MALWARE DETECTION USING MACHINE LEARNING TECHNIQUES

A report submitted in partial fulfillment of the requirements

Of

Mini-Project (ISL64)

In

Sixth Semester

By

1MS17IS047 HARSHITA S

1MS17IS053 KRITIKA CHOUDHARY

1MS17IS057 LOHITH R

1MS17IS075 NIKHIL KUMAR

Under the guidance of

George Philip C
Associate Professor
Dept. of ISE, RIT

RAMAIAH
Institute of Technology

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

RAMAIAH INSTITUTE OF TECHNOLOGY

(AUTONOMOUS INSTITUTE AFFILIATED TO VTU)

M. S. RAMAIAH NAGAR, M. S. R. I. T. POST, BANGALORE – 560054

2019-2020

RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute Affiliated to VTU)

M. S. Ramaiah Nagar, M. S. R. I. T. Post, Bangalore – 560054

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

RAMAIAH
Institute of Technology

CERTIFICATE

This is to certify that the project work entitled **"Malware Detection using Machine Learning Techniques"** is a bonafide work carried out by **HARSHITA S, KRITIKA CHOUDHARY, LOHITH R and NIKHIL KUMAR** bearing **USN: 1MS17IS045, 1MS17IS053, 1MS17IS057, 1MS17IS075** in partial fulfillment of requirements of Mini-Project (ISL64) of Sixth Semester B.E. It is certified that all corrections/suggestions indicated for internal assessment has been incorporated in the report. The project has been approved as it satisfies the academic requirements in respect of project work prescribed by the above said course.

_____	_____
Signature of the Guide	Signature of the HOD
George Philip C	**Dr. Vijaya Kumar B P**
Associate Professor	Professor and Head,
Dept. of ISE, RIT,	Dept. of ISE, RIT
Bangalore-54	Bangalore-54

Other Examiners

Name of the Examiners: **Signature**

1.

2.

Acknowledgement

The combined work done by the group members in making this project was deliberate and exacting. The whole theory of this project was a positive learning which would not have been ebbed any other way. We managed many perplexities and perpetuated the best result of our goal in accomplishment of the successful project. The homage goes to our supervisor, guide, mentor, George Philip C, Associate Professor for his abet on residing us through this pompous project with regular guidance and support. We would also like to bestow our gratitude to Dr. N V R Naidu, Principal, Dr. Vijaya Kumar B P, Professor and Head, Department of Information Science and Engineering for their benign support and permission to use facilities from libraries and other college resources when needed. The variety of aspects to deal with the information and learning was also an effort gained by the group through the help of consistent guidance by our mentor. We are immensely intrigued by the faculties that helped us in manifold ways. The indomitable effort by the group itself is overwhelming and to work in such boundaries of guidance were fascinating.

Abstract

Malware has become a major threat to computer security with the increase in the number of systems connected to the internet. The types of malware have increased over the years and new malware has emerged, which are capable of evading conventional detection systems through obfuscations. Currently employed signature-based methods cannot provide accurate detection of these attacks and polymorphic viruses. Signature based detection techniques scan for the specific sequence of bytes (i.e. signature) to detect malware, whereas non-signature based techniques detect malware by observing behaviors and patterns against predefined profile. Signature based detection has high detection accuracy but is limited to the signature database, hence requires frequent updates. That is why the need for machine learning-based detection arises. This project aims to propose a versatile framework in which one can employ different machine learning algorithms to successfully distinguish between malware files belonging to different families and benign files of different formats, while aiming to minimise the number of false negatives.

Contents

List of Figures

Chapter 3:

Chapter 4:

Chapter 5:

List of Tables

Chapter 3:

Chapter 5:

Chapter 1
Introduction

Malware is a harmful software (viruses, worms, Trojan horses, rootkits, botnets,backdoors, and alternative malicious software) that pretends to be a legitimate program to infiltrate the system. Hackers build malware presentable to persuade users into downloading them. Often, the users are unaware that the program is malware as it looks legitimate. Once installed, the malware hides in numerous folders within the system. If it's a sophisticated form of malware, it will directly access the software system. Then it starts to encrypt files and record personal data posing strong security and privacy issues to users and infrastructure operators. That's how malware gets downloaded on the system.

1.1 Motivation

The increase of malware that is exploiting the net daily has become a significant threat. Within recent times malware has been spreading at a high rate.Since 2018, worldwide malware attacks have risen by 350% in total. Hence, we've got an approach by using malware analysis combined with data processing tasks to attain effectiveness and potency in detection malware. It acts as an early warning system for the system to secure relating to malware and cyber-attacks. It keeps hackers out of the system and prevents the knowledge from being compromised.

1.2 Scope

The scope of this project includes performing classification, feature extraction and applying machine learning techniques to detect malware and evince accuracy with the least error rate. This study only focuses on malware detection and malware removal or prevention will not be touched upon. A pre-built dataset with the extracted PE features has been used. Creation of the dataset by extracting features from PE files has not been done.

The project presents the results of various automated classification methods based on machine learning and compares them.

1.3 Objectives

The aim of doing this project was to gain a deeper understanding of malwares and the techniques that are currently used for their detection. Another reason for taking up this project was to understand machine learning algorithms using a practical approach. The machine learning process appraises the flexibility of the features that are to be chosen for the working of the project. The best feature elaborates on the closest accurate rate of malware detection. Features may vary for the benefit of the detection and needs to be understood in a staid manner. The right feature will complete the need for the algorithm to attain the best results.

1.4 Proposed Model

The purpose of this project was to build a model that could predict whether a given Windows Portable Executable file is malware or benign statically. A

framework was designed using data mining algorithms to train multiple classifiers on a set of malicious and benign executables to detect new samples. The data was analysed first and then classifiers trained over a subset of the data. A comparative analysis of the algorithms was done using various performance metrics to determine the best one.

1.5 Organisation of Report

In order to explain the developed system, the following sections are covered:

1. **Literature Review** describes the study of the existing systems and techniques taken into account prior to development of the proposed system.
2. **System Analysis and Design** provides a detailed walk through of the system architecture adopted to implement the model, an overview of the system and the modules incorporated into the system. Each module is described in detail.
3. **Modelling and Implementation** provides a deeper insight into the working of the model. The various modules and their interactions are depicted using relevant descriptive diagrams.
4. **Testing** the model to ensure it is bug/error free and all the units give the same result when executed several times. **Results and Discussion** section presents the results obtained on running the model and analyzes them.
5. **Conclusion** about the results obtained after successfully running the model and **Future Scope** of the model is highlighted.

Chapter 2

Literature Review

A Survey on Malware Detection Schemes Using Machine Learning Techniques

Mukesh et. al. [1] attempted to overview and dissect every malware detection system.The study found that there are diverse strategies like authorization based methodology, signature based procedures, and design based systems which are being used in this field .It was desired to have a low false positive ratio. A comparative study of various approaches was presented and it was concluded that none of the approaches were capable of reducing false positive ratio to zero.

Selecting Features to Classify Malware

Raman through his research [2] aimed to identify the portable executable(PE) format specific features that can be used to distinguish clean files from malware files. To identify those,it was assumed that features from different parts of a PE file would be correlated less with each other and more with the class of the file, dirty or clean. A dataset of 100,000 malware and 16,000 clean files was used, seven features were selected as input to various classifiers.

J48 proved to be the best classification algorithm.

Data Mining Methods for Detection of New Malicious Executables

Stolfo et. al. [3] attempted to implement a network-level email filter that uses algorithms to catch malicious executables before users receive them

through their mail.Data set consisting of 4,266 programs was split into 3,265 malicious binaries and 1,001 clean programs. To extract feature information from Windows executables GNU's Bin–Utils were used. Three different data mining algorithms were used to generate classifiers with different features: RIPPER, Naive Bayes, and a Multi-Classifier system. Signature-based method was used for comparison. The Multi-Naive Bayes method had the highest accuracy and detection rate.

Learning the PE Header, Malware Detection with Minimal Domain Knowledge

The goal of Raff et. al's. work [4] was to explore a number of standard data mining techniques to compute accurate detectors for new binaries.It uses neural networks that classify executables without any feature engineering or processing.Five models were evaluated, Extra Random Trees (ET), Random Forests (RF), Logistic Regression on byte n-grams and two neural networks (Fully Connected and LSTM). Three LSTM layers and four Fully Connected layers were used in the model, and tested networks with hidden state sizes of 128, 256, 512, 1024, and 2048 neurons.The study was conducted using 2,40,000 malicious files and 2,37,349 benign files.The results indicated that Fully Connected network performed better than all other methods, and the LSTM performed comparable to the domain knowledge approach at times.

A Novel Approach to Malware Detection using Static Classification

Singha et. al. [5] explored malware detection using function call frequency and opcode frequency as static features and classifiers like J48, Decision Trees , MultiLayer Perceptron and KStar. The study was performed with 1,230 executable files including 800 malware and 430 clean files.With both features combined together it is found that DT gives the best accuracy followed by J48. The results indicated that function call frequency is a more robust feature than opcode frequency.

But the technique was not effective to classify zero day malware and furthermore disassembling for extracting opcodes required a lot of time.

Machine Learning Methods for Malware Detection

Through this work [6] Kaspersky made the following observations:

Have the right data. This is the fuel of machine learning. The data must be representative, relevant to the current malware landscape and correctly labeled when needed. Understand theoretical machine learning and how to apply it to cybersecurity. Understand user needs and implement machine learning into products that help users with their practical needs. Keep detection methods in multi-layered synergy.

Malware Detection Using Machine Learning

The main target of Gavriluț et. al.'s study[7] was to come up with a machine learning framework that generically detects as many malware samples as it can, with the tough constraint of having a zero false positive rate. although we still have a non-zero false positive rate.malware detection via machine learning will not replace the standard detection methods used by anti-virus vendors, but will come as an addition to them**.**Any commercial anti-virus product is subject to certain speed and memory limitations, therefore the most reliable algorithms among those presented here are the cascade one-sided perceptron and and its explicitly mapped variant. Since most AntiVirus products manage to have a detection rate of over 90%, it follows that an increase of the total detection rate of 3% − 4% as the one produced by our algorithms,is very significant

Malware detection and classification using machine learning techniques

Three techniques were presented by Dhiman [8] for malware detection and classification.

First approach uses data mining and machine learning algorithms like k-Nearest Neighbor, Naive Bayes, Support Vector Machine (SVM), J48

decision tree, and Multilayer perceptron (MLP) neural network.For this approach we can measure the performance based on three measure's true positives, false positives and finally we can compare accuracy of different models. Second approach was using image recognition algorithms. For the third approach we can measure performance statistically, based on malicious or benign tests. These statistical measures precision,sensitivity, recall, hit rate, fall-out and accuracy.

Malware Detection Using Machine Learning

Saha's paper [9] dealt with the machine learning process to detect the malware in a computer system. The analysis is statically driven for antivirus scanning,hashing,file format inspection and signature based detection whereas for dynamic analysis malicious activities launched by applications are considered. The two machine learning approaches used are supervised and unsupervised. The feature extraction is used to extract the features from the set of PE files so that they can be analyzed for learning. Feature selection is used for filtering irrelevant features from the extracted ones. Some of the feature selection techniques are removing features with missing data, removing highly correlated features. The classification algorithms opted are K-Nearest, Naive Bayes, Decision Tree, Random Forest, Gradient Boosting, Logistic Regression. It concludes that in different feature selection techniques it results in a different number of features and for those features it gives different accuracy.

Machine Learning Methods For Malware Detection And Classification

The purpose of Chumachenko's paper [10] wass to determine the best feature extraction, feature representation, and classification methods that result in the best accuracy when used. The dataset used for this study consisted of the 1156 malware files of 9 families of different types and 984 benign files of various formats. In classification problems, different models gave different results. The feature selection used were filtered methods, wrapper methods, embedded methods. The lowest accuracy was achieved by Naive Bayes (72.34% and 55%), followed by K-Nearest-Neighbors and

Support Vector Machines (87%, 94.6% and 87.6%, 94.6% respectively). The highest accuracy was achieved with the J48 and Random Forest models, and it was equal to 93.3% and 95.69% for multi-class classification and 94.6% and 96.8% for binary classification respectively. The maliciousness of the file was a regression problem, and the severity score was its output.

