantage of static analysis is the ability to discover all possible behavioral scenarios. Researching the code itself allows the researcher to see all ways of malware execution, that are not limited to the current situation. Moreover, this kind of analysis is safer than dynamic, since the file is not executed and it cannot result in bad consequences for the system. On the other hand, static analysis is much more time-consuming. Because of these reasons it is not usually used in real-world dynamic environments, such as anti-virus systems, but is often used for research purposes, e.g. when developing signatures for zero-day malware.

Another analysis type is **dynamic analysis**. Unlike static analysis, here the behavior of the file is monitored while it is executing and the properties and intentions of the file are inferred from that information. Usually, the file is run in the virtual environment, for example in the sandbox. During this kind of analysis, it is possible to find all behavioral attributes, such as opened files, created mutexes, etc. Moreover, it is much faster than static analysis. On the other hand, the static analysis only shows the behavioral scenario relevant to the current system properties.

Now, having the background on malware analysis, we can define the detection methods. The **signature-based analysis** is a static method that relies on pre-defined signatures. These can be file fingerprints, e.g. MD5 or SHA1 hashes, static strings, file metadata. The scenario of detection, in this case, would be as follows: when a file arrives at the system, it is statically analyzed by the anti-virus software. If any of the signatures is matched, an alert is triggered, stating that this file is suspicious. Very often this kind of analysis is enough since well-known malware samples can often be detected based on hash values.

However, attackers started to develop malware in a way that it can change its signature. This malware feature is referred to as polymorphism. Obviously, such malware cannot be detected using purely signature-based detection techniques. Moreover, new malware types cannot be detected using signatures, until the signatures are created. Therefore, AV vendors had to come up with another way of detection – behavior-based also referred to as **heuristics-based analysis**.

In this method, the actual behavior of malware is observed during its execution, looking for the signs of malicious behavior: modifying host files, registry keys, establishing suspicious connections. By itself, each of these actions cannot be a reasonable sign of malware, but their combination can raise the level of suspiciousness of the file. There is some threshold level of suspiciousness defined, and any malware exceeding this level raises an alert.

The accuracy level of heuristics-based detection highly depends on the implementation. The best ones utilize the virtual environment, e.g. the sandbox to run the file and monitor its behavior. Although this method is more time-consuming, it is much safer, since the file is checked before actually executing. The main advantage of behavior-based detection method is that in theory, it can identify not only known malware families but also zero-day attacks and polymorphic viruses. However, in practice, taking into account the high spreading rate of malware, such analysis cannot be considered effective against new or polymorphic malware.

2.3 Need for machine learning

As stated before, malware detectors that are based on signatures can perform well on previously-known malware, that was already discovered by some anti-virus vendors. However, it is unable to detect polymorphic malware, that has an ability to change its signatures, as well as new malware, for which signatures have not been created yet. In turn, the accuracy of heuristics-based detectors is not always sufficient for adequate detection, resulting in a lot of false-positives and false-negatives.

Need for the new detection method is dictated by the high spreading rate of polymorphic viruses. One of the solutions to this problem is reliance on the heuristics-based analysis in combination with machine learning methods that offer a higher efficiency during detection. When relying on heuristics-based approach, there has to be a certain threshold for malware triggers, defining the amount of heuristics needed for the software to be called malicious. For example, we can define a set of suspicious features, such as “registry key changed”, “connection established”, “permission changed”, etc. Then we can state, that any software, that triggers at least five features from that set can be called malicious. Although this approach provides some level of effectiveness, it is not always accurate, since some features can have more “weight” than others, for example, “permission changed” usually results in more severe impact to the system than “registry key changed”. To take these correlations into account and provide more accurate detection, machine learning methods can be used.

2. Overall Process Of Malware Detection Using Machine Learning

In the past years, many research efforts have been reported on malware detection based on machine learning algorithms. These techniques are capable of classifying previously unseen malware samples, identifying the malware families of malicious samples, and/or inferring signatures. In these systems, the detection is generally a two-step process: *feature extraction* and *classification*/*clustering*.

Machine Learning Basics

The rapid development of data mining techniques and methods resulted in Machine Learning forming a separate field of Computer Science. It can be viewed as a subclass of the Artificial Intelligence field, where the main idea is the ability of a system (computer program, algorithm, etc.) to learn from its own actions. It was firstly referred to as "field of study that gives computers the ability to learn without being explicitly programmed" by Arthur Samuel in 1959. A more formal definition is given by T. Mitchell: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." .The basic idea of any machine learning task is to train the model, based on some algorithm, to perform a certain task: classification, clusterization, regression, etc. Training is done based on the input dataset, and the model that is built is subsequently used to make predictions.To develop a deeper understanding, it is worth going through the general workflow of the machine learning process, which is shown in Figure 1.

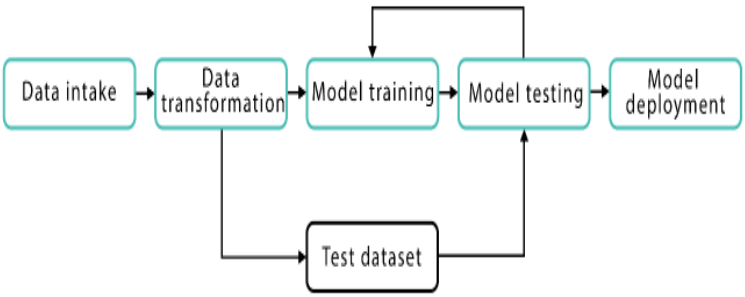

Figure 1. General workflow process

As it can be seen, the process consists of 5 stages:

1. **Data intake.** At first, the dataset is loaded from the file and is saved in memory.
2. **Data transformation**. At this point, the data that was loaded at step 1 is transformed, cleared, and normalized to be suitable for the algorithm. Data is converted so that it lies in the same range, has the same format, etc. At this point feature extraction and selection, which are discussed further, are performed as well. In addition to that, the data is separated into sets – 'training set' and 'test set'. Data from the training set is used to build the model, which is later evaluated using the test set.
3. **Model Training**. At this stage, a model is built using the selected algorithm.
4. **Model Testing**. The model that was built or trained during step 3 is tested using the test data set, and the produced result is used for building a new model, that would consider previous models, i.e. "learn" from them.
5. **Model Deployment**. At this stage, the best model is selected (either after the defined number of iteration or as soon as the needed result is achieved).

