



DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to VTU, Belagavi, Approved by AICTE & ISO 9001:2008 Certified)

Accredited by National Assessment & Accreditation Council (NAAC) with 'A' grade,

Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078.



A Mini-Project Report On “Malware Analysis Using Machine learning”

Submitted in the partial fulfillment of requirements for the award of degree of

BACHELOR OF ENGINEERING IN INFORMATION SCIENCE AND ENGINEERING

Submitted by

PAVAN KUMAR N(1DS19IS067)

PRAJWAL N(1DS19IS069)

PRAMOD J M(1DS19IS070)

ARIF HUSSAIN(1DS20IS401)

Under the guidance of

(Dr. CHANDRAKALA B M)
(Associate Professor)

Department Of Information Science and Engineering

DAYANANDA SAGAR COLLEGE OF ENGINEERING

S M Hills, Kumara Swamy Layout, Bengaluru-560078

2021-22

DAYANANDA SAGAR COLLEGE OF ENGINEERING

Shavige Malleshwara Hills, Kumaraswamy Layout
Bengaluru-560078.

Department of Information Science and Engineering

ACCREDITED BY NBA & NAAC



CERTIFICATE

This is to certify that Mini-Project Work entitled “**MALWARE ANALYSIS USING MACHINE LEARNING**” is a bonafide work carried out by **PAVAN KUMAR N[1DS19IS067], PRAJWAL N[1DS19IS069], PRAMOD JM[1DS19IS70], ARIF HUSSAIN [1DS20IS401]** in partial fulfillment for the 6th semester of Bachelor of Engineering in Information Science & Engineering of the Visvesvaraya Technological University, Belgaum during the year 2021-22. The Mini-Project Report has been approved as it satisfies the academic requirements prescribed for the Bachelor of Engineering degree.

Signature of Guide

[Dr. CHANDRAKALA B M]

Signature of HOD

[Dr. Udaya Kumar Reddy K R]

Name of the Examiners

Signature with Date

1.

2.

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of a large number of individuals who have been responsible for the successful completion of this Mini-Project.

We take this opportunity to express our sincere gratitude to **Dayananda Sagar College of Engineering** for having provided us with a great opportunity to pursue our Bachelor's Degree in this institution.

In particular, we would like to thank **Dr.C P S Prakash**, Principal, Dayananda Sagar College of Engineering for his constant encouragement and advice.

Special thanks to **Dr. Udaya Kumar Reddy K R**, HOD, Department of Information Science & Engineering, Dayananda Sagar College of Engineering for his motivation and invaluable support through the development of this Mini-Project.

We are highly indebted to our internal guide **Dr. Chandrakala B M, Associate Professor**, Department of Information Science & Engineering, Dayananda Sagar College of Engineering for constant support and guidance.

Finally, we gratefully acknowledge the support of our families during the completion of the project.

PAVAN KUMAR N (1DS19IS067)

PRAJWAL N (1DS19IS069)

PRAMOD J M (1DS19IS070)

ARIF HUSSAIN (1DS20IS401)

ABSTRACT

This paper aims to present the functionality and accuracy of five different machine learning algorithms to detect whether an executable is infested or clean. The phenomenon of Malware include software programs or pieces of code that aim to hijack computer systems to steal information or to destroy it. Machine Learning has its benefits in addressing this situation. Machine learning is widely used in this field by antivirus and antimalware programs as well as by these malicious programs, for example Polymorphic Malware uses machine learning algorithms to encrypt itself in a differently each time it infests a new mass, becoming increasingly difficult to detect. Then we deals with understanding PE files and its structure using pefile module.

CONTENTS

Chapter 1 Introduction.....	Pg No. 1
1.1 Problem statement	
1.2 Objectives and Scope of Project	
1.3 Motivation of Project	
Chapter 2 Literature Survey.....	Pg No. 3
Chapter 3 Requirements	Pg No. 4
3.1 Software Requirements	
3.2 Hardware Requirements	
Chapter 4 System Analysis	Pg No. 5
4.1 Malware Analysis Techniques	
4.2 Malware Detection Techniques	
Chapter 5 System Design	Pg No. 7
5.1 Introduction	
5.2 Architecture diagram	
5.3 Flow chart diagram	
Chapter 6 Modules	Pg No.9
6.1 Malware Definition	
6.2 Machine Learning	
Chapter 7 Pseudocode	Pg No.18
Chapter 8 Testing	Pg No.21

8.1 Test cases

Chapter 9 ResultsPg No.22

Chapter 10 Conclusion and Future Scope.....Pg No.23

ReferencesPg No.24

Chapter 1

Introduction

"Malware" is an abbreviation for "malicious software", it is used as a single term to refer to Viruses, Trojans, Worms, etc. These programs have a variety of features, such as stealing, encrypting or deleting sensitive data, modifying or hijacking basic computer functions, and monitoring computer activity. show user permission. Malware detection techniques are used to detect malware and prevent its infestation of the system, protecting it from potential information loss and compromising the system. Machine Learning is a category of algorithms that allow software applications to predict much better results without being specifically programmed. Malware analysis is necessary for the development of effective techniques for detecting infected files. This analysis is the process of observing the purpose and functionality of a malware program. There are 3 analysis techniques that have the same purpose: to explain how a malware works and what its effects are on the system, but the time and knowledge required are very different.

1.1 Problem Statement

With the growth of technology, the number of malware are also increasing day by day. Malware now are designed with mutation characteristic which causes an enormous growth in number of the variation of malware. Not only that, with the help of automated malware generated tools, novice malware author is now able to easily generate a new variation of malware. With these growths in new malware, traditional signature based malware analysis and detection are proven to be ineffective against the vast variation of malware. On the other hand, machine learning methods for malware analysis are proved effective against new malwares.