Analysis of machine learning techniques used in behaviour-based malware detection

The classifiers used in Firdausi et.al. research [11] were k-Nearest Neighbors , Naïve Bayes, J48 Decision Tree, Support Vector Machine, and Multilayer Perceptron Neural Network. The overall best performance was achieved by J48 decision tree with a recall of 95.9%, a false positive rate of 2.4%, a precision of 97.3%, and an accuracy of 96.8%. In summary, it can be concluded that a proof of-concept based on automatic behaviour-based malware analysis and the use of machine learning techniques could detect malware quite effectively and efficiently. Both malware and benign instance data sets were in the format of Windows Portable Executable file binaries. A total of 220 unique malware samples were acquired. Feature selection was performed with Best First Search algorithm.

Feature selection and machine learning classification for malware detection

Khammasa et. al.'s work [12] presented a comparative study of several feature selection methods with four different machine learning classifiers in the context of static malware detection based on n-grams analysis. The result showed that the use of Principal Component Analysis (PCA) feature selection and Support Vector Machines (SVM) classification give the best classification accuracy using a minimum number of features. The feature selection methods used in this work were: CFs Subset, Principal Components,Info Gain Attribute,Correlation Attribute Eval,Gain Ratio Attribute, and Symmetrical Uncert Attribute.

The dataset that was used for analysis in this work contained 85 malware samples collected from different malware families. The benign files comprised 215 normal samples. These samples were collected from Windows executable files.

Dynamic malware detection using ML technique

This study [13] consisted of data generation phase, data extraction phase, classification phase, and performance metric computation phase. The results also showed that the SVM classifier was highly accurate.

The reporting results indicate that Instance based IBk and Tree Based J48 ML techniques have shown the results of TPR close to 100% and FPR close to 0%. So, these techniques can be considered as a candidate technique for building an effective malware detection system for novel and known malware samples dynamically.

Based on previous research it was decided that PE features would be the lightweight and efficient way to train our classifiers.

We decided to choose Portable Executable format for simple reasons:

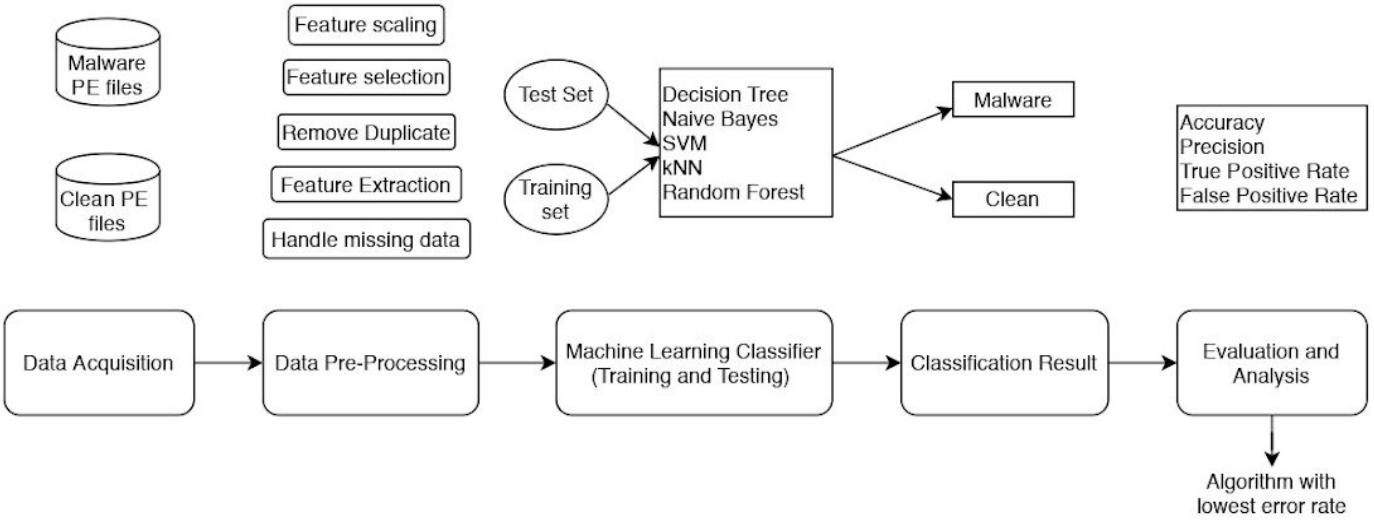
Windows is the most common operating system, therefore the vast majority of malicious files worldwide are targeted towards Windows users. At the same time collecting benign executables for Windows should be fairly easy, therefore the availability of data (files in PE format) is the next reason. Another reason being the usability of methods for detection based on PE header attributes. As mentioned before, Windows is the prevalent operating system, developing working solutions for protection against malicious PE files will affect most users. Third reason is the detailed documentation of this file format. Lastly, around 47.80% of files submitted to *Virustotal* are PE files.

Chapter 3

System Analysis and Design

All the studies presented in the previous chapter ended up with different results.The accuracy of each case depended on the specifics of malware families used and on the actual implementation. This chapter presents the approach used for the classification problem using machine learning algorithms and the system architecture with a detailed description of each stage.

3.1 System Architecture



3.1 System architecture

As machine learning is based on available data for the system to make a decision hence the first step in the architecture is **data acquisition**. This involves data collection, preparing and segregating the case scenarios based on certain features involved with the decision making cycle and forwarding the data to the processing unit for carrying out further categorization.

A real-world data consists of noise, missing values and sometimes is in unusable format that cannot be directly used for machine learning models. Hence,**Data pre-processing** is done to prepare raw data and make it suitable for these machine learning models.

Classification is a technique where we categorize data into a given number of classes. The main goal of a classification problem is to identify the category/class to which a new data will fall under.

Classification result is the outcome we get after using various classifiers. **Evaluation and Analysis** is done to check the performance of the algorithm and find the best algorithm that works for our model.

3.1.1 Data Acquisition

PE FILES

Windows program files are in a specific format called Portable Executable (PE). They were first introduced back in Windows NT 3.1. Since its introduction, extensions have been made for additional binary formats including .NET and 64 bit support (PE+32). This is the standard binary format for EXE, object code, DLLs, FON font files, and core dumps under Windows.[1] A PE file's data structure contains information necessary for the Windows OS loader to manage the wrapped executable code. Before the PE file there was a format called COFF used in Windows NT systems.

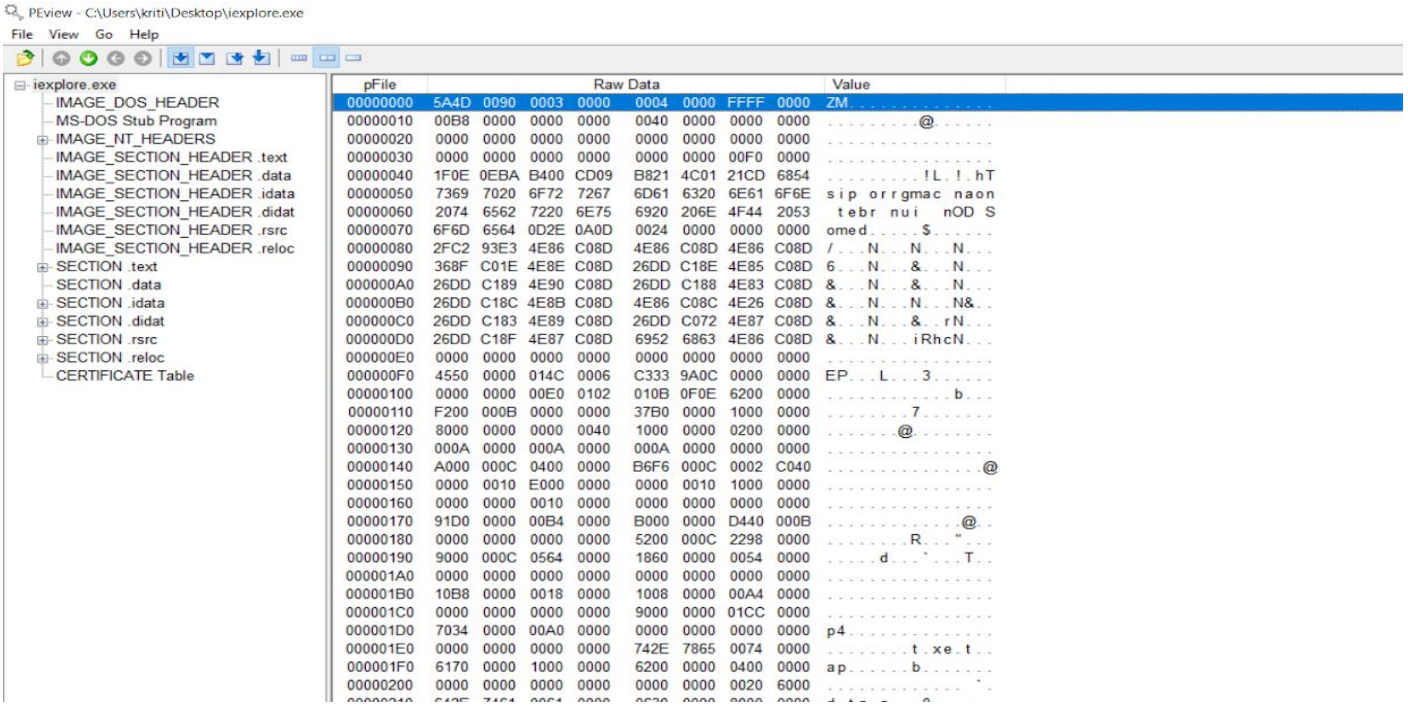


Fig 3.2. Hex view of the PE file of Internet Explorer

A PE basically contains two sections viz. Header and Sections which can be subdivided into several sections.

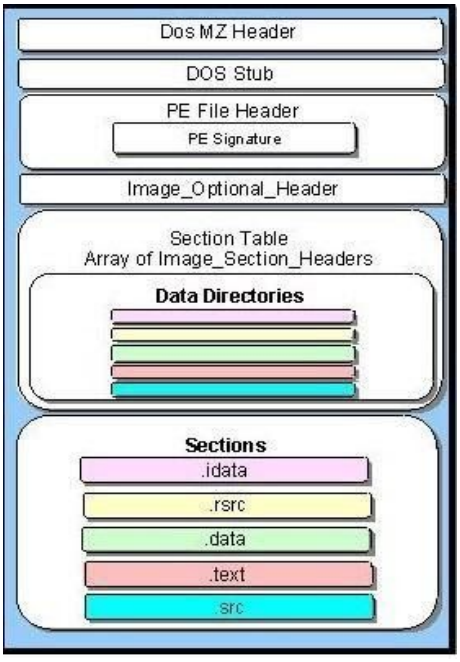


Fig 3.3 PE File Format

Features found in a PE file

DOS header:

Sl. No.	Name of the features	Description
1.	e_magic	Magic number ('MZ' in ASCII)
2.	e_cblp	Bytes on last page of file
3.	e_cp	Pages in file
4.	e_crlc:	Relocations
5.	e_cparhdr	Size of header in paragraphs
6.	e_minalloc:	Minimum extra paragraphs needed
7.	e_maxalloc:	Maximum extra paragraphs needed
8.	e_ss	Initial (relative) SS value
9.	e_sp	Initial SP value
10.	e_csum	Checksum
11.	e_ip	Initial IP value
12.	e_cs	Initial (relative) CS value
13.	e_lfarlc	File address of relocation table
14.	e_ovno	Overlay number
15.	e_res	Reserved words
16.	e_oemid	OEM identifier (for e_oeminfo)
17.	e_oeminfo	OEM information; e_oemid specific
18.	e_res2[10]	Reserved words
19.	e_lfanew	File address of the new exe header

Table 3.1 DOS Header

PE header:

Sl. no	Name of feature	Description
1.	Signature	The PE signature.
2.	Machine	The architecture type of the computer. An image file can only be run on the specified computer or a system that emulates the specified computer
3.	NumberOfSections	The number of sections. This indicates the size of the section table, which immediately follows the headers. Note that the Windows loader limits the number of sections to 96.
4.	TimeDateStamp	The low 32 bits of the time stamp of the image. This represents the date and time the image was created by the linker. The value is represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock.
5.	PointerToSymbolTable	The offset of the symbol table, in bytes, or zero if no PE symbol table exists.
6.	NumberOfSymbols	The number of symbols in the symbol table.
7.	SizeOfOptionalHeader	The size of the optional header, in bytes. This value should be 0 for object files.
8.	Characteristics	The characteristics of the image.