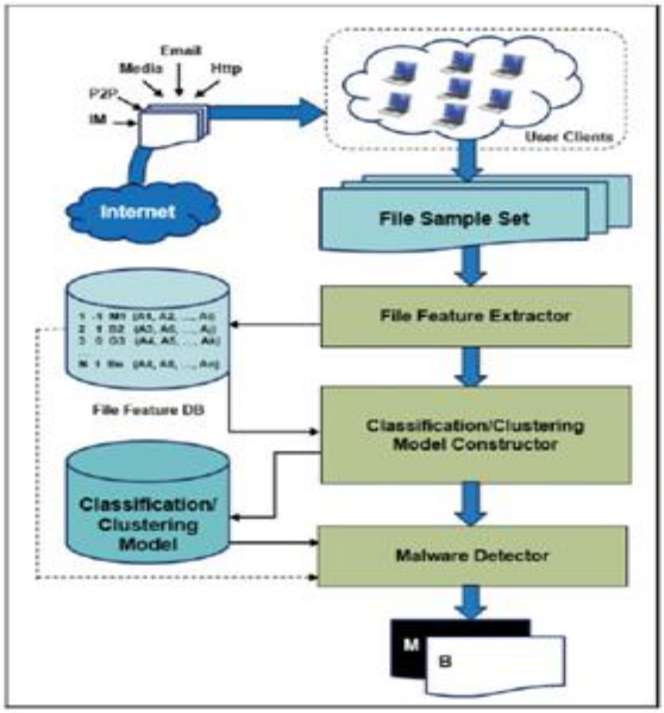

Figure 2.The overall process of malware detection using data mining techniques.

Figure 2 shows the overall process of malware detection using data mining techniques. In the first step, various features such as API calls, binary strings, and program behaviors are extracted statically and/or dynamically to capture the characteristics of the file samples. In the second step, intelligent techniques such as classification or clustering are used to automatically categorize the file samples into different classes/groups based on the analysis of feature representations. Note that these data-mining-based malware detectors mainly differ on the feature representation and the employed data mining techniques.

Classification: To classify any unknown file, which could be either benign or malicious, the classification process can be divided into two consecutive steps: model construction and model usage. In the first step, training samples including malware and benign files are provided to the system. Then, each sample is parsed to extract the features representing its underlying characteristics. The extracted features are then converted to vectors in the training set. Both the feature vectors and the class label of each sample (i.e., malicious or benign) are used as inputs for a classification algorithm.

Clustering: In many cases, very few labeled training samples exist for malware detection. Hence, researchers have proposed the use of clustering to automatically group malware samples that exhibit similar behaviors into different groups. Clustering is the task of grouping a set of objects such that objects in the same group (called a cluster) are more similar (e.g., using certain distance or similarity measures) to each other than to those in other groups (clusters). Clustering allows automatic malware categorization and also enables the signature generation for detection.

For evaluation purposes, the following classical measures shown in Table I are usually employed to evaluate the performance of classification-based malware detection. Note that in malware detection, malware samples are often used as positive instances. The True Positive Rate (TPR) measure is the rate of malware samples (i.e., positive instances) correctly classified by the classification model, while the False Positive Rate (FPR) is the rate of benign files (i.e., negative instances) wrongly classified (i.e., misclassified as malware samples).

The Accuracy (ACY) measures the rate of the correctly classified file instances, including both positive and negative instances. There is another type of measurement approach, named the interactive-based measurement approach, which attempts to measure the performance of malware detection methods/systems at the transaction level (i.e., considering different user bases).

Table1.Measures of Classification-Based Malware Detection Performance

Measures	**Specifications**
True Positive(*TP*)	Number of file samples correctly classified as malicious
True Negative(*TN*)	Number of file samples correctly classified as benign
False Positive(*FP*)	Number of file samples wrongly classified as malicious
False Negative(*FN*)	Number of file samples wrongly classified as benign
TP Rate(*TPR*)	*TP/(TP+FN)*
FP Rate(*FPR*)	*FP/(FP+TN)*
Accuracy(*ACY*)	*(TP+TN)/(TP+TN+FP+FN)*

The interactive-based measurement approach measures the true protection and false positive impact of malware detection methods/systems based on the actual user base. For example, imagine that at time *T*, there are two testing malware files, *A* (which has 1 million users) and *B* (which has 10 users), and the malware detection method/system wrongly classified the first file *A* (e.g., a false negative, classifying the malware as a benign file), but classified the second file *B* correctly as a malware sample. Based on just these two files, the cumulative measurement would give the ACY of 50%, since it classified one file correctly and one file incorrectly; but in the interactive-based measurement approach, it would assign different weights to the false negatives and true positives based on the users of file *A* and *B*. For the *clustering*-based methods, the performance of different algorithms are usually evaluated by using Macro-F1 and Micro-F1 measures, which emphasize the performance of the system on rare and common categories, respectively.

a) Feature Extraction

Feature extraction method extracts the patterns used for representing the file samples.In this article, we mainly discuss the detection on Windows Portable Executable (PE) files. Note that PE is a common file format for Windows operating systems and PE malware is the majority of malware samples. Note that CIH, CodeBlue, CodeRed, Killonce, LoveGate, Nimda, Sircam, and Sobig all aim at PE files. There are mainly two different types of feature extraction in malware detection: static analysis and dynamic analysis.