1.2 Objective and Scope of the Project

To understand the type of malware and its functionality.

Determine how the system was infected by malware and define if it was a targeted attack or a phishing attack. Is it attempting to stop any service? How malware communicates with attacker.

Future detection of malware and generating signatures. The scope of the project was to know what the malware is attacking, which would help us to step-up our security around those servers or applications or patch vulnerabilities. The primary purpose of the malware analysis project was to identify an investigative solution.

1.3 Motivation of Project

The primary motive behind performing malware analysis is to extract information from the malware sample, which can help in responding to a malware incident. The motive of malware analysis is to determine the capability of malware, detect it, and contain it. To determine the nature and purpose of the malware. For example, it can help you determine whether malware is an information stealer, HTTP bot, spam bot, rootkit, keylogger, or RAT, and so on. To gain an understanding of how the system was compromised and its impact. To determine the attacker's intention and motive.

Chapter 2

Literature Survey

The survey written by Shabtai, “A Behavioural Malware Detection Framework”, 2014, deals with how classifiers are used on static features to detect malware.

Souri and Hosseini, “A State-of-the-art Survey of Malware Detection Approaches”, 2018, proposes a taxonomy of malware detection approaches based on machine learning.

LeDoux and Lakhotia, “Malware and Machine Learning”, 2015, describe how machine learning is used for malware analysis, whose end goal is defined there as “automatically detect malware as soon as possible, remove it, and repair any damage it has done”.

Barriga and Yoo, “Malware Detection and Evasion using Machine Learning”, 2017, briefly survey literature on malware detection and malware evasion techniques, to discuss how machine learning can be used by malware to bypass current detection mechanisms.

Signature-based detection is a process in which a unique identifier about a threat is established, it is known, so the threat can be detected. identified in the future. In the case of a virus scan, this can be a unique code template that attaches to a file, or it can be as simple as the hash of a bad file. known. If that specific pattern or signature is rediscovered, the file may be reported as infected. As malware has become more sophisticated, malware authors have begun to use new techniques, such as polymorphism, to change the pattern each time the object has spread from one system to another. As such, a simple model fit would not be useful beyond a “small handful” of discovered devices. Unlike signature-based scanning, which shows If the signatures found in the file match that of a known malware database, the heuristic scan [5] uses rules and / or algorithms to search for commands that may indicate intent, evil. Using this method, some heuristic scanning methods are able to detect malware without the need for a signature. This is why most antivirus programs use both signature and heuristic methods in combination to capture any malware that might try to evade detection.

Feature detection is a derivative of behavior-based detection that attempts to overcome the typical rate of false alarms associated with it. Characteristic detection is based on program characteristics that describe the security behavior of critical programs. This involves monitoring program executions and detecting deviations from the specification and its behavior, instead of seemingly detecting specific attack patterns. This technique is similar to the detection of anomalies, different, being that it is based on features developed manually to capture the behavior of the system instead of relying on machine learning techniques. The advantage of this technique is that it can instantly detect known and unknown malware, and the level of false positives is lower, but the level of false negatives is high and not as effective as behavioral detection. We will be using this technique.

Chapter 3

Requirements

3.1 Software Requirements:

- PYTHON
- PESTUDIO
- VSCODE AS IDE

3.2 Hardware Requirements:

- OS : WINDOWS 7, 8, AND 10 (32 AND 64 BIT)
- RAM :4GB

Chapter 4

System Analysis

4.1 Malware Analysis Techniques

Malware analysis is necessary for the development of effective techniques for detecting infested files. This analysis is the process of observing the purpose and functionality of a malware program. There are 3 analysis techniques that have the same purpose: to explain how a malware works and what its effects are on the system, but the time and knowledge required are very different.

Static analysis

It is also called code analysis. That is, the malware software code is observed to gain knowledge about the operation of malware functions. This reverse engineering technique is performed using disassembly, decompilation, debugging, and source code analysis tools. We will be following this technique as it is free from the overhead of execution time.

Dynamic analysis

It is also called behavioral analysis. Infected files are analyzed during execution in an isolated environment such as a virtual machine, simulator, or emulator. After the execution of the file, the behavior and its effects on the system are monitored.

Hybrid analysis

This technique is proposed to overcome the limitations of static and dynamic analysis. First, it analyses the specification of the signature for any malware code and then combines it with the other behavioural parameters to improve the complete analysis of malware. Due to this approach, hybrid scanning exceeds the limits of static and dynamic scans

4.2 Malware Detection Techniques

Malware detection techniques are used to detect malware and prevent its infestation of the system, protecting it from potential information loss and compromising the system. They are categorized into: signature detection, behaviour detection and feature detection.

Signature detection

Signature-based detection is a process in which a unique identifier about a threat is established, it is known, so the threat can be detected. identified in the future. In the case of a virus scan, this can be a unique code template that attaches to a file, or it can be as simple as the hash of a bad file. known. If that specific pattern or signature is rediscovered, the file may be reported as infected. As malware has become more sophisticated, malware authors have begun to use new techniques, such as polymorphism, to change the pattern each time the object has spread from one system to another. As such, a simple model fit would not be useful beyond a “small handful” of discovered devices.

Behaviour Detection

Unlike signature-based scanning, which shows If the signatures found in the file match that of a known malware database, the heuristic scan [5] uses rules and / or algorithms to search for commands that may indicate intent, evil. Using this method, some heuristic scanning methods are able to detect malware without the need for a signature. This is why most antivirus programs use both signature and heuristic methods in combination to capture any malware that might try to evade detection.