Table 3.2 PE Header

Some of the fields in the optional header:

Optional header:

Sl . no	Name of the feature	Description
1.	Magic	The state of the image file
2.	MajorLinkerVersion	The major version number of the linker.
3.	MinorLinkerVersion	The minor version number of the linker.
4.	SizeOfCode	The size of the code section, in bytes, or the sum of all such sections if there are multiple code sections.
5.	SizeOfInitializedData	The size of the initialized data section, in bytes, or the sum of all such sections if there are multiple initialized data sections.
6.	SizeOfUninitializedDat a	The size of the uninitialized data section, in bytes, or the sum of all such sections if there are multiple uninitialized data sections.
7.	AddressOfEntryPoint	A pointer to the entry point function, relative to the image base address. For executable files, this is the starting address. For device drivers, this is the address of the initialization function. The entry point function is optional for DLLs. When no entry point is present, this member is zero.
8.	BaseOfCode	A pointer to the beginning of the code section, relative to the image base.
9.	BaseOfData	A pointer to the beginning of the data section, relative to the image base.
10.	ImageBase	he preferred address of the first byte of the image when it is loaded in

Table 3.3 Optional Header

It is common for malware developers to manipulate the PE header to hide the malware instance from anti virus software.

3.1.2 Data Pre-processing

A real-world data consists of noises, missing data, inconsistent, and sometimes in an unusable format that cannot be directly used for machine learning models. Data pre-processing can be defined as a process of preparing raw data and making it suitable for these machine learning models. It is the first and crucial step while creating a machine learning model. It cleans the data and makes it suitable for machine learning increasing the accuracy and efficiency of a machine learning model.

While creating a project that involves machine learning we always don't come across the clean and formatted data. it is mandatory to clean and format it while doing any operation with data. Hence for this we use the task of data pre-processing.

Data pre-processing involves several steps:

1. Data Collection

As a machine learning model completely works on data, the first thing we need to create a machine learning model is a dataset. The data collected for a problem in a proper format is known as the dataset.

Each dataset is different from another dataset. To use the dataset in our code, we usually put the data into a CSV file. CSV stands for "Comma-Separated Values". It allows us to save tabular data.

2. Importing Libraries

We need to import some predefined Python libraries to perform data pre-processing using Python. They are used to perform some specific jobs. The three specific libraries for data pre-processing are Numpy, Matplotlib and Pandas. Numpy is used for mathematical operation in the code,

matplotlib for plotting charts and pandas for importing and managing datasets.

3. Importing the Dataset

To import the dataset we use read_csv() function of pandas library, which is used to read a csv file and perform various operations on it. Using this function, we can read a csv file locally.

4 . Handling Missing Data

A huge problem may be created for our machine learning model if our dataset contains some missing data. Hence it is necessary to handle missing values present in the dataset. The two ways to handle missing data are:

By deleting the particular row: The first way deals with null values. Here, we just delete the specific row or column which consists of null values. But this way is not so efficient as removing data may lead to loss of information which will not give the accurate output.

By calculating the mean: Here, we calculate the mean of that column or row which contains any missing value and will put it in the place of missing value. This strategy is useful for the features which have numeric data.

5. Encoding Categorical Data

Since the machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So, it is necessary to encode these categorical variables into numbers.

6. Splitting the dataset into training set and testing set

The dataset is divided into a training set and a test set. By doing this, we can enhance the performance of our machine learning model. Hence, it is considered as one of the crucial steps of data pre-processing.

Training Dataset:

The sample of data used to fit the model.

The actual dataset that we use to train the model (weights and biases in the case of Neural Network). The model *sees* and *learns* from this data.

Validation Dataset:

The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

The validation set is used to evaluate a given model, but this is for frequent evaluation. We as machine learning engineers use this data to fine-tune the model hyperparameters. Hence the model occasionally *sees* this data, but never does it "*Learn*" from this. We use the validation set results and update higher level hyperparameters. So the validation set in a way affects a model, but indirectly.

Test Dataset:

The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained(using the train and validation sets). The test set is generally what is used to evaluate competing models. Many times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled

data that spans the various classes that the model would face, when used in the real world.

About the dataset split ratio:

This mainly depends on 2 things. First, the total number of samples in your data and second, on the actual model you are training.

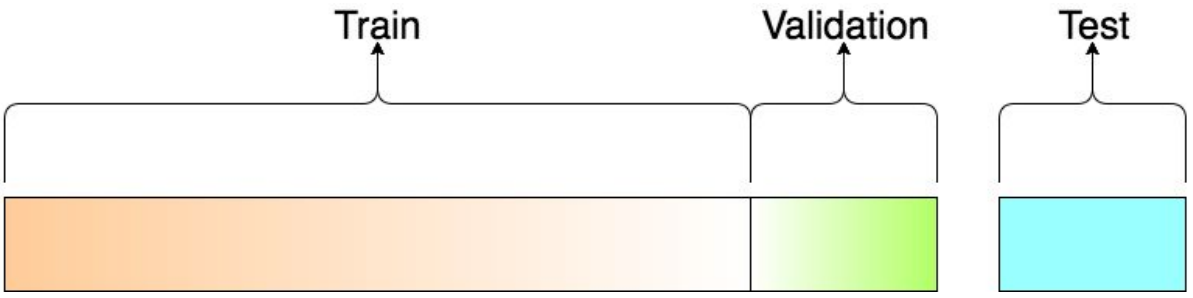


Fig 3.4 Visualisation of the dataset split

Some models need substantial data to train upon, so in this case you would optimize for the larger training sets. Models with very few hyperparameters will be easy to validate and tune, so you can probably reduce the size of your validation set, but if your model has many hyperparameters, you would want to have a large validation set as well(although you should also consider cross validation). Also, if you happen to have a model with no hyperparameters or ones that cannot be easily tuned, you probably don't need a validation set too!

7. FeatureScaling

The final step of data pre-processing in machine learning is feature scaling. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, variables are put in the same range and also on the same scale so that no variable dominates the other variable.

8. Feature Selection

Principal component analysis (PCA) is one of the techniques that reduces the dimensionality of large datasets that are difficult to interpret. This is achieved by increasing interpretability but at the same time minimizing information loss. New uncorrelated variables are created that maximize the variance. This reduces to solving an eigenvalue/eigenvector problem.A model with PCA can significantly reduce feature redundancy and learning time with minimum impact on data information loss.

[2]The whole process of obtaining principal components from a raw dataset can be simplified in six parts :

- Take the whole dataset consisting of *d+1 dimensions (features)* and ignore the target label such that the new dataset becomes *d dimensional.*
- Compute the *mean* for every dimension of the whole dataset.
- Compute the *covariance matrix* of the whole dataset.
- Compute *eigenvectors* and the corresponding *eigenvalues*.
- Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ *dimensional* matrix **W.**
- Use this $d \times k$ *eigenvector matrix* to transform the samples onto the new subspace.

3.1.3 Machine Learning Classifiers

3.1.3.1 Naive Bayes

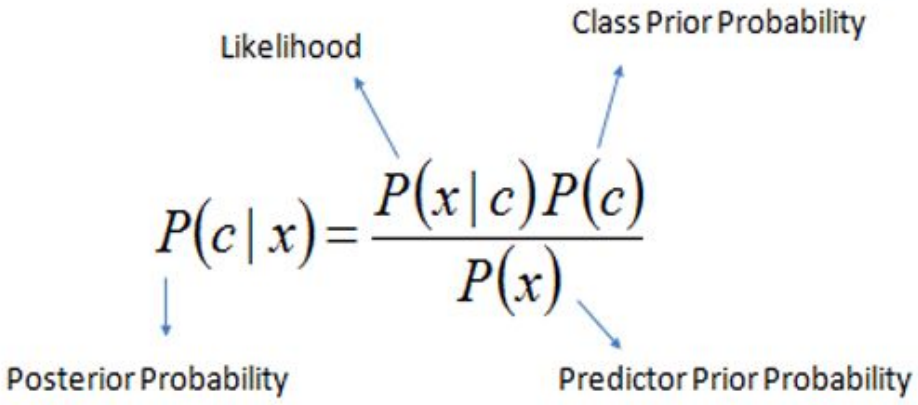
Naive Bayes is a supervised probabilistic machine learning classifier. The crux of the classifier is based on the Bayes theorem.

The fundamental Naive Bayes assumption is that each feature makes an

- independent
- equal

contribution to the outcome, ie. all the features in the dataset are independent of each other and have no correlation between them.

Bayes Theorem:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

The classification is conducted by deriving the maximum posterior probability for malware and benign files with the above assumption applying to Bayes theorem. This assumption greatly reduces the computational cost by only counting the class distribution. Even though the assumption is not valid in most cases since the attributes are dependent, surprisingly Naive Bayes has performed impressively.

Naive Bayes is a very simple algorithm to implement and can be easily scalable to larger datasets since it takes linear time, rather than by expensive iterative approximation as used for other classifiers. However, Naive Bayes can suffer from a problem called the zero probability problem. When the conditional probability P(c|X) is zero for a particular attribute, it fails to give a valid prediction. This needs to be fixed explicitly using a Laplacian

estimator where the count of the variable is increased from zero to a small value (usually 1) in the numerator, so that the overall probability doesn't become zero.[3]

3.1.3.2 k-Nearest Neighbours

The k-nearest neighbors (KNN) algorithm is a supervised machine learning algorithm that can be used as a classifier and a predictor. It assumes that similar things exist in close proximity and classifies a data point based on how its neighbours are classified. In our case an unknown file is classified into malware or benign based on how its k closest neighbours are classified.

Only major drawback is it becomes significantly slower as the size of the data in use grows. However, provided you have sufficient computing resources to speedily handle the data, KNN can be useful.[4]

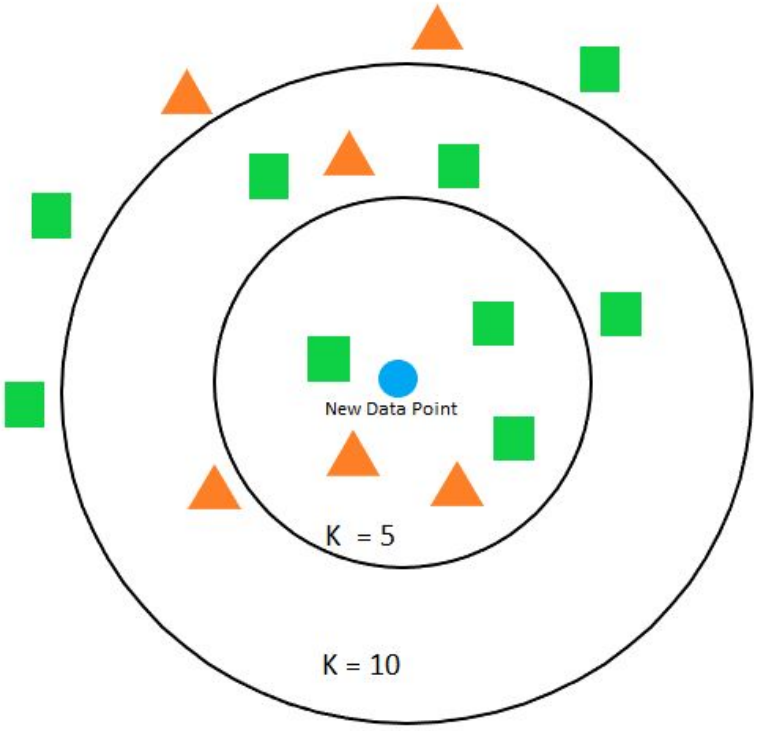


Fig 3.5 kNN Classification

[5]Pseudo Code for KNN

1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points

   i. Calculate the distance between test data and each row of training data. Euclidean distance is the most popular distance metric.

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

   ii. Sort the calculated distances in ascending order based on distance values

   iii. Get top k rows from the sorted array
4. Get the most frequent class of these rows
5. Return the predicted class

Choosing the right value for K

To select K, the algorithm is run several times with different values of K and that value is chosen which reduces the number of errors while maintaining the algorithm's ability to accurately make predictions when it is given data it hasn't seen before.