1. Static Analysis

Static analysis analyzes the PE files without executing them. The target of static analysis can be binary or source codes [Christodorescu and Jha 2003]. A PE file needs to be decompressed/unpacked first if it is compressed by a third-party binary compression tool (e.g., UPX and ASPack Shell) or embedded within a homemade packer [Ye et al. 2007]. To decompile windows executables, the disassembler and memory dumper tools can be used. Disassemble tools (e.g., IDA Pro [IDAPro 2016]) display malware codes as Intel ×86 assembly instructions. Memory dumper tools (e.g., OllyDump [2006]and LordPE [2013]) are used to obtain protected codes located in the main memory and dump them to a file [Gandotra et al. 2014]. Memory dump is quite useful for analyzing packed executables that are difficult to disassemble. After the executable being unpacked and decrypted, the detection patterns used in static analysis can be extracted, such as Windows API calls, byte n-grams, strings, opcodes (operational codes), andcontrol flow graphs.

2. Dynamic Analysis

Dynamic analysis techniques (e.g., debugging and profiling) observe the execution (on a real or virtual processor) of the PE files to derive features. Various techniques, such as autostart extensibility points, function parameter analysis, function call monitoring, information flow tracking, and instruction traces can be applied to perform dynamic analysis. Typical dynamic analysis tools include Valgrind, QEMU, and strace. There has been a substantial amount of studies on dynamic analysis of malware, varying in the execution environment for the malware and analysis granularity.

3. Hybrid Analysis

Both static and dynamic feature extraction approaches have their own advantages and limitations. Compared with dynamic feature representation, the static approach is cheaper and can cover all code paths (including the program pieces that are not always executed), therefore providing a more accurate and complete characterization of the program functionalities. However, it has high performance overhead due to low-level mutation techniques (such as obfuscation and packing). On the contrary, dynamic analysis is resilient to low-level obfuscation and is suitable for detecting malware variants and new families, but often performs poorly on trigger-based malware samples.

Besides, dynamic analysis is cost expensive and not scalable due to its limited coverage. Based on the statistics from Comodo Cloud Security Center, about 80% of the file samples can be well-represented using static features, while just around 40% of the file samples can successfully run dynamically. Due to their respective pros and cons, neither static nor dynamic-based feature extraction approaches can provide a perfect solution to the feature extraction in malware analysis. Therefore, a comprehensive approach that integrates both static and dynamic analysis and gains the benefits of both would be desirable. *Hybrid analysis* is such an approach that combines the respective advantages of both static and dynamic analysis. For example, the packed malware can first go through a dynamic analyzer such as PolyUnpack, where the hidden-code bodies of a packed malware instance are extracted by comparing the runtime execution of themalware instance with its static code model. Once the hidden-code bodies are uncovered, a static analyzer can continue the analysis of the malware.

b) Feature Selection

The previous section introduced methods for feature extraction for malware analysis and detection. Before describing popular classification methods used for malware detection in the next section, here we first discuss key methods used for feature selection and how they have been used in malware detection. Feature selection is an important step in many classification or prediction problems. The need for feature selection is motivated by the fact that, in certain classification and prediction problems, the number of features could be quite large, at times running into the millions, and thus could significantly exceed the capacity of the underlying machine to process within a reasonable time. Further, improvement in classification performance and minimization of classification errors could significantly depend on the ability of the system to quickly identify, select, and use only the most representative features. For the case of malware detection in particular, it might be impractical to construct the required model using each and every extracted feature. For instance, the number of features generated using *n*-grams grows exponentially with *n*, and using all the features could be quite computationally intensive, in terms of both memory usage and CPU time. The large number of features could also introduce unnecessary noise and large amounts of redundant and irrelevant features, further confounding the classifier. To reduce these problems, malware classification methods often adopt a feature selection for dimensionality reduction and to improve the compactness of the feature representation.

Feature Selection is essentially the process of selecting a subset of relevant and informative features from a larger collection of features for use in model construction. The central assumption is that the data contains many redundant or irrelevant features, which can be eliminated without a significant negative impact on later classification or prediction performance. Redundant features are those that provide no additional information beyond what is already provided by the currently selected features. Irrelevant features are those that do not provide any useful information in the given context. Thus, what is irrelevant could be dependent on the specific application being considered. Three important considerations in feature selection are determining the starting point for the selection process, how the selection proceeds given this starting point, and the stopping criteria for the selection procedure. For instance, in Forward Feature Selection (FFS), the process starts with an empty feature set and with new features added successively to the set. An alternative is Backward Feature Selection (BFS), whereby the process starts by using the set of all available features, and then successively removes features from the set. Removal of features (in BFS) or addition of features (as in FFS) are usually performed based on certain selection criteria, for instance, by ranking the features based on their estimated discrimination ability. Thus, independent of the direction of the search for features to select, another key consideration is how the feature subsets are evaluated for inclusion in the selected subset.

c) Classification

After feature extraction (and feature selection), each file sample is now represented in a feature space using the extracted feature vectors. The feature space is the basis for subsequent analysis stages, such as classification or clustering. Classification is performed by using the feature space generated from a sample training set as the input to a learning algorithm. By analyzing the input vectors, the learning algorithm determines parameters and factors that will be applied on the feature vectors in order to classify them into some pre-defined classes. At the time of testing, a testing set containing a separate collection of malware and benign file samples is used. Similar to the training samples, representative features are extracted from each test sample. Each sample in the testing is then classified as either benign or malicious, using the same set of parameters and factors determined from the training samples.