Feature detection

Feature detection is a derivative of behaviour-based detection that attempts to overcome the typical rate of false alarms associated with it. Characteristic detection is based on program characteristics that describe the security behaviour of critical programs. This involves monitoring program executions and detecting deviations from the specification and its behaviour, instead of seemingly detecting specific attack patterns. This technique is similar to the detection of anomalies, different, being that it is based on features developed manually to capture the behaviour of the system instead of relying on machine learning techniques. The advantage of this technique is that it can instantly detect known and unknown malware, and the level of false positives is lower, but the level of false negatives is high and not as effective as behavioural detection. We will be using this technique.

Chapter 5

System Design

5.1 Introduction

A serious threat today is malicious executables. It is designed to damage computer system and some of them spread over network without the knowledge of the owner using the system. Two approaches have been derived for it i.e. Signature Based Detection and Heuristic Based Detection. These approaches performed well against known malicious programs.

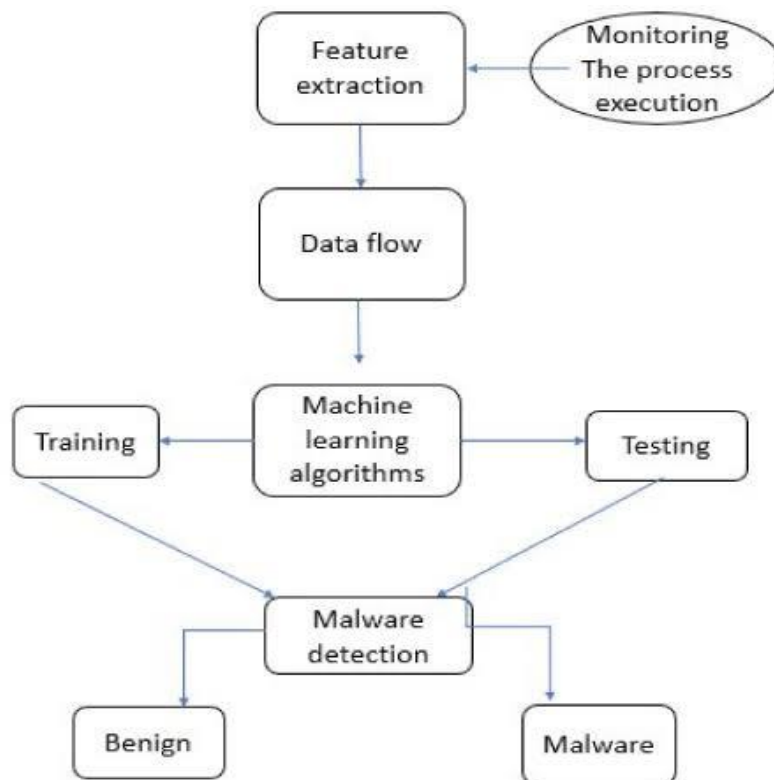
PE stands for portable executable. It is the native executable format of Win32. Its specification is derived somewhat from the Unix COFF (common object file format). The format information of PE file is illustrated in Figure 8. It is basically a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code.

To train the data in supervise learning we first use pandas module to load the .csv file into our VScode IDE.

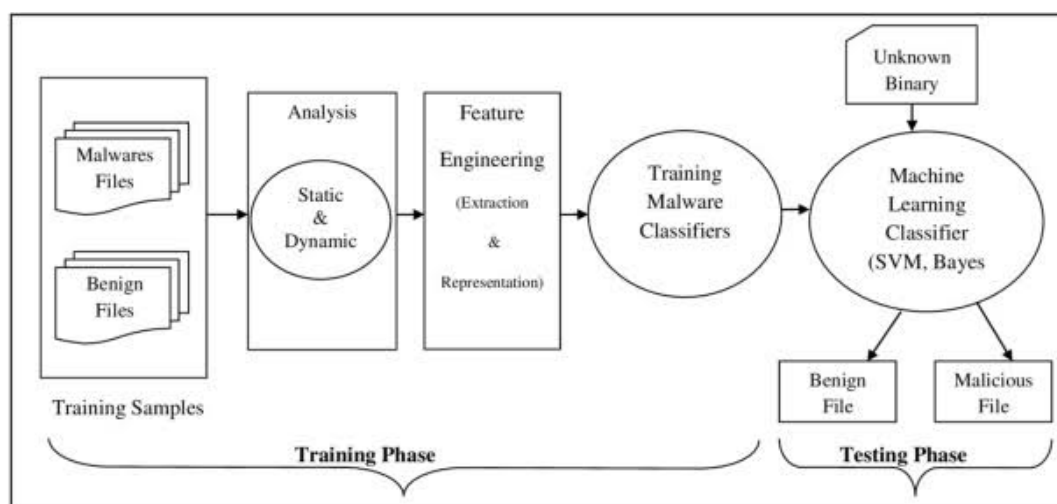
Extra Trees Classifier is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output it’s classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest.

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set $X_{\text{test}}, y_{\text{test}}$. Note that the word “experiment” is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally. Here is a flowchart of typical cross validation workflow in model training. The best parameters can be determined by grid search techniques.

5.2 Flow chart



5.3 System Architecture



Chapter 6

Modules

6.1 Malware Definition:

"Malware" is an abbreviation for "malicious software", it is used as a single term to refer to Viruses, Trojans, Worms, etc. These programs have a variety of features, such as stealing, encrypting or deleting sensitive data, modifying or hijacking basic computer functions, and monitoring computer activity. show user permission.

6.1.1 Types of Malware:

Computer virus

It is generally a program that is installed outside the user's will and can cause damage to both the operating system and the hardware (physical) elements of a computer.

Effects generated by the virus:

- destruction of files.
 - changing the file size.
 - Delete all information on the disc, including formatting it.
 - destruction of the file allocation table, which makes it impossible to read the information on the disk.
 - various harmless but disturbing graphic / sound effects
 - slowing down the computer's working speed until it crashes
- Worms
- Computer worms are programs with destructive effects that use communication between computers to spread. Worms have common features with viruses, ii. Worms are able to multiply like viruses, but not locally, but on other computers. I use computer networks to spread to other systems.