K is usually made an odd number to have a tiebreaker.

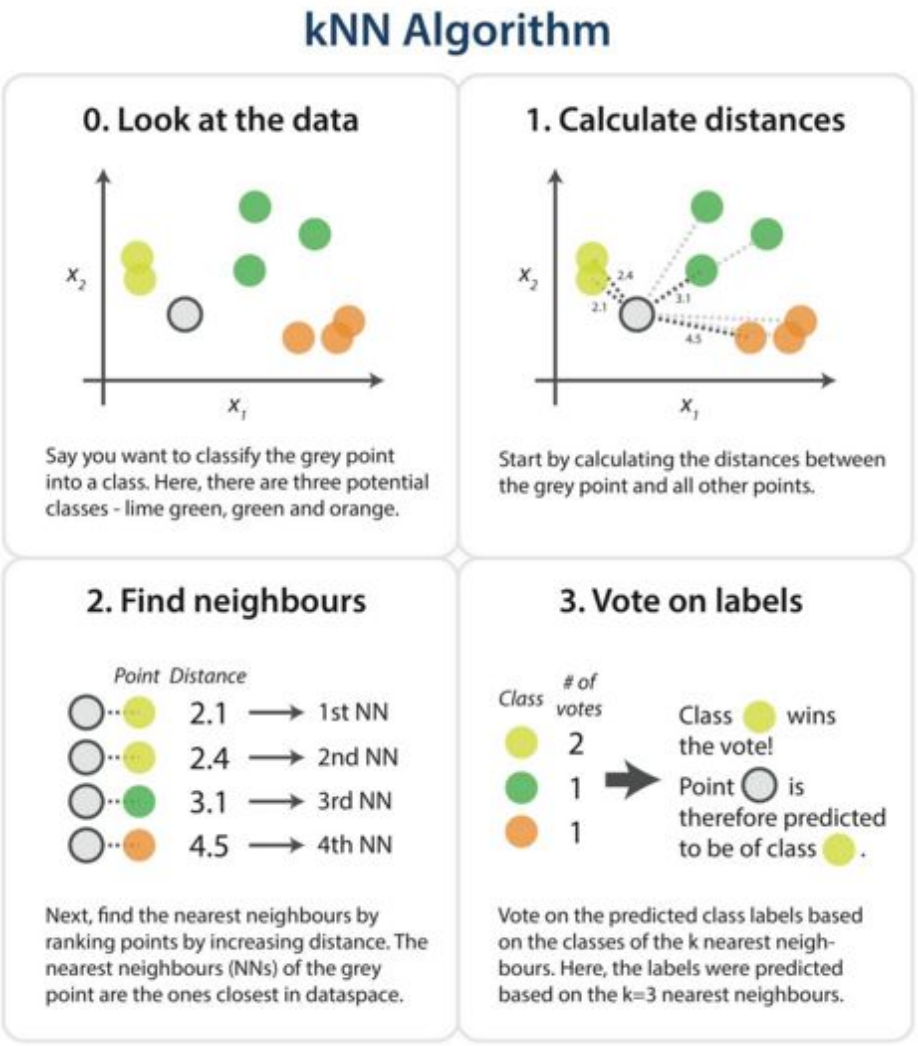


Fig 3.6 kNN Algorithm

3.1.3.3 Decision Tree

Decision tree is a supervised machine learning algorithm with a hierarchical tree structure constructed in a top-down recursive divide-and-conquer manner containing conditions in inner nodes and decisions in leaf nodes.

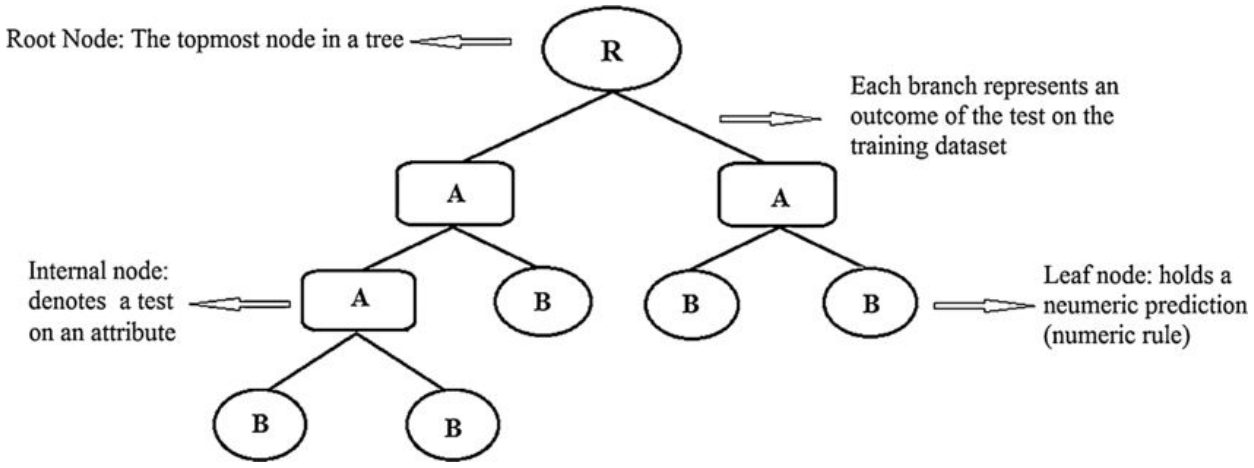


Fig 3.7 Decision Tree

When a sample is being classified by a tree, the condition for the node it reaches is evaluated and based on the result, the corresponding edge is followed. Whenever a leaf node is reached or when no more attributes for splitting are available, the classification process is over, and the sample is assigned a class belonging to the given leaf.[6]

The primary challenge in the decision tree implementation is to identify which attribute is to be considered the root node at each level. Handling this is known as the attribute selection. Attribute Selection Measure(ASM) is a heuristic for selecting the splitting criterion that partitions the data into the best possible manner. ASM provides a rank to each attribute. Attribute with the best score is selected as the splitting attribute.

Most common algorithm used for decision trees is ID3 (Iterative Dichotomiser 3). It uses the concept of Information gain and Entropy for attribute selection.

Entropy refers to the level of uncertainty in the data content.

Information gain is the decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values.

[7]**Pseudocode for ID3**

1. compute the entropy for data-set
2. for every attribute/feature:

a. calculate entropy for all categorical values

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

b. take average information entropy for the current attribute

$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

c. calculate gain for the current attribute

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

3. pick the highest gain attribute as the decision node.
4. Repeat until we get the tree we desired.

3.1.3.4 Random Forest

Random Forest is a supervised machine learning algorithm applicable to both regression and classification problems. It requires almost no data preparation or modeling and results in accurate results. Random Forests are based on decision trees. More specifically, Random Forest is a forest of

decision trees, producing a better prediction accuracy. The basic idea is to grow multiple decision trees based on the independent subsets of the dataset. At each node, `sqrt(n_features)` features out of the feature set are selected randomly, and the best split on them is found. If there are 16 features, at each node in each tree, only 4 random features will be considered for splitting.[8]

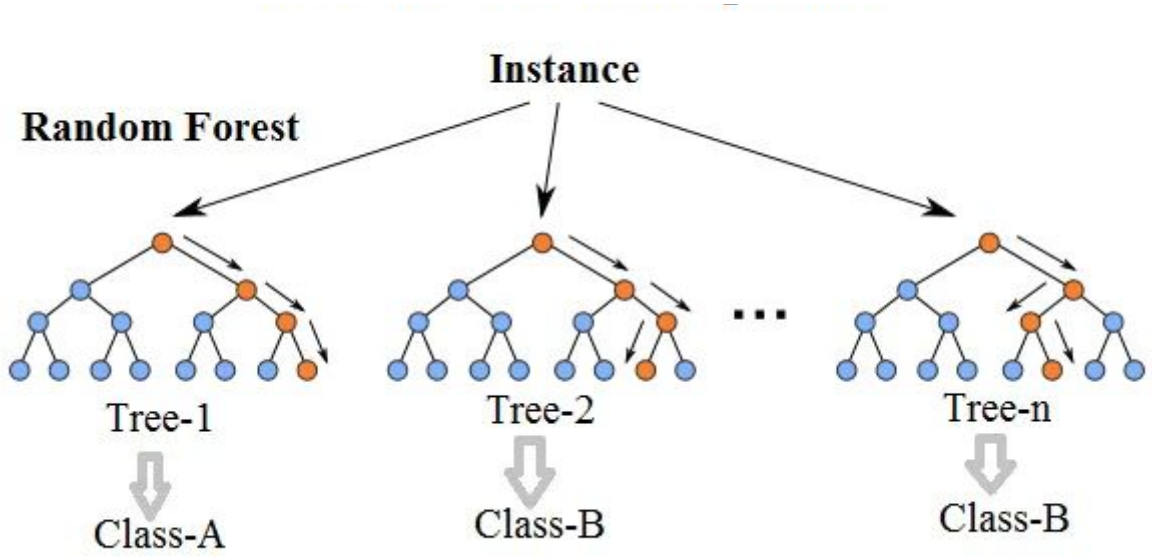


Fig 3.8 Simplified Random Forest

Pseudo Code for Random Forest

1. For each decision tree that is going to be grown:

a. Choose a random subset from the data set, this will be used to build each decision tree.

b. Grow each tree, apply the following steps on each node until the minimum depth is reached.

i.Randomly select a set of features.

ii. Pick the best split point based on the randomly selected features.

iii. Split the node into 2 child nodes.

2. Output the forest (all of the trees).

The final predictions of the random forest are made by averaging the predictions of each individual tree.

For example, consider a 5 tree Random Forest. Each tree would be created using one of the 5 subsets that was created in part (a) of the algorithm. Individual trees would be created by selecting a random fixed set of features at each node, then picking the best split point based on those features that will create two child nodes. This will be repeated until the minimum chosen depth is reached. Once the forest is created, the average probability on all 5 trees for each leaf classification can be computed to determine the classification

Random forests like decision trees are stable, easy to compute and quick to fit. They provide better accuracy than decision trees and remove the need for feature selection for removing irrelevant features. However, unlike decision trees, it is not very easy to interpret the results obtained from Random forests.

3.1.3.5 Support Vector Machines

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes.The main idea relies on finding such a hyperplane, that would separate the classes in the best way. It works well for linear and non linear problems.

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.[9]

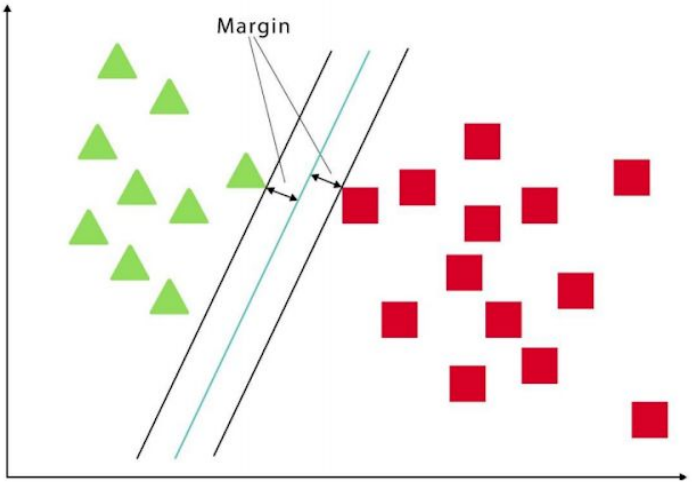


Fig 3.9 Support Vector Machine

Intuitively, we understand that the further from the hyperplane our classes lie, the more accurate predictions we can make. That is why, although multiple hyperplanes can be found per problem, the goal of the SVM algorithm is to find such a hyperplane that would result in the maximum margin.