1. K-nearest neighbours

K-Nearest Neighbors (KNN) is one of the simplest, though, accurate machine learning algorithms. KNN is a non-parametric algorithm, meaning that it does not make any assumptions about the data structure. In real world problems, data rarely obeys the general theoretical assumptions, making non-parametric algorithms a good solution for such problems. KNN model representation is as simple as the dataset – there is no learning required, the entire training set is stored.

KNN can be used for both classification and regression problems. In both problems, the prediction is based on the k training instances that are closest to the input instance. The schematic example is outlined in Figure 3.

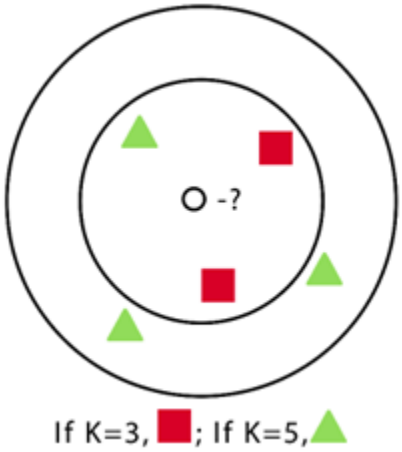

Figure 3. KNN example

Different distance measurement methods are used for finding the closest neighbors. The popular ones include Hamming Distance, Manhattan Distance, Minkowski distance:

$$Hamming\ Distance: dij=\Sigma|xik-xjk|pk=1\ Manhattan\ [1]$$

$$Distance: d1(p,q)=\|p-q\|1=\Sigma|pi-qi|ni=1\ Minkowski\ [2]$$

$$Distance=(\Sigma|xi-yi|pni=1)1p/\ [3]$$

The most used method for continuous variables is generally the **Euclidean Distance**, which is defined by the formulae below:

$$EuclidianDistance=\sqrt{\Sigma}(qi-pi)2ni=1\ ;\ p\ and\ q\ are\ the\ points\ in\ n-space[4]$$

Euclidian distance is good for the problems, where the features are of the same type. For the features of different types, it is advised to use, for example, Manhattan Distance. For the classification problems, the output can also be presented as a set of probabilities of an instance belonging to the class. For example, for binary problems, the probabilities can be calculated like $P(0)=N0N0+N1$, where P(0) is the probability of the 0 class membership and $N0$, $N1$ are numbers of neighbors belonging to the classes 0 and 1 respectively.

The value of k plays a crucial role in the prediction accuracy of the algorithm. However, selecting the k value is a non-trivial task. Smaller values of k will most likely result in lower accuracy, especially in the datasets with much noise, since every instance of the training set now has a higher weight during the decision process. Larger values of k lower the performance of the algorithm. In addition to that, if the value is too high, the model can overfit, making the class boundaries less distinct and resulting in lower accuracy again. As a general approach, it is advised to select k using the formula below:

$$k=\sqrt{n}\ [5]$$

For classification problems with an even number of classes, it is advised to choose an odd k since this will eliminate the possibility of a tie during the majority vote. The drawback of the KNN algorithm is the bad performance on the unevenly distributed datasets.

2. Support Vector Machines

Support Vector Machines (SVM) is another machine learning algorithm that is generally used for classification problems. The main idea relies on finding such a hyperplane, that would separate the classes in the best way. The term 'support vectors' refers to the points lying closest to the hyperplane, that would change the hyperplane position if removed. The distance between the support vector and the hyperplane is referred to as margin.

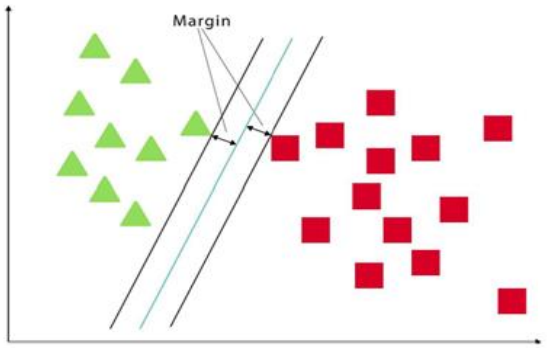

Figure 4. SVM scheme

On Figure 4, there is a dataset of two classes. Therefore, the problem lies in a two-dimensional space, and a hyperplane is represented as a line. In general, hyperplane can take as many dimensions as we want.

The algorithm can be described as follows:

1. We define X and Y as the input and output sets respectively. $(x1,1),\ldots,(xm,ym)$ is the training set.
2. Given x, we want to be able to predict y. We can refer to this problem as to learning the classifier y=f(x, a), where a is the parameter of the classification function.

3. F(x, a) can be learned by minimizing the training error of the function that learns on training data. Here, L is the loss function, and $Remp$ is referred to as empirical risk.

$$(a)=1m\Sigma(f(xi,a),yi)=Training\ Errormi=1 \text{ [6]}$$

4. We are aiming at minimizing the overall risk, too. Here, P(x,y) is the joint distribution function of x and y.

$$R(a)=\int l(f(x,a),y)dP(x,y)=Test\ Error \text{ [7]}$$

5. We want to minimize the Training Error + Complexity term. So, we choose the set of hyperplanes, so f(x) = (w·x)+b:

$$1m\Sigma(w\cdot xi+b,yi)+||w||2mi=1 \text{ subject to } mini|w\cdot xi|=1 \text{ [8]}$$

SVMs are generally able to result in good accuracy, especially on ”clean” datasets. Moreover, it is good with working with the high-dimensional datasets, also when the number of dimensions is higher than the number of the samples. However, for large datasets with a lot of noise or overlapping classes, it can be more effective. Also, with larger datasets training time can be high.

a. Class Probability is a probability of a class in the dataset. In other words, if we select a random item from the dataset, this is the probability of it belonging to a certain class.

b. Conditional Probability is the probability of the feature value given the class.