Types of computer worms:

- E-Mail worms
 - Instant messaging worms
 - Internet worms
 - IRC worms
 - File-sharing files on the network
- Trojan horses
- Trojan horses are "disguised" programs that try create gaps in the operating system to allow a user to access the system. Trojans do not have the facility to self-multiply like computer viruses.
- Trojan horses can be divided into several categories:
 - backdoors: allows the attacker to take control of the victim's computer via the Internet;

- password stealer: programs that steal passwords (read data from the keyboard and store them in files that can be read later by the attacker or can be sent directly to the e-mail account). mail of the attacker);
- logical bombs: when certain conditions are met these Trojans can perform operations that compromise system security;
- Denial of Service tools: programs that send certain sequences of data to the target audience (usually a website), with the intention of interrupting the Internet services of that target. Ransomware Ransomware is a type of malware that blocks the victim's access to the computer and demands payment of a reward. The reward and the official reason why the victim should pay depends on the type of virus. Some versions of ransomware claim that the payment should be made to avoid punishment by a government authority (usually the FBI or a local agency), others inform that this is the only way to decrypt encrypted data. Effects generated by ransomware:
 - are able to encrypt sensitive user data
 - can delete predetermined documents, multimedia objects and any other files that contain important information. They can also try to delete essential components of the system or important parts of other software.
- Ransomware threats can be used to steal authentication names, passwords, valuable personal documents, identity data and other confidential information
- can quickly terminate the activity of an anti-virus, anti-spyware or any other software blocking its processes and disabling essential services in the system. root kit 7 A rootkit software is generally a program that manages through a vulnerability of the host system to get full rights on a system, a system that modifies it so that it can use its resources undetected.
- can modify the “ps” utility on a Linux system, a utility that displays active processes, to make it NOT display the rootkit process
- It can hide certain files (usually their own) in case an antivirus program scans Spyware is a category of cyber threats, which describes malware created to infect PC systems and then initiate illegal activities. these. In most cases, the functionality of these threats depends on the intentions of their vendors: some parts of spyware threats can be used to collect personal information (login names, passwords, and other personally identifiable data) and send them to their owners via hidden internet connections, while other spyware viruses can track their victims and collect information about their browsing habits. These are used to track people and record their most visited websites as well as the actions taken when they were visited. This information is generally used by various third parties for marketing and promotional purposes, so spyware can also lead to an increase in the number of spams.

What spyware can be used for:

- To steal sensitive information. Such programs are interested in personal information, such as credentials, passwords, bank details, and other

similar information. In addition, they can monitor the user's online activity, track their web browsing habits, and send all this data to a remote server.

- Show unwanted creatives. Spyware can display a large number of annoying pop-up ads. Such activity is more associated with adware parasites.
- Redirecting users to questionable or malicious websites against their will. In addition, some types of spyware threats are able to change web browser settings and change the search engine and home page.
- Create numerous links in the search results of the victim and redirect him / her to the desired places (third party spyware sites, websites and other associated fields).
- Cause essential changes to system settings. These changes can reduce overall security and trigger performance issues.
- Connecting to a compromised computer using backdoors. Most spyware threats are capable of giving hackers remote access to the system without the user's knowledge.

Degradation of the overall performance of the system and causing its instability.

6.2 Machine Learning

6.2.1 Definition:

Machine Learning is a category of algorithms that allow software applications to predict much better results without being specifically programmed. The basic premise of machine learning is to build algorithms that receive input data and use statistical analysis to predict output data while output data is updated like many input data become valid. The processes involved in machine learning are similar to the processes of data mining and predictive modelling. Both require searching for certain patterns by date, and adjusting program actions accordingly. Many people are also familiar with machine learning from internet shopping and the advertisements that are shown to them depending on what they are buying. This is because referral engines use machine learning to customize ads that are delivered online in near real time. In addition to personalized marketing, other well-known cases in which machine learning is used are fraud detection, spam filtering, threat detection of countries in the network, maintenance, predictability, and building the flow of news.

6.2.1.1 How machine learning works:

Machine learning algorithms are categorized as both supervised and unsupervised.

Supervised algorithms

They require a data researcher, or data analyst, who has the knowledge of machine learning to supply the desired input and output data, in addition to delivering

feedback on the accuracy of the predictions; acute during algorithm training. Data researchers determine which variables, or characteristics, should be analysed by the model and used to develop predictions. Once the training is complete, the algorithm will apply what it has learned to new data. Supervised learning problems can be further grouped into regression and classification problems. Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”. Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”. Some common types of problems built on top of classification and regression include recommendation and time series prediction respectively. Some popular examples of supervised machine learning algorithms are: Linear regression for regression problems. Random forest for classification and regression problems, Support vector machines for classification problems.