In order to classify a non-linearly separable dataset like the one on the left given below, it is necessary to move away from a 2d view of the data to a 3d view. Features need to be added to have a hyper-plane, this is done through a technique called the kernel trick. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.[10]

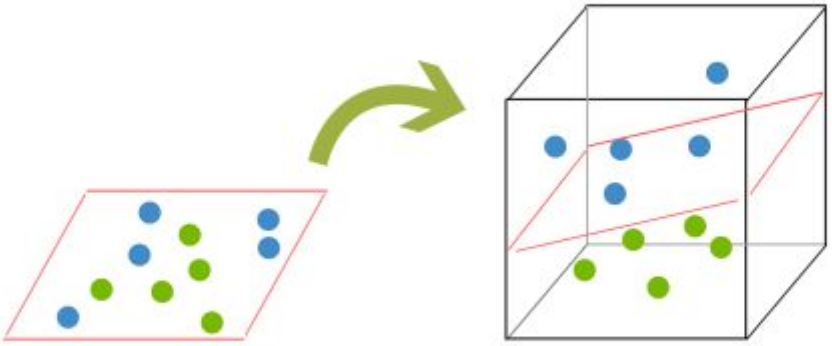


Fig 3.10 SVM Kernel

The idea is that the data will continue to be mapped into higher and higher dimensions until a hyperplane can be formed to segregate it.

SVM algorithm can be described as follows:

1. We define X and Y as the input and output sets respectively.

   $(x_1, y_1),\ldots,(x_m, y_m)$ 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.

$$R_{emp}(a) = \frac{1}{m}\sum_{i=1}^{m} l(f(x_i, a), y_i) = Training\ Error$$

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

$$R(a) = \int l(f(x,a),y)dP(x,y) = Test\ Error$$

5. We want to minimize the Training Error + Complexity term. So, we choose the set of hyperplanes,

f(x) = (w□x)+b:

$$\frac{1}{m}\sum_{i=1}^{m} l(w \cdot x_i + b, y_i) + ||w||^2 \text{ subject to } min_i|w \cdot x_i| = 1$$

SVMs are generally able to result in good accuracy, especially on ”clean” datasets. Moreover, it is good for working with high-dimensional datasets, even 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 doesn't prove to be very effective. With larger datasets training time can be high.

3.1.4. Classification Result

Using the above mentioned classifiers on our dataset we find the outcome to be either PE file (clean) or malware file. We then move forward towards the evaluation process to check the performance of each classifier and improve the efficiency of the model.

3.1.5 Evaluation and Analysis

3.1.5.1 Performance Metrics

Choosing the right metric is crucial while evaluating machine learning (ML) models. Various metrics are proposed to evaluate ML models in different applications. In some applications looking at a single metric may not give you the whole picture of the problem you are solving, and you may want to use a subset of the metrics to have a concrete evaluation of your models.

Confusion Matrix : Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

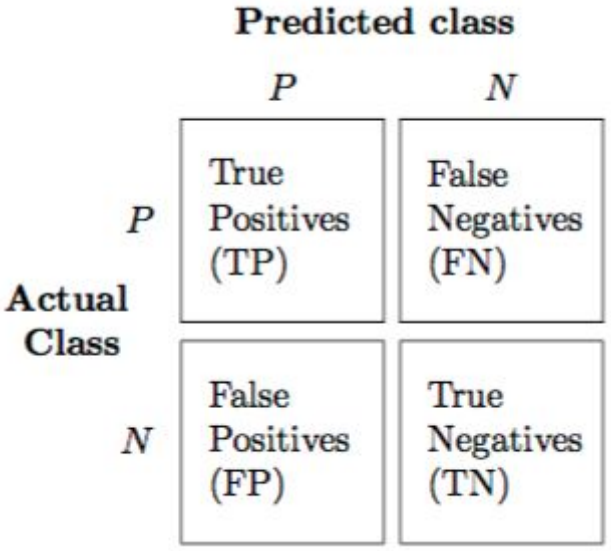


Fig 3.11 Confusion Matrix

- **True positives:** data points labeled as malware that are actually malware
- **False positives:** data points labeled as malware that are actually clean
- **True negatives:** data points labeled as clean that are actually clean
- **False negatives:** data points labeled as clean that are actually malware

Accuracy : It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Accuracy rate = (TP + TN)/(TP + FP +TN +FN)

There are many cases in which classification accuracy is not a good indicator of your model performance. One of these scenarios is when your class distribution is imbalanced (one class is more frequent than others). In this case, even if you predict all samples as the most frequent class you would get a high accuracy rate, which does not make sense at all (because your model is not learning anything, and is just predicting everything as the top class).
Therefore we need to look at class specific performance metrics too. Precision is one of such metrics.

Precision: It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Recall: It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives}$$

F1 Score: Depending on application, you may want to give higher priority to recall or precision. But there are many applications in which both recall and precision are important. Therefore, it is natural to think of a way to combine these two into a single metric. One popular metric which combines precision and recall is called F1-score, which is the harmonic mean of precision and recall defined as:

*F1-score= 2*Precision*Recall/(Precision+Recall)*

True Positive Rate (Sensitivity): True Positive Rate is defined as *TP/ (FN*+TP). True Positive Rate corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

$$TruePositiveRate = \frac{TruePositive}{FalseNegative + TruePositive}$$

False Positive Rate (Specificity) : False Positive Rate is defined as *FP / (*FP+TN). False Positive Rate corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

$$FalsePositiveRate = \frac{FalsePositive}{FalsePositive + TrueNegative}$$

AUC-ROC Curve : AUC - ROC curve is a performance measurement for classification problems at various threshold settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much a model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.[11]

ROC curve is plotted with TPR against the FPR

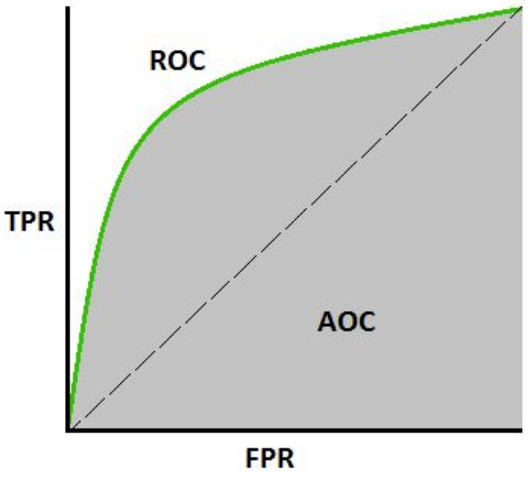
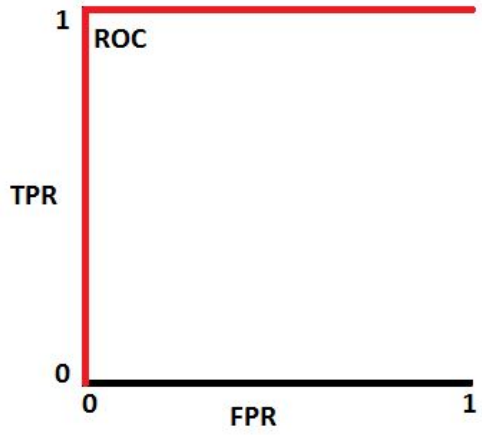


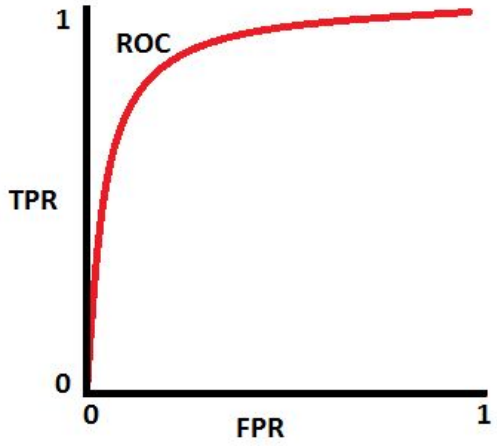
Fig 3.12 AUC-ROC curve

Different values of AUC



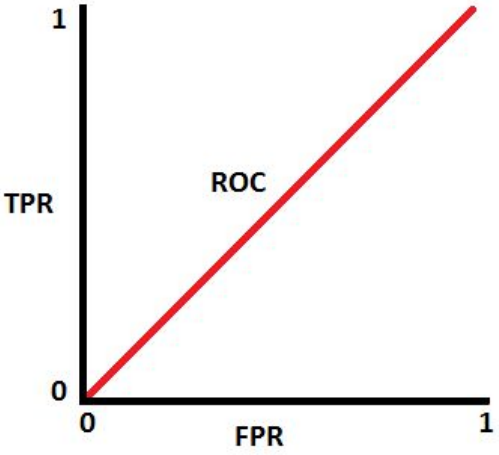
An excellent model has AUC near to the 1 which means it has good measure of separability.

Fig 3.13 ROC curve(1)



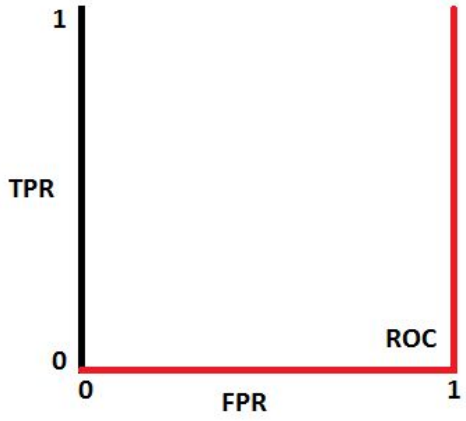
When AUC is 0.7, it means there is a 70% chance that the model will be able to distinguish between positive class and negative class.

Fig 3.14 ROC curve(2)



When AUC is approximately 0.5, model has no discrimination capacity to distinguish between positive class and negative class.

Fig 3.15 ROC curve(3)



A poor model has AUC near to the 0 which means it has worst measure of separability

Fig 3.16 ROC curve(4)

3.2 Technical Specifications

Technical requirements of the project.

Hardware Requirements	Software Requirements
1. Intel processor i3 or higher	1. Windows 7 onwards
2. Minimum 4GB RAM	2. Python 3.0 and above
3. Minimum 500GB HardDisk	

Chapter 4

Modelling and Implementation

In this chapter we implement the system based on the design presented in the previous chapter.