3. Naive Bayes

Naive Bayes is the classification machine learning algorithm that relies on the Bayes Theorem. It can be used for both binary and multi-class classification problems. The main point relies on the idea of treating each feature independently. Naive Bayes method evaluates the probability of each feature independently, regardless of any correlations, and makes the prediction based on the Bayes Theorem. To understand the algorithm of Naive Bayes, the concepts of class probabilities and conditional probabilities should be introduced first.

1. Class probability is calculated simply as the number of samples in the class divided by the total number of samples:

$$(C)=(instances\ in\ C)count(instances\ in\ Ntotal) \text{ [9]}$$

2. Conditional probabilities are calculated as the frequency of each attribute value divided by the frequency of instances of that class.

$$(V|C)=(instances\ with\ V\ and\ C)count(instances\ with\ V) \text{ [10]}$$

3. Given the probabilities, we can calculate the probability of the instance belonging to a class and therefore make decisions using the Bayes Theorem:

$$(A|B)=(B|A)P(A)P(B) \text{ [11]}$$

4. Probabilities of the item belonging to all classes are compared and the class with the highest probability if selected as a result.

The advantages of using this method include its simplicity and easiness of understanding.

4. J48Decision Tree

As it implies from the name, decision trees are data structures that have a structure of the tree. The training dataset is used for the creation of the tree, that is subsequently used for making predictions on the test data. In this algorithm, the goal is to achieve the most accurate result with the least number of the decisions that must be made. Decision trees can be used for both classification and regression problems.

Table 2. Decision tree example dataset

Predictors				**Target**
Outlook	**Temperature**	**Humidity**	**Windy**	**Play tennis**
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Overcast	Cool	High	True	No
Rainy	Cool	High	True	Yes
Rainy	Mild	High	False	No
Sunny	Cool	Normal	False	Yes

The common algorithm for decision trees is **ID3 (Iterative Dichotomiser 3)**. It relies on the concepts of the **Entropy** and **Information Gain**. Entropy here refers to the level of uncertainty in the data content. For example, the entropy of the coin toss would be indefinite, since there is no way to be sure in the result. Contrarily, a coin toss of the coin with two heads on both sides would result in zero entropy, since we can predict the outcome with 100% probability before each toss.

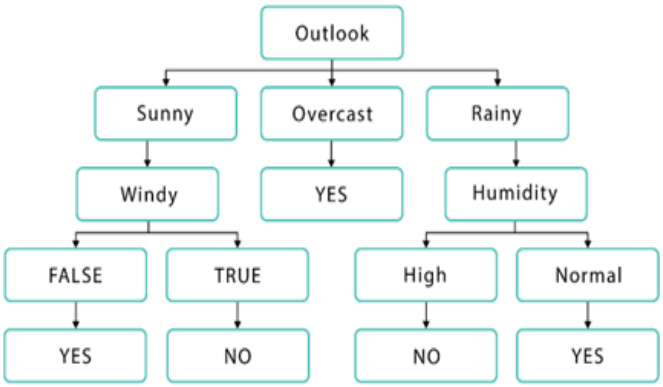

Figure 5. Decision tree example

In simple words, the ID3 algorithm can be described as follows: starting from the root node, at each stage we want to partition the data into homogenous (similar in their structure) dataset. More specifically, we want to find the attribute that would result in the highest information gain, i.e. return the most homogenous branches :

1. Calculate the entropy of the target.

$$(T,X)=\Sigma(c)E(c)c\in X \text{ [12]}$$

$$(S)=\Sigma-pi\log 2pici=1 \text{ [13]}$$

2. Split the dataset and calculate the entropy of each branch. Then calculate the information gain of the split, that is the differences in the initial entropy and the proportional sum of the entropies of the branches.

$$(T,X)=(T)-Entropy(T,X) \text{ [14]}$$

3. The attribute with the highest Gain value is selected as the decision node.
4. If one of the branches of the selected decision node has an entropy of 0, it becomes the leaf node. Other branches require further splitting.
5. The algorithm is run recursively until there is nothing to split anymore.

J48 is the implementation of the ID3 algorithm, that is included in one of the R packages, and this is the implementation we are going to use in our study.Decision tree method achieved its popularity because of its simplicity. It can deal well with large datasets and can handle the noise in the datasets very well. Another advantage is that unlike other algorithms, such as SVM or KNN, decision trees operate in a “white box”, meaning that we can clearly see how the outcome is obtained and which decisions led to it. These facts made it a popular solution for medical diagnosis, spam filtering, security screening and other fields.

5. Random Forest

Random Forest is one of the most popular machine learning algorithms. It requires almost no data preparation and modeling but usually results in accurate results. Random Forests are based on the decision trees described in the previous section. More specifically, Random Forests are the collections of decision trees, producing a better prediction accuracy. That is why it is called a ’forest’ – it is basically a set of decision trees.

In simple words, the algorithm can be described as follows :

1. Multiple trees are built roughly on the two third of the training data (62.3%). Data is chosen randomly.
2. Several predictor variables are randomly selected out of all the predictor variables. Then, the best split on these selected variables is used to split the node. By default, the amount of the selected variables is the square root of the total number of all predictors for classification, and it is constant for all trees.
3. Using the rest of the data, the misclassification rate is calculated. The total error rate is calculated as the overall out-of-bag error rate.
4. Each trained tree gives its own classification result, giving its own ”vote”. The class that received the most ”votes” is chosen as the result.