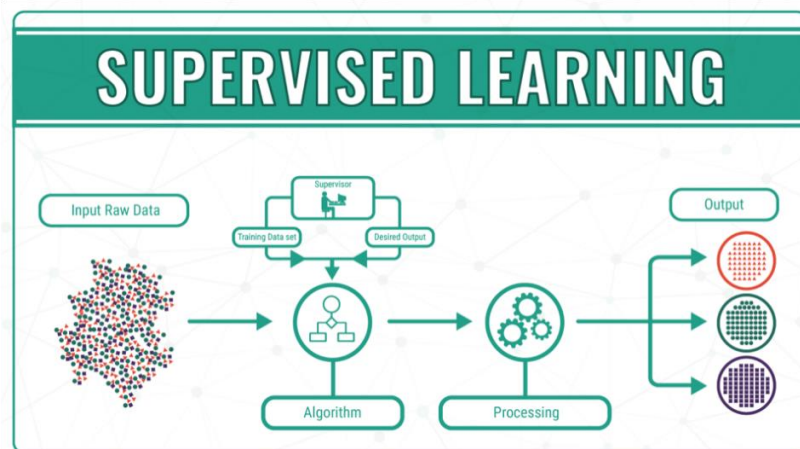
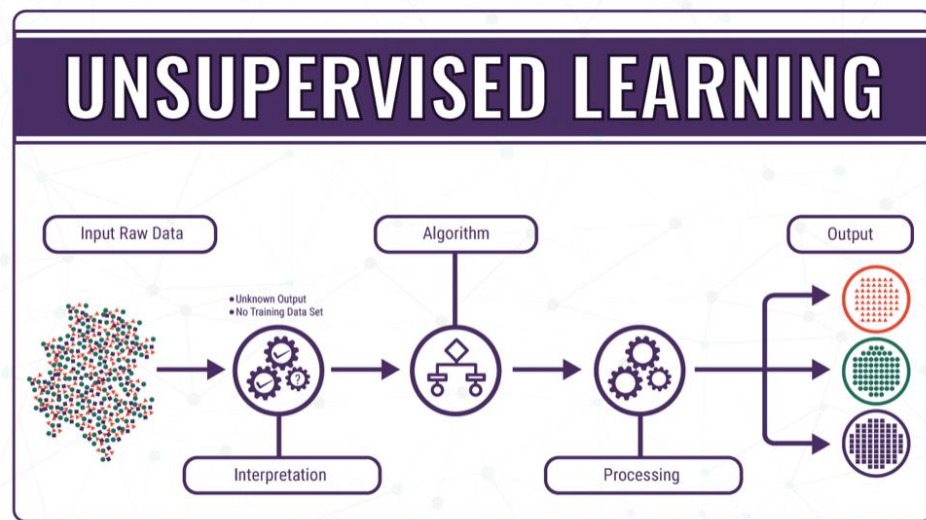


Figure 1 Supervised learning

Unsupervised algorithms

They do not need training with output data. Instead, they use a method called deep learning to review the data and come to conclusions. Unsupervised and learned algorithms, also known as neural networks, are used for more complex processes than supervised algorithms, which include image recognition, speech-to-text, and natural language generation. These neural networks work by first combining millions of training examples with data and automatically identifying subtle correlations between multiple variables. Once trained, the algorithm can be used by associates to interpret new data. These algorithms become feasible only in the information age, because they require massive amounts of data to train. These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devices to discover and present the interesting structure in the data. Unsupervised learning problems can be further grouped into clustering and association problems. Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior. Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y. Some popular examples of unsupervised learning algorithms

are:k-means for clustering problems.,Apriori algorithm for association rule learning problems.



6.2.1.1 Algorithms used in this paper:

Random Forest

An effective alternative is to use trees with fixed structures and random features . Tree collections are called forests, and classifiers built in so-called random forests. The random water formation algorithm requires three arguments: the data, a desired depth of the decision trees, and a number K of the total decision trees to be built, i. The algorithm generates each of the K trees. independent, which makes it very easy to parallelize. For each tree, build a complete binary tree. The characteristics used for the branches of this tree are selected randomly, usually with replacement, which means that the same characteristic can occur more than 20 times, even in a single branch. a. the leaves of this tree, where predictions are made, are completed based on training data. The last step is the only point at which the training data is used.

The resulting classifier is just a K-lot vote, and random trees. The most amazing thing about this approach is that it actually works remarkably well. They tend to work best when all the features are at least, well, relevant, because the number of features selected for a particular tree is small. One intuitive reason that it works well is the following. Some trees will query unnecessary features. These trees will essentially make random predictions. But some of the trees will happen to question good characteristics and make good predictions (because the leaves are estimated based on training data).

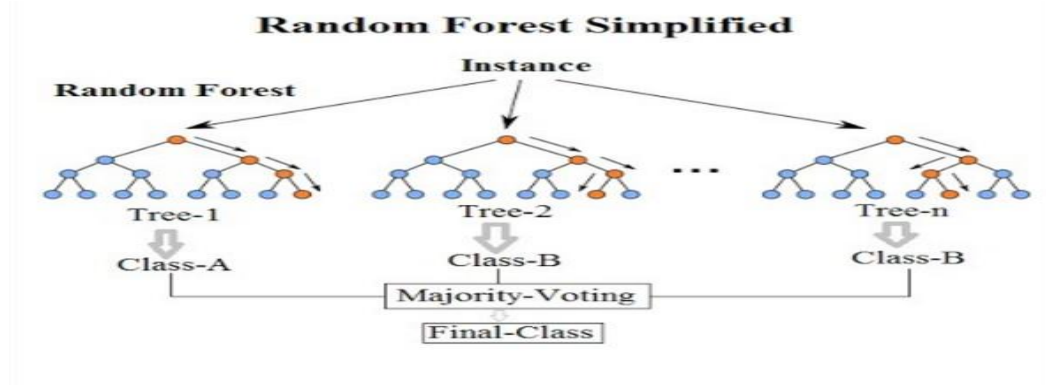


Figure 2 Random forest view

ADA Boost

Ada-boost or Adaptive Boosting is one of ensemble boosting classifier proposed by Yoav Freund and Robert Schapire in 1996. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations. Any machine learning algorithm can be used as base classifier if it accepts weights on the training set.

The classifier should be trained interactively on various weighed training examples. In each iteration, it tries to provide an excellent fit for these examples by minimizing training error. Initially, Adaboost selects a training subset randomly. It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training. It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification. Also,

It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight. This process iterate until the complete training data fits without any error or until reached to the specified maximum number of estimators. To classify, perform a "vote" across all of the learning algorithms you built.