4.1 Use Case Diagram

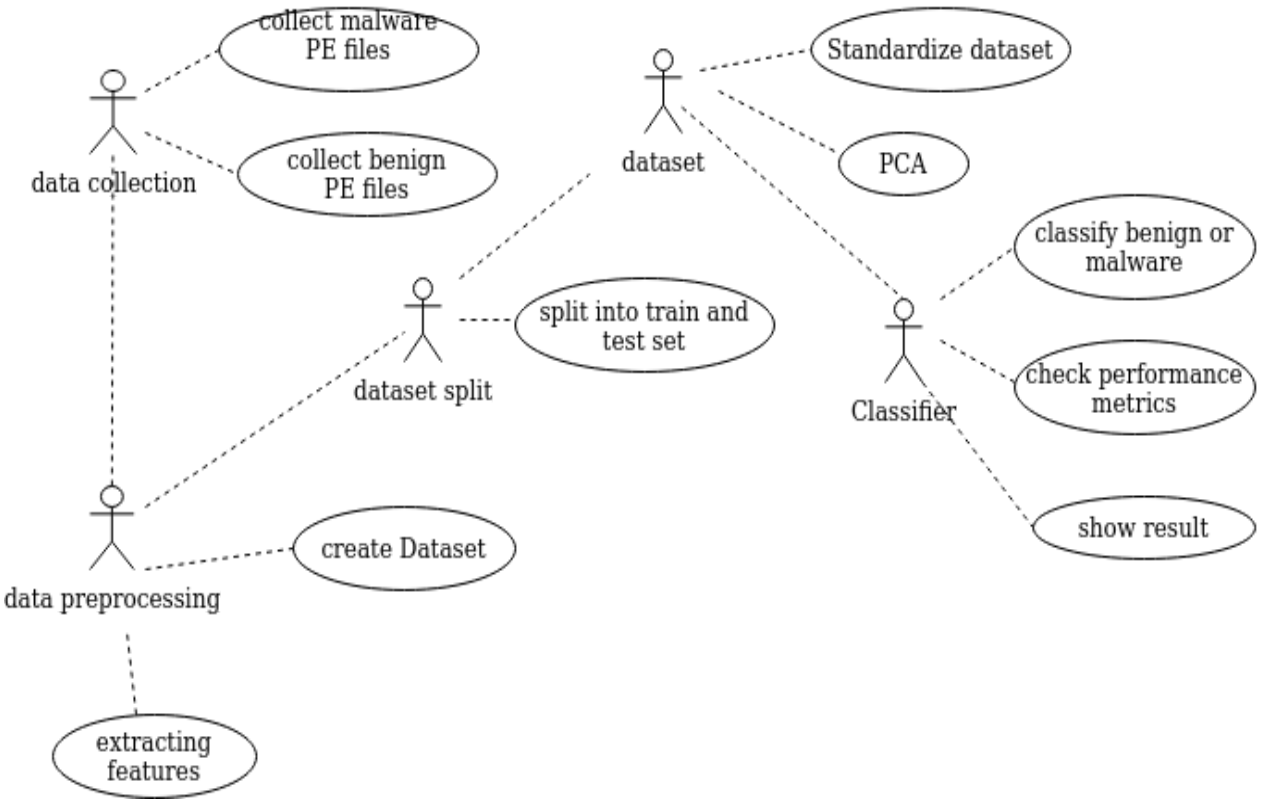


Fig. 4.1 Use Case Diagram

4.2 Class Diagram

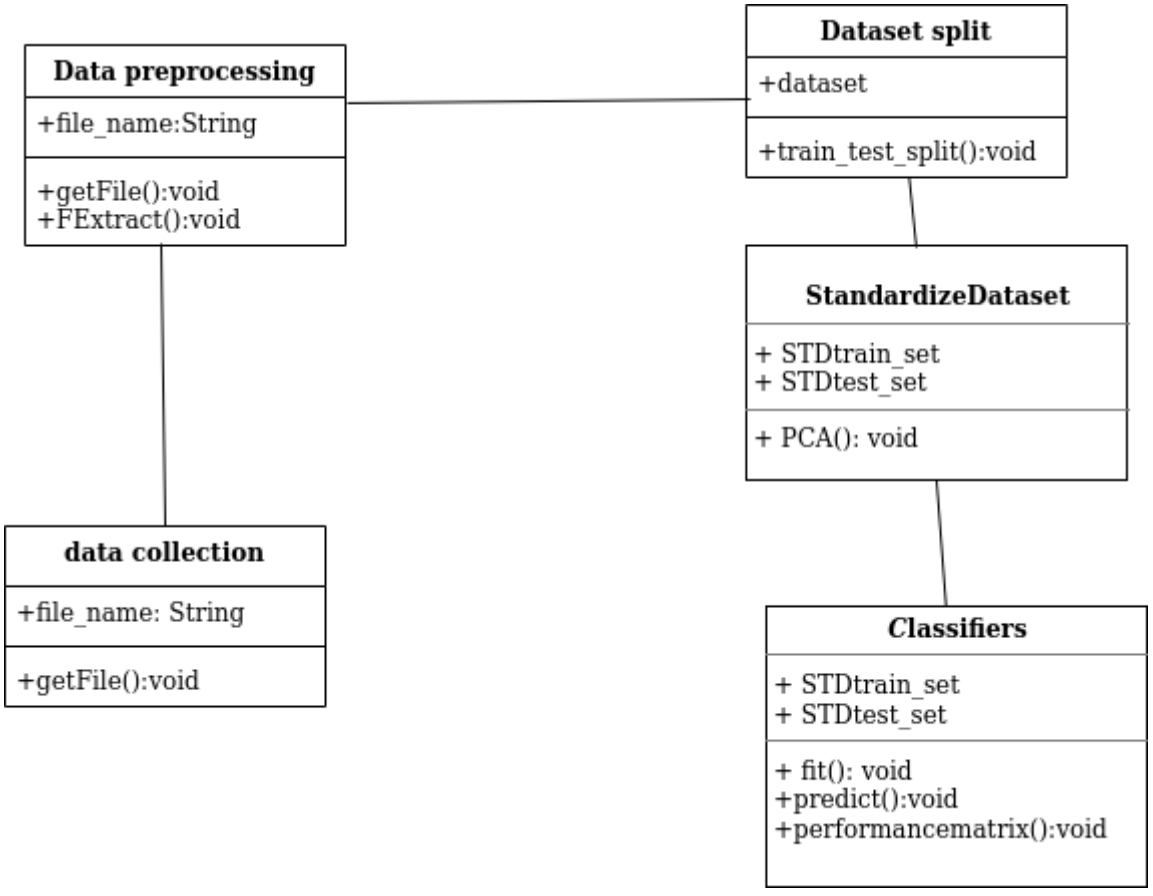


Fig. 4.2 Class Diagram

4.3 Sequence Diagram

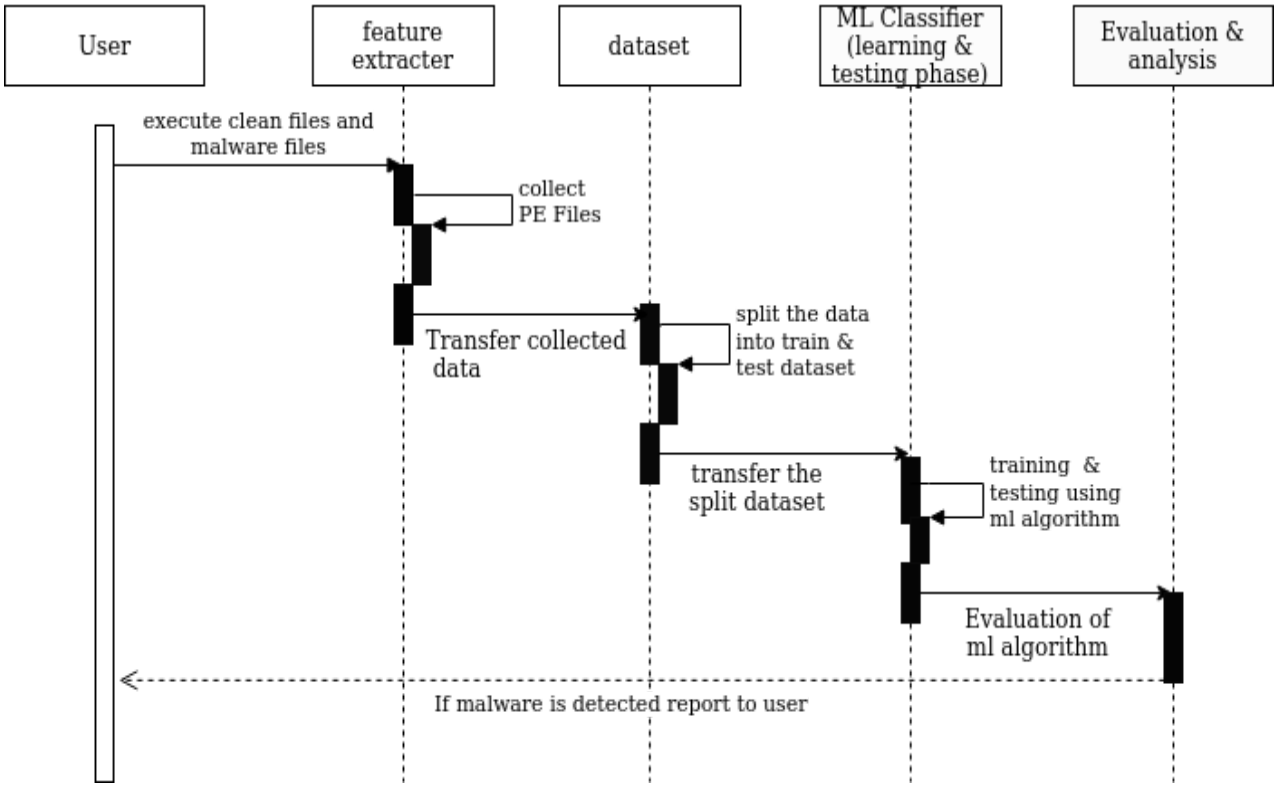


Fig. 4.3 Sequence Diagram

4.4 Tools Used

For this project Python was used as the programming language as it comes equipped with several modules for PE files parsing as well as machine learning modules. We profited from these modules and from the simplicity of this language. The tools and python modules we used are:

4.4.1 Jupyter notebook

Jupyter notebook is an open source web-based application that enables structuring and re-running of chunks of code in a user-friendly way. We used it for the data pre-processing and classifying phases.

4.4.2 pandas library

Pandas library is a python module widely used in the data science community. It provides functionality to encapsulate data manipulation, preparing data frames for the training phase that is provided by scikit-learn.

4.4.3 scikit-learn library

Scikit-learn is a python framework for machine learning. Encompassing feature selection methods,classifier algorithms, validation and performance evaluation functionality.

4.5 Dataset description

The dataset for this study was taken from Kaggle's 'benign & malicious PE files dataset for Malware Detection'. Total number of records in the dataset were 19612 out of which 14600 were malicious and the rest 5012 benign. It was built using Python's 'pefile' library that is used to extract attributes of a PE file. A total of 79 features along with class labels were extracted.As

mentioned earlier, along with performing a comparative study of the machine learning frameworks the objective is to reduce false negatives as much as possible. That is the reason why the number of malware samples in the dataset is more than benign samples as the models need to be trained on the malware records well.

Link to the dataset:

https://www.kaggle.com/amauricio/pe-files-malwares

4.6 Implementation

Major modules of the program are presented below.

1. The code below checks whether the dataset is balanced or not using Shannon's Entropy. A value close to 1 indicates a balanced dataset. An unbalanced dataset is one that consists of samples of one class more than the other. A classifier trained on an unbalanced dataset results in a low accuracy as it trains on one class better than the other. Shannon's entropy quantifies the amount of information in a variable using mathematical formulas.

The function below takes the target column (containing 0 or 1 for malicious and benign samples) as the argument and returns a value close to 1.

```
def balance(seq):
    n = len(seq)
    classes = [(clas,float(count)) for clas,count in Counter(seq).items()]
    k = len(classes)
    H = -sum([ (count/n) * log((count/n)) for clas,count in classes]) #shannon entropy
    return H/log(k)
```

Fig. 4.4 Balancing Dataset

2. Features that show no variation are dropped from the dataset along with 'Name' and 'Malware' attributes.

```
x = df.drop(['Name','Malware',  'e_magic', 'SectionMaxRawsize', 'SectionMaxVirtualsize', 'Sectic
```

Fig.4.5 Dropping unwanted columns

3. Splitting of dataset into train and test set. The split ratio is set to 70:30 i.s. 70% of the samples for training the classifier and 30% for testing.Training set consists of 3514 benign and 10213 malicious PE files. Test set contains 1498 benign and 4386 malicious executables.

```
xTrain, xTest, yTrain, yTest = train_test_split(x, df['Malware'], test_size = 0.3, random_state = 0)
```

Fig.4.6 Splitting dataset

4. Before performing dimensionality reduction using PCA the dataset is standardized. Standardization refers to shifting the distribution of each attribute to have a mean of zero and a standard deviation of one (unit variance). It is useful to standardize attributes for a model that relies on the distribution of attributes. n_components (number of components) for PCA() is chosen using trial and error.

The function below takes the dataset as the argument leaving out the target column and returns standardized dataset.

```
def standardizeDataset(xTrain, xTest):
    scaler = StandardScaler()
    xTrain = scaler.fit_transform(xTrain)
    xTest = scaler.transform(xTest)

    pca = PCA(n_components = 0.55)
    xTrainStd = pca.fit_transform(xTrain)
    xTestStd = pca.transform(xTest)
    return xTrainStd, xTestStd
```

Fig.4.7 Dataset Standardisation

The classifiers are fitted on the training set and tested using predict. Probability estimates are made on the classifiers at different thresholds to compute the AUC-ROC curve.