As in the decision trees, this algorithm removes the need for feature selection for removing irrelevant features – they will not be taken into account in any case. The only need for any feature selection with the random forest algorithms arises. when there is a need for dimensionality reduction. Moreover, the out-of-bag error rate, which was mentioned earlier can be considered the algorithm’s own cross-validation method. This removes the need for tedious cross-validation measures, that would have to be taken otherwise.

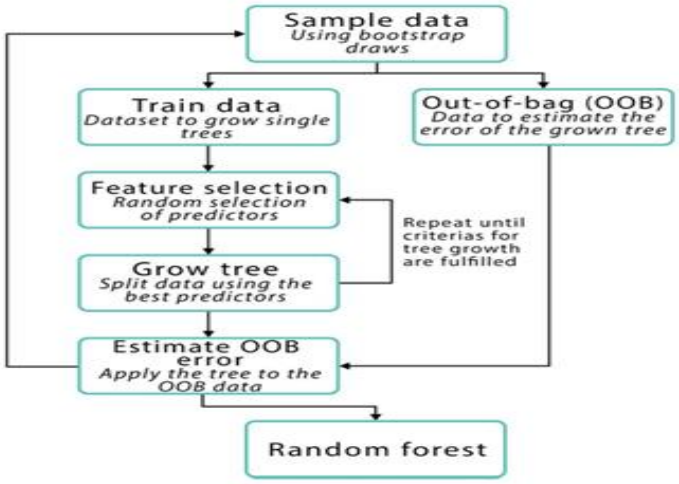

Figure 6. Random Forest scheme

Random forests inherit many of the advantages of the decision trees algorithms. They are applicable to both regression and classification problems; they are easy to compute and quick to fit. They also usually result in the better accuracy. However, unlike decision trees, it is not very easy to interpret the results. In decision trees, by examining the resulting tree, we can gain valuable information about which variables are important and how they affect the result. This is not possible with random forests. It can also be described as a more stable algorithm than the decision trees – if we modify the data a little bit, decision trees will change, most likely reducing the accuracy. This will not happen in the random forest algorithms – since it is the combination of many decision trees, the random forest will remain stable.

6. Multilayer Perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable. Multilayer perceptron are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.

7. Logistic Regression

The binary logistic regression model has extensions to more than two levels of the dependent variable: categorical outputs with more than two values are modelled by multinomial logistic regression, and if the multiple categories are ordered, by ordinal logistic regression, for example the proportional odds ordinal logistic model. The model itself simply models probability of output in terms of input, and does not perform statistical classification (it is not a classifier), though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make a binary classifier.

8. Logitboost

In machine learning and computational learning theory, LogitBoost is a boosting algorithm formulated by Jerome Friedman, Trevor Hastie, and Robert Tibshirani. The original paper casts the AdaBoost algorithm into a statistical framework. Specifically, if one considers AdaBoost as a generalized additive model and then applies the cost functional of logistic regression, one can derive the LogitBoost algorithm.

9. Bayesian network

A Bayesian network, Bayes network, belief network, Bayes(ian) model or probabilistic directed acyclic graphical model is a probabilistic graphical model (a type of statistical model) that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

10. Bagging

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

11. Sequential minimal optimization

Sequential minimal optimization (SMO) is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support vector machines. It was invented by John Platt in 1998 at Microsoft Research. SMO is widely used for training support vector machines and is implemented by the popular LIBSVM tool. The publication of the SMO algorithm in 1998 has generated a lot of excitement

in the SVM community, as previously available methods for SVM training were much more complex and required expensive third-party QP solvers.

12. AdaBoost

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

d) Clustering

The classification methods typically require a large number of labeled samples. In recent years, there have been research initiatives in automatic malware categorization using unsupervised techniques (i.e., clustering techniques). *Clustering*, a form of unsupervised learning, is the process of partitioning a given data set into groups (i.e., clusters) based on the pre-defined distance measures, such that the data points in a cluster should be close to each other and data points in different clusters are far away from each other. In malware detection, a cluster is a group of file samples sharing some common traits while being "dissimilar" to the malware samples from different clusters. *Partitioning* and *Hierarchical* clustering are two common types of clustering methods with different characteristics. Hierarchical clustering methods can handle data sets with irregular cluster structures, while partitioning clustering (e.g., *K-means*) is generally effective when the clusters have a globular shape. The choice of clustering algorithms in malware detection is largely based on the extracted features and their underlying feature distributions.

This chapter provided background on the machine learning that is essential for understanding the practical implementation of the project, that is described in the next chapter. The concepts of feature set, feature extraction, and selection methods were discussed along with the machine learning algorithms that will be used in practical part.

Table 3. Summary of Typical Classification Methods Used in Malware Detection

Survey	Classification Methods	Description
Kolter and Maloof [2004]	SVM, NB, DT, and their boosted versions	Using binary n-grams, based on their collected data set with 1,971 benign and 1,651 malicious executables, their experimental results indicated that boosted DTs performed best in malware detection.
Abou-Assaleh et al [2004]	SVM, DT, kNN classifiers	Based on a small sample collection of 65 malware and benign files, using binary n-grams of various lengths, the proposed method achieved 98% detection accuracy.
Henchiri and Japkowicz [2006]	NB, DTs, SVM, and Sequential Minimal Optimization (SMO)	Based on the 16-byte sequences, the obtained results were better than those obtained through traditional feature selection.
Wang et al. [2006]	SVM	Using both static and dynamic extracted features, based on their data collection consisting of 407 spyware and 740 benign programs, when 10-fold cross validation was considered, its overall accuracy reached 96.43%.
Ye et al. [2007, 2009]	NB, DTs, SVM, Associative Classification	Based on the Windows API call features, the associative classifier outperformed other classifiers.
Masud et al. [2007, 2008]	SVM, DT, NB, BDT, and BNB	Based on the hybrid feature sets, the results from Boosted J48 were almost the same as SVM.
Moskovitch et al.[2008a, 2008c]	ANN, DTs, BDT, NB, BNB, and SVMs	The results indicated that the BDT, DT, and ANN outperformed the NB, BDT, BNB, and SVM classifiers, based on the n-gram OpCode sequences and byte n-grams.
Siddiqui et al. [2009]	DT, Random Forest, and Bagging	Their experimental results showed that Random Forest performed best based on the variable length instruction sequences extracted from 2,774 samples including 1,444 worms and 1,330 benign files.
Ye et al. [2009]	DT, NB, SVM, Bagging	The experimental results showed that the ensemble of SVMs with bagging