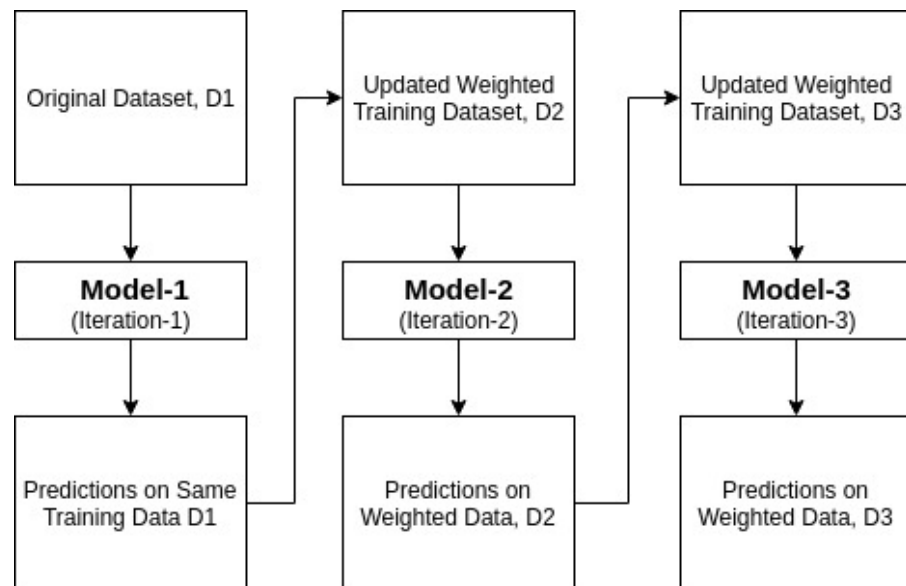


Figure 3 Ada Boost

Gradient boosting

The idea behind "gradient boosting" is to take a weak hypothesis or weak learning algorithm and make a series of tweaks to it that will improve the strength of the hypothesis/learner. This type of Hypothesis Boosting is based on the idea of Probability Approximately Correct Learning (PAC). This PAC learning method investigates machine learning problems to interpret how complex they are, and a similar method is applied to Hypothesis Boosting.

In hypothesis boosting, you look at all the observations that the machine learning algorithm is trained on, and you leave only the observations that the machine learning method successfully classified behind, stripping out the other observations. A new weak learner is created and tested on the set of data that was poorly classified, and then just the examples that were successfully classified are kept.

The Gradient Boosting Classifier depends on a loss function. A custom loss function can be used, and many standardized loss functions are supported by gradient boosting classifiers, but the loss function has to be differentiable. Classification algorithms frequently use logarithmic loss, while regression algorithms can use squared errors. Gradient boosting systems don't have to derive a new loss function every time the boosting algorithm is added, rather any differentiable loss function can be applied to the system.

Gradient boosting systems have two other necessary parts: a weak learner and an additive component. Gradient boosting systems use decision trees as their weak learners. Regression trees are used for the weak learners, and these regression trees output real values. Because the outputs are real values, as new learners are added into the model the output of the regression trees can be added together to correct for errors in the predictions.

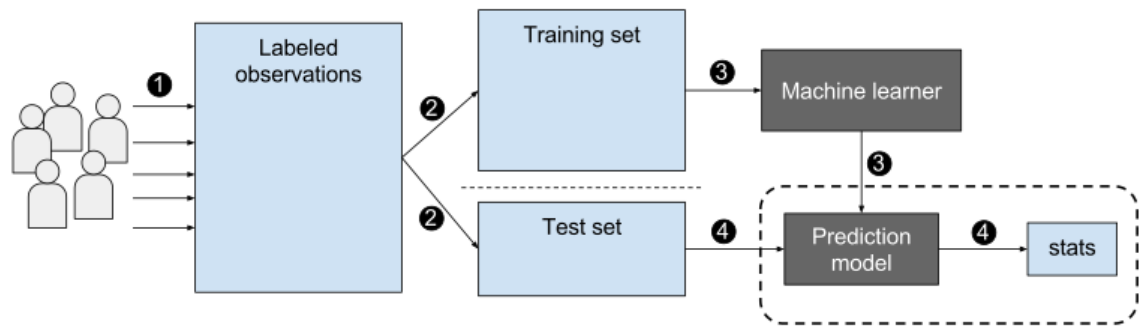


Figure 4 Gradient boost

Decision Tree

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

Decision Tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm. The time complexity of decision trees is a function of the number of records and number of attributes in the given data. The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy.