5. Naive Bayes classifier. Gaussian Naive Bayes has been used which means that the data from each label is drawn from a simple Gaussian distribution.

```
def NaiveBayes():
    classifier = GaussianNB()
    classifier.fit(xTrainStd, yTrain)
    predicted_labels = classifier.predict(xTestStd)
    probability = classifier.predict_proba(xTestStd)
    probability = probability[:, 1]
    expected_labels = yTest
    Accuracy(classifier, expected_labels, predicted_labels)
```

Fig.4.8 Naive Bayes Implementation

6. kNN Classifier. Votes of 5 nearest neighbours are taken to determine the result and each neighbour is weighted equally.

```
def KNN():
    classifier = KNeighborsClassifier()
    classifier.fit(xTrainStd, yTrain)
    predicted_labels = classifier.predict(xTestStd)
    probability = classifier.predict_proba(xTestStd)
    probability = probability[:, 1]
    expected_labels = yTest
    Accuracy(classifier, expected_labels, predicted_labels)
```

Fig. 4.9 kNN Implementation

7. Decision Tree Classifier. Gini index is used to make the split. The algorithm results in a fully grown tree.

```
def DecisionTree():
    classifier = DecisionTreeClassifier()
    classifier.fit(xTrainStd, yTrain)
    predicted_labels = classifier.predict(xTestStd)
    probability = classifier.predict_proba(xTestStd)
    probability = probability[:, 1]
    expected_labels = yTest
    Accuracy(classifier, expected_labels, predicted_labels)
```

Fig.4.10 Decision Tree Implementation

8. SVM Classifier. This kernel type in the algorithm is set to 'rbf' by default. The tolerance limit is set to 1e-3 which means the margin searches for data points that are as close as 10-3.

```
def SVM():
    classifier = SVC(probability=True)
    classifier.fit(xTrainStd, yTrain)
    predicted_labels = classifier.predict(xTestStd)
    probability = classifier.predict_proba(xTestStd)
    probability = probability[:, 1]
    expected_labels = yTest
    Accuracy(classifier, expected_labels, predicted_labels)
```

Fig.4.11 SVM Implementation

9. Random Forest Classifier. 100 decision trees are fitted on the dataset and each uses gini index to make the split.

```
def RandomForest():
    classifier = RandomForestClassifier()
    classifier.fit(xTrainStd, yTrain)
    predicted_labels = classifier.predict(xTestStd)
    probability = classifier.predict_proba(xTestStd)
    probability = probability[:, 1]
    expected_labels = yTest
    Accuracy(classifier, expected_labels, predicted_labels)
```

Fig.4.12 Random Forest Implementation

Chapter 5

Testing, Results and Discussion

All the experiments were conducted on a Windows 10 operating system. The workstation used has an 8 core, 2.1 GHz AMD Ryzen 5 3500U processor with 8 GB RAM and a 256 GB SSD. Jupyter Notebook's iPython Notebook project was used as the development environment for writing and executing the Python code.

5.1 Testing

5.1.1 Unit Testing

Each unit of the program code was tested and the outcome was consistent with our expectations.

Data acquisition: Testing of data acquisition involved checking if the dataset file was properly imported and the dataframe object had the right type of data in it.

```
df.head()
```

	Name	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	...	SectionMaxChar
0	VirusShare_a878ba26000edaac5c98eff4432723b3	23117	144	3	0	4	0	65535	0	184	...	3758096608
1	VirusShare_ef9130570fddc174b312b2047f5f4cf0	23117	144	3	0	4	0	65535	0	184	...	3791650880
2	VirusShare_ef84cdeba22be72a69b198213dada81a	23117	144	3	0	4	0	65535	0	184	...	3221225536
3	VirusShare_6bf3608e60ebc16cbcff6ed5467d469e	23117	144	3	0	4	0	65535	0	184	...	3224371328
4	VirusShare_2cc94d952b2efb13c7d6bbe0dd59d3fb	23117	144	3	0	4	0	65535	0	184	...	3227516992

5 rows × 79 columns

Fig.5.1 Dataset rows

Data preprocessing: In this module raw data was converted into clean data. Drop of unwanted columns ,removal of duplicate records, presence of empty cells and standardisation of the dataset was tested.The dataframe object was also tested for correct split of data.

```
df.shape #79 columns before dropping
(19611, 79)

x.shape  #70 columns remain after dropping
(19611, 70)
```

```
#number of empty cells in the dataset
x.isnull().sum().sum()
0
```

```
#14 columns remain after standardization and applying PCA
xTestStd.shape
(5884, 14)
xTestStd[:3,:]
array([[-2.20528542e-01,  2.89010070e-03,  3.99299730e+00,
         2.40574936e+00, -1.86696068e+00,  1.70448809e-01,
         3.99856582e-01,  9.76901345e-01, -1.53346915e+00,
         4.03681812e-01,  6.29893622e-01, -1.65017051e-01,
         1.02706365e+00,  3.66194021e-01],
       [-2.58314069e-01, -9.65629088e-02, -4.55812401e-01,
        -4.78330443e-01,  3.25042689e-01, -7.01442703e-02,
        -1.36368974e-01, -9.98348224e-02, -4.92676672e-01,
         3.50074681e-01, -1.93409960e-01, -1.53871276e+00,
        -8.73778856e-01,  7.35340257e-01],
       [-3.29847511e-02,  1.41925781e-02,  1.18622434e+00,
        -3.80345499e+00, -2.34444692e-01,  7.18603856e-02,
        -7.70603643e-01,  5.06457545e-01,  2.03682366e-01,
        -4.84256878e-01,  4.13638542e-01,  9.55371829e-02,
         1.70034427e+00, -9.77245200e-01]])
```

```
#checking for duplicate records
x.duplicated(keep='first')
0        False
1        False
2        False
3        False
4        False
         ...
19606    False
19607    False
19608    False
19609    False
19610    False
Length: 19611, dtype: bool
```

```
(len(xTrain)/len(x))*100    #train and test set ratio is 70:30
69.99643057467748

(len(xTest)/len(x))*100
30.003569425322524
```

Fig.5.2 Data Preprocessing

Machine learning Classifiers: Each classifier was trained with the training set and then tested. On testing following confusion matrices were obtained showing the classification of the samples.

```
NaiveBayes()
Confusion Matrix (Naive Bayes) [[tn, fp], [fn, tp]]
[[1466   32]
 [2919 1467]]
```

```
KNN()
Confusion Matrix (kNN) [[tn, fp], [fn, tp]]
[[1423   75]
 [  79 4307]]
```

```
DecisionTree()
Confusion Matrix (Decision Tree) [[tn, fp], [fn, tp]]
[[1421   77]
 [  89 4297]]
```

```
SVM()
Confusion Matrix (SVM) [[tn, fp], [fn, tp]]
[[1364  134]
 [  84 4302]]
```

```
RandomForest()
Confusion Matrix (Random Forest) [[tn, fp], [fn, tp]]
[[1428   70]
 [  43 4343]]
```

Fig.5.3 Confusion matrices of the classifiers

Evaluation and Analysis: Confusion reports given below show the various performance metrics obtained on running the classifiers

```
NaiveBayes()
Classification Report
              precision    recall  f1-score   support

           0       0.33      0.98      0.50      1498
           1       0.98      0.33      0.50      4386

    accuracy                           0.50      5884
   macro avg       0.66      0.66      0.50      5884
weighted avg       0.81      0.50      0.50      5884
```

```
KNN()
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      1498
           1       0.98      0.98      0.98      4386

    accuracy                           0.97      5884
   macro avg       0.97      0.97      0.97      5884
weighted avg       0.97      0.97      0.97      5884
```

DecisionTree()

Classification Report

	precision	recall	f1-score	support
0	0.93	0.95	0.94	1498
1	0.98	0.98	0.98	4386
accuracy			0.97	5884
macro avg	0.96	0.96	0.96	5884
weighted avg	0.97	0.97	0.97	5884

SVM()

Classification Report

	precision	recall	f1-score	support
0	0.94	0.91	0.93	1498
1	0.97	0.98	0.98	4386
accuracy			0.96	5884
macro avg	0.96	0.95	0.95	5884
weighted avg	0.96	0.96	0.96	5884

RandomForest()

Classification Report

	precision	recall	f1-score	support
0	0.97	0.95	0.96	1498
1	0.98	0.99	0.99	4386
accuracy			0.98	5884
macro avg	0.98	0.97	0.97	5884
weighted avg	0.98	0.98	0.98	5884

Fig.5.4 Performance metrics obtained from the classifiers

5.1.2 Integration and System Testing

All the units were integrated into a single system and run to evaluate how the system performs. The system performed the tasks as designed and gave the same result each time it was run. The figure below shows that Random Forest performed the best out of all the classifiers.

```
accuracies
```

```
{'Naive Bayes': 0.4984704282800816,
 'kNN': 0.9738273283480625,
 'SVM': 0.9629503738953094,
 'Decision Tree': 0.9687287559483345,
 'Random Forest': 0.9814751869476547}
```

```
max_key = max(accuracies, key=accuracies.get)
```

```
max_key
```

```
'Random Forest'
```

Fig.5.5 Classifier with maximum accuracy

5.2 Results and Discussion

All the malware records constitute the positive class (or class 1) while benign records constitute negative class (or class 0).

5.2.1 Classification results of Naive Bayes

First algorithm to be tested was Naive Bayes. It gave an accuracy of 49.8%.

Area under the curve achieved for the algorithm was 0.91.

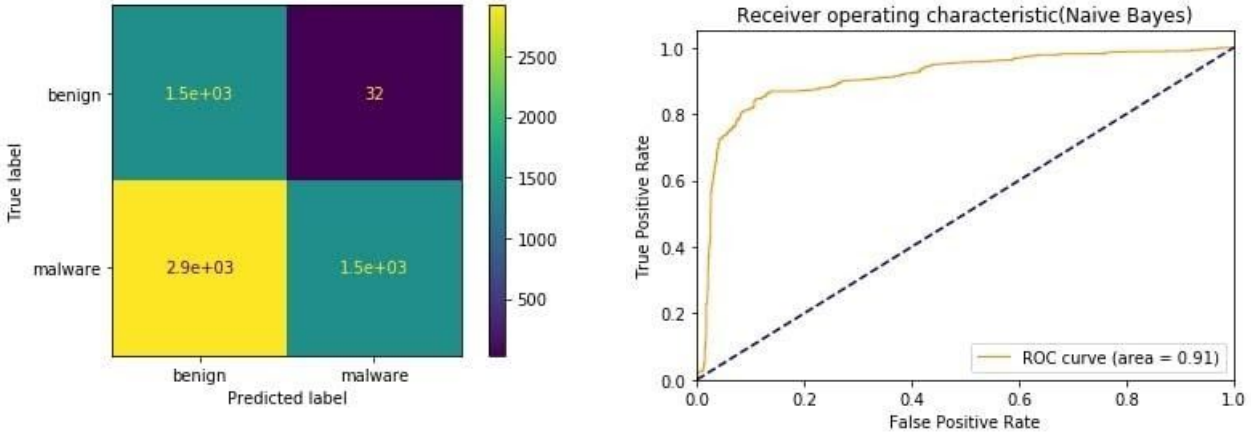


Fig 5.6 Naive Bayes

As can be seen in fig. 5.3.1 out of 1498 benign files 1466 were detected correctly and out of 4386 malware files 1467 were detected correctly. Naive Bayes gave a false negative of 2919 ie. 2919 malicious files were detected incorrectly. It can be dangerous for the system if a malicious file is mistaken for a benign file by the antivirus. While, a benign file being mistaken for a malicious file would not cause much problem as it can be further analysed.

Overall, the performance of Naive Bayes was not good.

5.2.2 Classification results of kNN

Second algorithm tested was kNN. It gave an accuracy of 97%.

Area of the curve obtained was 0.98.

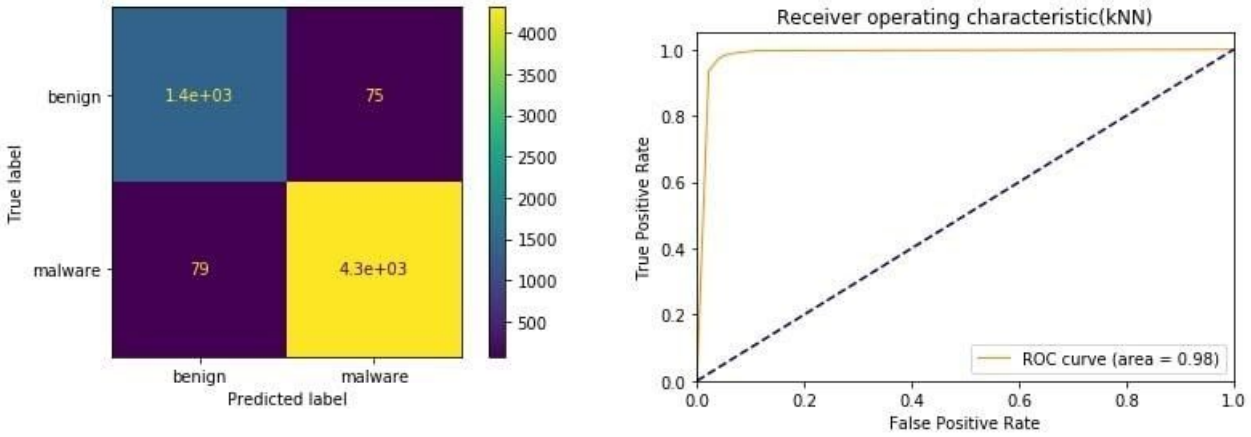


Fig 5.7 kNN algorithm

As can be seen in fig. 5.3.2 out of 1498 benign files 1423 were detected correctly and out of 4386 malware files 4307 were detected correctly. kNN gave a false negative of 79 ie.79 malicious files were detected incorrectly.