		performed best based on the extracted interpretable strings.
Tian et al. [2010]	SVM, DT, Random Forest, Instance-based Classifier	They used the API call sequences dynamically extracted from 1,368 malware and 456 benign files to demonstrate their work and achieved an accuracy of over 97%.
Firdausi et al. [2010]	KNN, NB, DT, SVM, Multilayer Perceptron Neural Network (MLP)	The obtained results based on the dynamically extracted behaviors depicted that overall best performance was achieved by J48 DT.
Anderson et al. [2012]	SVM	By combining both static and dynamic analysis, it was tested on a dataset of 780 malware and 776 benign instances giving and accuracy of 98.07%.
Santos et al. [2013]	DT, kNN, BN, and SVM	It has been found that the hybrid approach enhanced the performance of both approaches when run separately, based on the static and dynamic analysis.
Islam et al. [2013]	DT, Random Forest, SVM, and Instance- based Classifier	By combining both static and dynamic analysis, the obtained results showed that meta-Random Forest performed best.
Nissim et al.[2017]	J48, Random Forest, Logitboost, Logistic Regression, SVM	It provided a reduction in the labeling efforts of 95.5% and 95.7%, respectfully with Random and the SVM-Margin
Šrndi, Nedim et al.[2016]	Random Forest	It has been implemented and evaluated on two formats, PDF and SWF (Flash)

3. Issues And Challenges

Previous studies proved that machine learning methods have been successfully used in malware detection and in the anti-malware industry. However, there are still many additional issues that need further investigation.

—**Evaluation and use of the detection results:** Though applying data mining techniques, like classification/clustering methods, can detect malware from the unknown file sample collection, the verification of the potentially malicious files is one of the challenging issues in real application. This is because the inspection of these potentially malicious files always requires the knowledge of domain experts and the manual inspection is always time-consuming.

—**Incremental learning:** Training a classifier using an historical file sample collection (containing both benign and malicious samples) is able to detect newly released malware. However, new malware samples are constantly produced on a daily basis and malware techniques are continuously evolving. To account for the temporal trends of malware writing, data-mining-based malware detection systems need to take the most recent file sample collection into consideration. In other words, to make the classifier(s) remain effective, the training sets should dynamically change to include new samples while retaining the main properties of historical data collection. Therefore, incremental learning is one of the issues of data-mining-based malware detection systems.

—**Active learning:** To further improve the detection accuracy, selecting representative sample(s) from large unknown file collections for labeling is also very important. For example, before being detected, the newly released Trojan-Downloader and its related trojans are collected from the clients and marked as unknown. If we can recognize the Trojan-Downloader and have it labeled, then, based on the extracted features and using classification/clustering methods, its related trojans can be correctly detected. Active learning, as an effective paradigm to address the data scarcity problem, optimize the learning benefit from domain experts' feedback, and reduce the cost of acquiring labeled examples for supervised learning, has been intensively studied in recent years. In malware detection, there is limited research using active learning to select a representative sample(s) from the large file sample collection. For instance, to further improve the performance of a classifier,

—**Prediction of malware prevalence:** Except for detecting malware from the unknown file collection, predicting the trend of malware prevalence is also very important. However, there is little research on the prediction of malware prevalence.

—**Adversarial Learning:** In malware detection, using the data mining approaches opens the possibility for the malware attachers to come up with ways to "mistrain" the classifiers (e.g., by changing the data distribution or feature importance). Thus, questions arise as to how to develop techniques that are robust and secure in adversarial scenarios.

4. Conclusion

In recent years, a few research efforts have been conducted on surveys of machine learning based malicious document detection, the authors reviewed the malware propagation, analysis and detection; the researchers surveyed the feature representation and classification methods for malware detection. In this article, we not only overview the development of malware and present the needs on malware detection, but also provide a comprehensive study on machine-learning-based methods for malware detection based on both static and dynamic representations. Furthermore, we also discuss the additional issues and challenges of malware detection using machine learning. Due to the exponential growth of malware samples, intelligent methods for efficient and effective malware detection at the cloud (server) side are urgently needed. As a result, much research has been conducted on developing intelligent malware detection systems using machine learning techniques. In these methods, the process of malware detection is generally divided into two steps: *feature extraction* and *classification/clustering*. We provide a comprehensive investigation on both the feature extraction and the classification/clustering steps. We conclude that machine learning based malware detection framework can be designed to achieve good detection performance with high accuracy while maintaining low false positives.

Many developed systems have been successfully integrated into commercial anti-malware products. The following are some practical insights from our view:

(1) Based on different data sets with different feature representation methods, there is no single classifier/ clustering algorithm always performing best. In other words, the performances of such malware detection methods critically depend on the extracted features, data distributions, and the categorization methods.

(2) Generally, compared with individual classifiers, an ensemble of classifiers can always help improve the detection accuracy. In real applications, a successful malware detection framework should utilize multiple diverse classifiers on various types of feature representations.