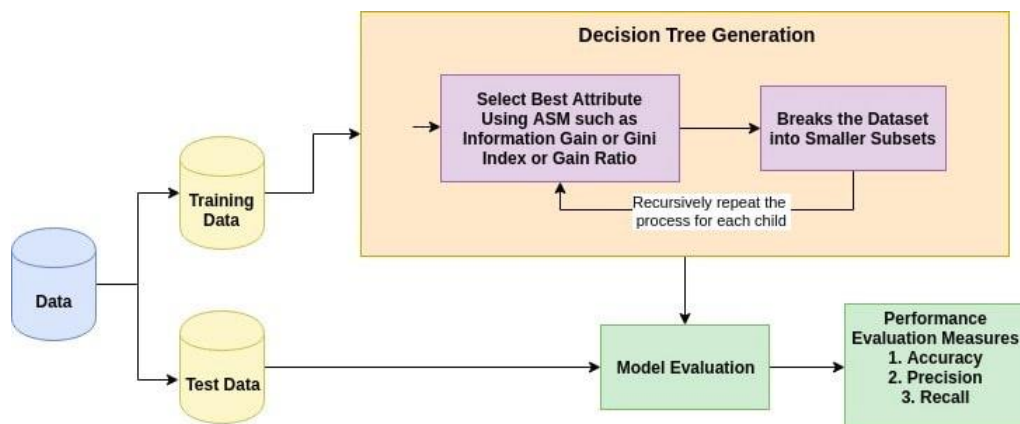


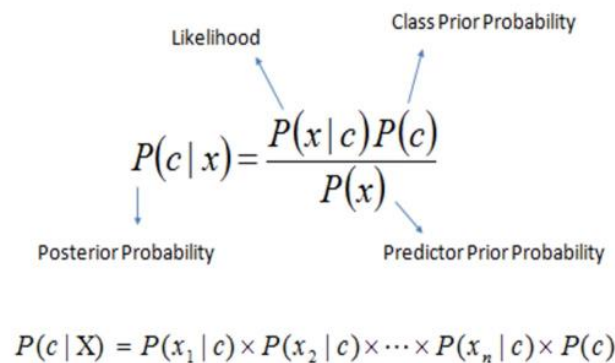
Figure 5 Decision Tree

Naïve Bayes

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that's why it is considered as naive. This assumption is called class conditional independence.

Naive Bayes Classifier



The diagram shows the Naive Bayes formula with labels for its components:

$$P(c | x) = \frac{P(x | c) P(c)}{P(x)}$$

- $P(c | x)$ is labeled as **Posterior Probability** (indicated by a downward arrow).
- $P(x | c)$ is labeled as **Likelihood** (indicated by an upward arrow).
- $P(c)$ is labeled as **Class Prior Probability** (indicated by an upward arrow).
- $P(x)$ is labeled as **Predictor Prior Probability** (indicated by a downward arrow).

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

Figure 6 Naive Bayes

Chapter 07

Pseudocode

There Will be two python scripts learning.py and checker.py . First one is for training and creating machine learning model and second one is for implementing the model on real time files

```
dataset=pd.read_csv(r'data.csv',sep='|')
dataset.groupby(dataset['legitimate']).size()
dataset
X = dataset.drop(['Name','md5','legitimate'],axis=1).values
y = dataset['legitimate'].values
extratrees = ek.ExtraTreesClassifier().fit(X,y)
model = SelectFromModel(extratrees, prefit=True)
X_new = model.transform(X)
nbfeatures = X_new.shape[1]
print(nbfeatures)
X_train, X_test, y_train, y_test = train_test_split(X_new, y
,test_size=0.2)
features = []
index = numpy.argsort(extratrees.feature_importances_)[::-1][:nbfeatures]
for f in range(nbfeatures):
    print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]],
extratrees.feature_importances_[index[f]]))
    features.append(dataset.columns[2+f])
model = { "DecisionTree":tree.DecisionTreeClassifier(max_depth=10),
          "RandomForest":ek.RandomForestClassifier(n_estimators=50),
          "Adaboost":ek.AdaBoostClassifier(n_estimators=50),
          "GradientBoosting":ek.GradientBoostingClassifier(n_estimators
=50),
          "GNB":GaussianNB()
}
results = {}
for algo in model:
    clf = model[algo]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s %" % (algo, score*100))
    results[algo] = score
winner = max(results, key=results.get)
joblib.dump(model[winner],'classifier/classifier.pkl')
open('classifier/features.pkl', 'wb').write(pickle.dumps(features))
clf = model[winner]
res = clf.predict(X_new)
mt = confusion_matrix(y, res)
print("False positive rate : %f %" % ((mt[0][1] /
float(sum(mt[0])))*100))
print('False negative rate : %f %' % ( (mt[1][0] /
float(sum(mt[1]))*100))
```



```
# Load classifier
clf = joblib.load('classifier/classifier.pkl')
#load features
features = pickle.loads(open(os.path.join('classifier/features.pkl'), 'rb').read())
features
pe_df = dataset[['Machine',
'SizeOfOptionalHeader',
'Characteristics',
'MajorLinkerVersion',
'MinorLinkerVersion',
'SizeOfCode',
'SizeOfInitializedData',
'SizeOfUninitializedData',
'legitimate']]
pe_df
pe_df.to_csv("final_pe_data.csv", index = False)
```

Entropy calculation

In general words, entropy is referred as the measurement of particular data in digital values. Similar to this, the term File Entropy is the representation of data sets in specific file. That is, the phrase File Entropy is used to measure the amount of data which is present in a selected file. File Entropy is also use in the field of malware protection, in the process of malware analysis as there are all kind of security related tools that you check on the file to extract all kind of information from the file, to determine if the file is a malware or legit file, and if it is a malware this can be useful on the malware file entropy can be a useful method to quickly check if the malware file had been packed with one of the packed software it is also a good method to check if the file encrypted by one of the encryption algorithm

```
def get_entropy(data):
    if len(data) == 0:
        return 0.0
    occurrences = array.array('L', [0]*256)
    for x in data:
        occurrences[x if isinstance(x, int) else ord(x)] += 1
    entropy = 0
    for x in occurrences:
        if x:
            p_x = float(x) / len(data)
            entropy -= p_x*math.log(p_x, 2)
    return entropy
```

1.def get_resources(peFile)- to Extract resources [entropy, size]

```
def get_resources(pe):
    """Extract resources :
    [entropy, size]"""
    resources = []
    if hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'):
        try:
            for resource_type in pe.DIRECTORY_ENTRY_RESOURCE.entries:
                if hasattr(resource_type, 'directory'):
                    for resource_id in resource_type.directory.entries:
                        if hasattr(resource_id, 'directory'):
                            for resource_lang in
resource_id.directory.entries:
                                data =
pe.get_data(resource_lang.data.struct.OffsetToData,
resource_lang.data.struct.Size)
                                size = resource_lang.data.struct.Size
                                entropy = get_entropy(data)

                                resources.append([entropy, size])
        except Exception as e:
            return resources
    return resources
```