This classifier shows significant improvements over Naive Bayes. We let k take the default value of 5 neighbours as the dataset was standardized and brought down to the same scale prior to training.

5.2.3 Classification results of Decision Tree

Third algorithm to be tested was Decision Trees. It gave an accuracy of 94%. Area of the curve received was 0.96.

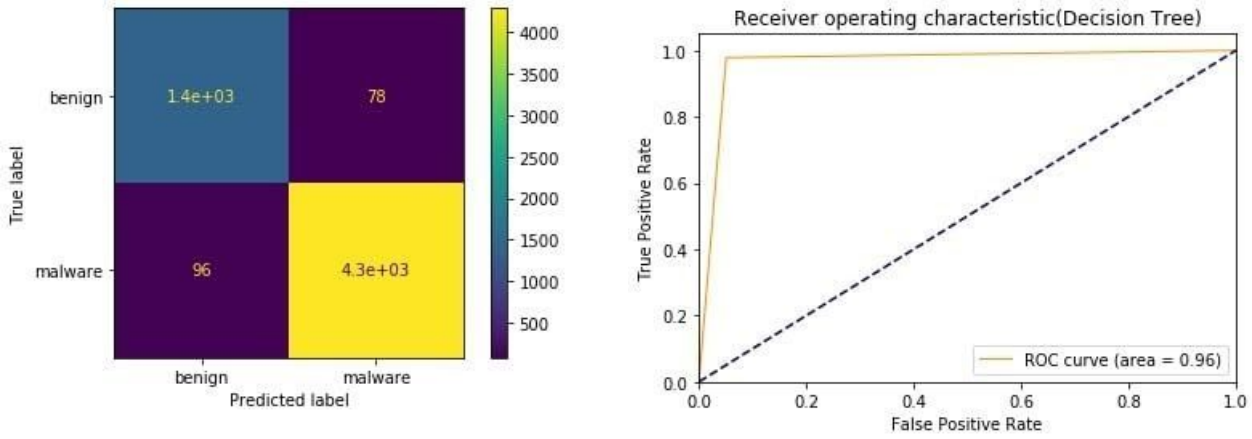


Fig 5.8 Decision Tree

As can be seen in fig5.3.3 out of 1498 benign files 1420 were detected correctly and out of 4386 malware files 4290 were detected correctly. Decision Tree gave a false positive of 96 ie. 96 malicious files were detected incorrectly. In terms of accuracy the model performed well, but the TPR is not as high as kNN.

5.2.4 Classification results of SVM

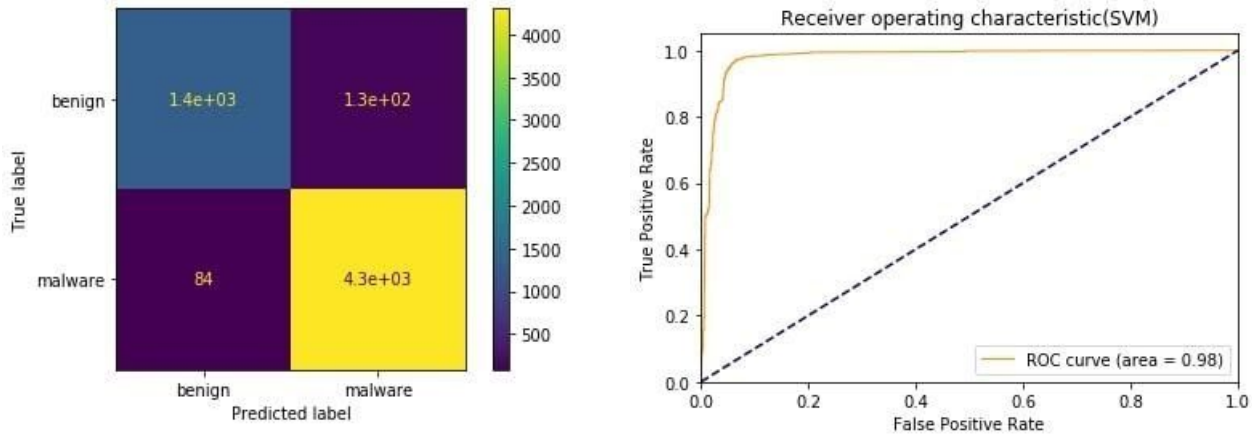


Fig.5.9 SVM algorithm

Fourth algorithm to be tested was SVM. It gave an accuracy of 96%.

Area of the curve received was 0.98.

As can be seen in fig5.3.4 out of 1498 benign files 1364 were detected correctly and out of 4386 malware files 4302 were detected correctly. SVM gave a false negative of 84 ie. 84 malicious files were detected incorrectly. SVM made some improvements over Decision Tree in terms of the true positive .

5.2.5 Classification results of Random Forest

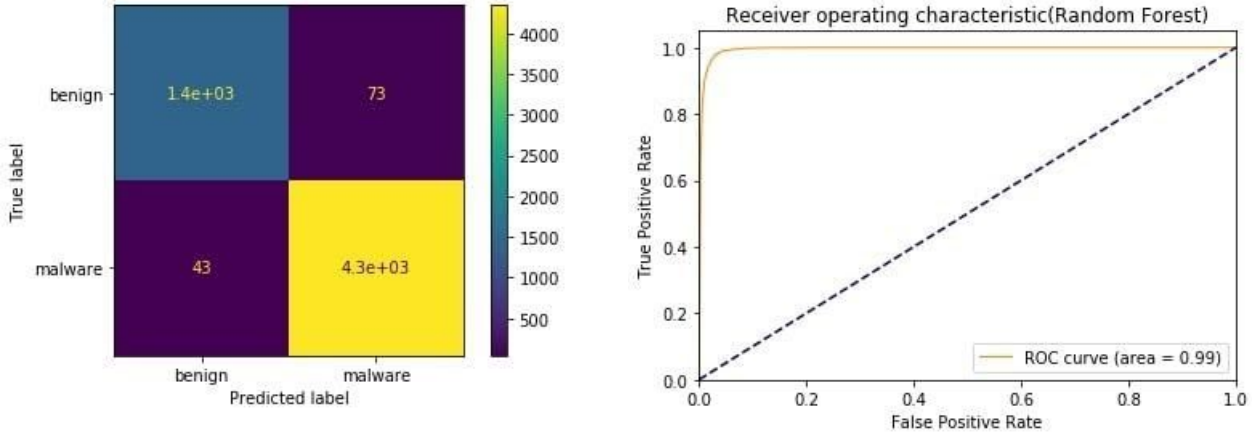


Fig.5.10 Random forest

Lastly Random Forest was tested. The best results were obtained from this classifier with an accuracy of 98%.Area under the curve obtained for the algorithm was 0.99. As can be seen in fig5.3.5 out of 1498 benign files 1425 were detected correctly and out of 4386 malware files 4343 were detected correctly. Random Forest gave a false negative of 43 ie. 43 malicious files were detected incorrectly.

Random Forest gave the highest accuracy with the least false negative value. Thus, Random Forest is the best out of all the classifiers used here for the malware detection technique used in this project.

Table 5.2.1 summarises the classification results obtained from the models.

Classifier	Classification Results					
	TP	FN	TN	FP	Accuracy	AUC
Naive Bayes	1467	2919	1466	32	49.8%	0.91
kNN	4307	79	1423	75	97.3%	0.98
Decision Tree	4290	96	1420	78	97.4%	0.96
SVM	4302	84	1364	134	96.2%	0.98
Random Forest	4343	43	1425	73	98%	0.99

Table 5.2.1 Classification Results

Fig. 5.11 Bar Graph of classification results

From the above bar graph it can be seen that Random Forest gave the best accuracy and Naive Bayes ,as expected, resulted in a very low accuracy with most of the samples misclassified. Recall and F1 score were the least for Naive Bayes with values 0.33 and 0.50 respectively. While other classifiers resulted in a value of 0.98. All the classifiers gave a precision score of 0.98.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The main target of this study was to test different machine learning frameworks that can detect as many malware samples as possible with a tough constraint of having a zero false negative rate. The results obtained were very close to the goal, although non-zero false negative rate could not be avoided completely.

Any commercial anti-virus product is subject to certain speed and memory limitations, hence the machine learning classifiers , in our opinion, malware detection via machine learning will not replace the standard detection methods used by anti-virus vendors, but will come as an addition to them.

6.2 Future Work

Future work of this study will be based on dynamic analysis in a safer environment to prevent the malware from deleting files or stealing data. A larger data set could be used because larger the data set, better the training. Datasets with massive numbers of records are good for statistical analysis. Real malware datasets and design ensemble techniques will be utilized. The study will also be extended to other machine learning algorithms.

Bibliography

[1] Sharma Divya Mukesh, Jigar A. Raval, Hardik Upadhyay, "A Survey on Malware Detection Schemes Using Machine Learning Techniques" International Journal of Research and Scientific Innovation(2013): 258-267.

[2] Raman, Karthik. “Selecting Features to Classify Malware.” Springer International Publishing(2012).

[3] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. 2001. Data Mining Methods for Detection of New Malicious Executables. In Proceedings of the 2001 IEEE Symposium on Security and Privacy (SP ’01). IEEE Computer Society, USA, 38.

[4] Edward Raff, Jared Sylvester, and Charles Nicholas. 2017. Learning the PE Header, Malware Detection with Minimal Domain Knowledge. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec ’17). Association for Computing Machinery, New York, NY, USA, 121–132. DOI:https://doi.org/10.1145/3128572.3140442

[5] Singla, Sanjam et al. “A Novel Approach to Malware Detection using Static Classification.” International Journal of Computer Science and Information Security (2015).

[6] Machine Learning Methods for Malware Detection, Kaspersky, 2019

[7] Gavriluț, Dragoş & Cimpoesu, Mihai & Anton, D. & Ciortuz, Liviu. (2009). Malware detection using machine learning. 4. 735 - 741. 10.1109/IMCSIT.2009.5352759.

[8] Dhiman, Abhishek. “MALWARE DETECTION AND CLASSIFICATION USING MACHINE LEARNING TECHNIQUES Submitted in fulfillment of seminar required for the Master of Technology Computer Science and Engineering.” Springer International Publishing (2018).

[9] Saha,Jayisha, “Malware Detection Using Machine Learning”, 2016

[10] Kateryna Chumachenko, “Machine Learning Methods For Malware Detection And Classification ”Kaakkois-Suomen ammattikorkeakoulu, 2017.

[11] I. Firdausi, C. lim, A. Erwin and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection," 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, Jakarta, 2010, pp. 201-203, doi: 10.1109/ACT.2010.33.

[12] Mohammed, Ban & Monemi, Alireza & Joseph, Stephen & Ismail, Ismahani & Nor, Sulaiman & Marsono, Muhammad Nadzir. (2015). Feature selection and machine learning classification for malware detection. Jurnal Teknologi. 77. 10.11113/jt.v77.3558.

[13] Dynamic malware detection using ML technique, 2015

[14] PE Header Analysis for Malware Detection 2018, Samuel Kim Pg-4

[15]https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

[16], [21], [22] Machine Learning Methods for Malware Detection and Classification 2017, Kateryna Chumachenko, Pg-21, 26, 20

[17]https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[18]https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

[19]Static detection of malicious PE files 2018/19, Jakub Ács Pg 23-24

[20]https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1

[23]https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[24]https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5