(3) For feature extraction, both static and dynamic analysis approaches have their own advantages and limitations.

(4) In order to achieve the best detection performance in real applications, it is often better to have enough training samples with balanced distributions for both classes (malware and benign files).

5. References

1. R. Bearden and D. C. Lo, “Automated Microsoft Office Macro Malware Detection Using Machine Learning,” pp. 4448–4452, 2017.
2. A. Cohen, N. Nissim, L. Rokach, and Y. Elovici, “SFEM: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods,” Expert Syst. Appl., vol. 63, pp. 324–343, 2016.
3. N. Nissim, A. Cohen, and Y. Elovici, “ALDOCX: Detection of Unknown Malicious Microsoft Office Documents using Designated Active Learning Methods Based on New Structural Feature Extraction Methodology.”Anubis. 2010
4. N. Šrndi, “Hidost : a static machine-learning-based detector of malicious files,” pp. 1–20, 2016.
5. J. Torres and S. D. L. Santos, “Malicious PDF Documents Detection using Machine Learning Techniques A Practical Approach with Cloud Computing Applications,” no. Icissp, pp. 337–344, 2018.
6. K. Chumachenko and I. Technology, “MACHINE LEARNING METHODS FOR MALWARE DETECTION AND,” 2017.
7. E. Menahem, A. Schclar, L. Rokach, and Y. Elovici, “XML-AD : Detecting anomalous patterns in XML documents,” Inf. Sci. (Ny)., 2015.
8. E. Gandotra, D. Bansal, and S. Sofat, “Malware Analysis and Classification : A Survey,” no. April, pp. 56–64, 2014.
9. P. Laptev, “Method for Effective PDF Files Manipulation Detection Supervisors :,” 2017.
10. N. Nissim et al., ALPD: Active Learning Framework for Enhancing the Detection of Malicious PDF Files. 2014.
11. N. Nissim et al., “Keeping pace with the creation of new malicious PDF files using an active - learning based detection framework,” Secur. Inform., pp. 1–20, 2016.
12. N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, “Expert Systems with Applications Novel active learning methods for enhanced PC malware detection in windows OS,” Expert Syst. Appl., no. March, 2014.
13. N. Nissim, A. Cohen, C. Glezer, and Y. Elovici, “Detection of malicious PDF files and directions for enhancements: A state-of-the art survey,” Comput. Secur., vol. 48, pp. 246–266, 2015.
14. X. Lu, J. Zhuge, R. Wang, Y. Cao, and Y. Chen, “De-obfuscation and detection of malicious PDF files with high accuracy,” Proc. Annu. Hawaii Int. Conf. Syst. Sci., pp. 4890–4899, 2013.
15. R. Moskovitch, N. Nissim, and Y. Elovici, “Malicious Code Detection Using Active Learning,” no. june 2007, pp. 74–75, 2009.
16. R. Bearden and D. C. Lo, “Automated Microsoft Office Macro Malware Detection Using Machine Learning,” pp. 4448–4452, 2017.

17. D. Files and A. H. Hadi, “Analyzing and Detecting Malicious Content :,” vol. 14, no. 8, pp. 404–412, 2016.
18. J. Hk, C. Warty, and S. Singh, “Threat Analysis and Malicious User Detection in Reputation Systems using Mean Bisector Analysis and Cosine Similarity (MBACS),” pp. 0–5, 2013.
19. J. Dechaux, E. Filiol, and J. Fizaine, “Office Documents : New Weapons of Cyberwarfare,” no. August 2007.
20. M. Mimura, Y. Otsubo, and H. Tanaka, “Evaluation of a Brute Forcing Tool that Extracts the RAT from a Malicious Document File,” Proc. - 11th Asia Jt. Conf. Inf. Secur. AsiaJCIS 2016, pp. 147–154, 2016.
21. M. Iwamoto, S. Oshima, and T. Nakashima, “A Study of Malicious PDF Detection Technique,” Proc. - 2016 10th Int. Conf. Complex, Intelligent, Softw. Intensive Syst. CISIS 2016, pp. 197–203, 2016.
22. S. R. Barre and S. U. D. N, “Detecting Targeted Malicious Email Through Mail Client,” Comput. Commun., pp. 2261–2264, 2014.
23. T. Schreck, S. Berger, and J. Göbel, “BISSAM: Automatic vulnerability identification of office documents,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 7591 LNCS, pp. 204–213, 2013.
24. S. Kim, S. Hong, J. Oh, and H. Lee, “Obfuscated VBA Macro Detection Using Machine Learning,” 2018 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, pp. 490–501, 2018.
25. S. Education, “Malware Analysis using Profile Hidden Markov Models and Intrusion Detection in a stream learning setting,” no. May, 2014.
26. N. Nissim, A. Cohen, and Y. Elovici, “Boosting the detection of malicious documents using designated active learning methods,” Proc. - 2015 IEEE 14th Int. Conf. Mach. Learn. Appl. ICMLA 2015, pp. 760–765, 2016.
27. J. Dechaux and E. Filiol, “Proactive defense against malicious documents: Formalization, implementation and case studies,” J. Comput. Virol. Hacking Tech., vol. 12, no. 3, pp. 191–202, 2016.
28. A. F. B, L. Pan, M. Abdelrazek, and R. Doss, Identifying Drawbacks in Malicious PDF. Springer International Publishing, 2018.
29. M. Xu and T. Kim, “PlatPal : Detecting Malicious Documents with Platform Diversity This paper is included in the Proceedings of the,” 2017.
30. M. Mimura and H. Tanaka, “Long-term Performance of a Generic Intrusion Detection Method Using Doc2vec,” 2017.