2.def get_version_info(pe)- Returns version information of test.exe file

```
def get_version_info(pe):
    """Return version infos"""
    res = {}
    for fileinfo in pe.FileInfo:
        if fileinfo.Key == 'StringFileInfo':
            for st in fileinfo.StringTable:
                for entry in st.entries.items():
                    res[entry[0]] = entry[1]
        if fileinfo.Key == 'VarFileInfo':
            for var in fileinfo.Var:
                res[var.entry.items()[0][0]] = var.entry.items()[0][1]
    if hasattr(pe, 'VS_FIXEDFILEINFO'):
        res['flags'] = pe.VS_FIXEDFILEINFO.FileFlags
        res['os'] = pe.VS_FIXEDFILEINFO.FileOS
        res['type'] = pe.VS_FIXEDFILEINFO.FileType
        res['file_version'] = pe.VS_FIXEDFILEINFO.FileVersionLS
        res['product_version'] = pe.VS_FIXEDFILEINFO.ProductVersionLS
        res['signature'] = pe.VS_FIXEDFILEINFO.Signature
        res['struct_version'] = pe.VS_FIXEDFILEINFO.StrucVersion
    return res
```

Chapter 08

Testing

8.1 Test case

Extra Trees Classifier is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output it’s classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest.

Now converting in training and testing data in 20% range ! as total x is 138047 and testing is $138047 * 0.2 = 27610$

Sorting Important features for more accuracy

```
... 13
1. feature DllCharacteristics (0.147793)
2. feature VersionInformationSize (0.104540)
3. feature Characteristics (0.093933)
4. feature Machine (0.089670)
5. feature SectionsMaxEntropy (0.064559)
6. feature Subsystem (0.056836)
7. feature SizeOfOptionalHeader (0.055695)
8. feature MajorSubsystemVersion (0.051040)
9. feature ImageBase (0.049714)
10. feature ResourcesMinEntropy (0.036492)
11. feature ResourcesMaxEntropy (0.024660)
12. feature SectionsMeanEntropy (0.021214)
13. feature MajorOperatingSystemVersion (0.020372)
DecisionTree : 99.08366533864542 %
RandomForest : 99.43860919956538 %
Adaboost : 98.4425932633104 %
GradientBoosting : 98.81564650488953 %
GNB : 69.8804780876494 %
False positive rate : 0.097184 %
False negative rate : 0.164557 %
```

Chapter 09

Result

malware_test.py which test the exe file is legitimate or malicious

```
%run malware_test.py "BlueStacksInstaller_5.3.130.1003_native_314e3224e3ed67bc63e8a2eefec781a9_0_QkFUVExFR1JPVU5EUyBNT0JJTEUgSU5ESUE=.exe"
✓ 0.2s
The file BlueStacksInstaller_5.3.130.1003_native_314e3224e3ed67bc63e8a2eefec781a9_0_QkFUVExFR1JPVU5EUyBNT0JJTEUgSU5ESUE=.exe is legitimate

%run malware_test.py "AnyDesk.exe"
✓ 0.3s
The file AnyDesk.exe is malicious
```

Chapter 10

Conclusion

The aim of this paper is to present a machine learning approach to the malware problem. Due to the sudden growth of malware, we need automatic methods to detect infested files. In the first phase of the work, the data set is created using infested and clean executables, in order to extract the data necessary for the creation of the data set, we used a script created in Python. After creating the data set, it must be ready to train machine learning algorithms. The algorithms used are: decision trees, Random Forest, Naive Bayes, Gradient Boost and ADA Boost presented comparatively. After applying the best accuracy algorithms, it had an Random Forest algorithm with an accuracy of 99.406012 %. This work demonstrates that Random Forest is the best algorithm for detecting malicious programs. In the future, this accuracy can be improved, if we add a much larger number of files in the data set to drive the algorithms. Each algorithm has several parameters that can be tested with different values to increase their accuracy. This project can reach the application level with the help of a library called pickle, to save what the algorithm has learned and then we can test a new file to see if it is clean or infected. Static analysis has also proven to be safer and free from the overhead of execution time.

References

- Malware Types and Classifications, Bert Rankin, 28.03.2018, published in LastLine, last accessed 12.09.2018.
- A Brief History of Malware - Its Evolution and Impact, Bert Rankin, 05.04.2018, published in LastLine, last accessed 12.09.2018.
- Detecting malware through static and dynamic techniques, Jeremy Scott, 14.09.2017, published in NTT Security, last accessed 12.09.2018.
- Hybrid Analysis and Control of Malware, Kevin A. Roundy and Barton P. Miller, International Workshop on Recent Advances in Intrusion Detection, pp. 317-338, 2010, Springer.
- Advanced Malware Detection - Signatures Vs. Behavior Analysis John Cloonan Director of Products, Lastline, 11.04.2017, published in Infosecurity Magazine, last accessed 12.09.2018.
- What is Machine Learning? Daniel Faggella, 12.08.2017, published in techemergence, last accessed 12.09.2018.
- Data mining, Margaret Rouse, Search SQL Server last accessed 12.09.2018, the article can be found here. <https://searchsqlserver.techtarget.com/definition/data-mining> [8]
- Supervised and Unsupervised Machine Learning Algorithms, Jason Brownlee, 16.03.2016, published in Machine Learning Algorithms, last accessed 12.09.2018.
- Decision trees, scikit-learn.org last accessed 12.09.2018.
- RandomForestClassifier, scikit-learn.org last accessed 12.09.2018.
- GradientBoostingClassifier, scikit-learn.org last accessed 12.09.2018.
- Malware Researcher's Handbook, Resources Infosecinstitute, last accessed 12.09.2018.