



REPUBLIC OF TÜRKİYE
ALTINBAŞ UNIVERSITY
Institute of Graduate Studies
Electrical And Computer Engineering

**AN OPTIMIZED MALWARE DETECTION
TECHNIQUE USING SUBSPACE
LEARNING-BASED ONE-CLASS
CLASSIFICATION**

Hasan Harith Jameel ALKHALI

Ph.D. Dissertation

Supervisor

Asst. Prof. Dr. Muhammad ILYAS

İstanbul, 2023

AN OPTIMIZED MALWARE DETECTION TECHNIQUE USING SUBSPACE LEARNING-BASED ONE-CLASS CLASSIFICATION

Hasan Harith Jameel ALKHALI

Electrical and Computer Engineering

Doctor of Philosophy

ALTINBAŞ UNIVERSITY

2023

The dissertation titled “AN OPTIMIZED MALWARE DETECTION TECHNIQUE USING SUBSPACE LEARNING-BASED ONE-CLASS CLASSIFICATION” prepared by HASAN HARITH JAMEEL ALKHSHALI and submitted on 27/12/2023 has been **accepted unanimously** for the degree of Ph.D. in Computer Engineering.

Asst. Prof. Dr. Muhammad ILYAS

Supervisor

Dissertation Defense Committee Members:

Asst. Prof. Dr. Muhammad ILYAS	Department of Computer Engineering, Altınbaş University	_____
Asst. Prof. Dr. Hakan KOYUNCU	Department of Computer Engineering, Altınbaş University	_____
Asst. Prof. Dr. Abdullahi Abdu IBRAHIM	Department of Computer Engineering, Altınbaş University	_____
Assoc. Prof. Dr. Jawad RASHEED	Department of Computer Engineering, İstanbul Sabahattin Zaim University	_____
Asst. Prof. Dr. Çağatay AYDIN	Department of Electrical-Electronics Engineering, Ege University	_____

I hereby declare that this dissertation meets all format and submission requirements of a Ph.D. dissertation.

I hereby declare that all information/data presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

Hasan Harith Jameel ALKHSHALI

Signature

DEDICATION

First, Thanks to Allah, who completed his blessing upon me and brought me to where I am. Allah is the owner of the credit and blessing in everything in my life, so, praise and thanks go to Allah first and last.

To my first teacher after ALLAH, whose wish was to see me at this stage, and my wish was for him to be with me to make him happy and raise his head in these moments, Dr. Harith “MY FATHER”, may ALLAH have mercy on him. It was the will of Allah Almighty, and there is no objection to it, for him to choose MY FATHER next to him while I was in the middle of the search. This man was the best teacher for me. He is the best advisor, how much I needed him to be with me during the last days of my research, to follow his advices, and to complete what he started with me in guiding me to the path of success and excellence. May ALLAH have mercy on you, MY FATHER, and make you dwell in His spacious Paradise.

To MY MOTHER, Maha, the most compassionate and the kindest heart, after ALLAH’s grace for me, I did not reach where I am except thanks to her prayers and concern for me. I hope that I can return her merit one day.

To my wife LINA and my three daughters, YUSUR, JUDE and MIN’NA, thank you for supporting me and for putting up with the difficult days I faced due to my study, I prey to ALLAH that one day I can compensate you for the hardships you have endured.

To My Advisor Assistant Professor MUHAMMAD ILYAS, thank you Doctor for accepting to supervise my dissertation and supporting me till reaching this level, thank you Doctor. Also, I want to thank Dr. FAHAD SOHRAB for his guidance and help in the final stages of my dissertation. Finally, my thanks to ALTINBAŞ UNIVERSITY which gave me this opportunity to study.

ABSTRACT

AN OPTIMIZED MALWARE DETECTION TECHNIQUE USING SUBSPACE LEARNING-BASED ONE-CLASS CLASSIFICATION

AL-KHSHALI, Hasan Harith Jameel

Ph.D., Electrical and Computer Engineering, Altınbaş University,

Supervisor: Asst. Prof. Dr. Muhammad ILYAS

Date: December / 2023

Pages: 155

Advancement in technology have imposed on security specialists to constantly find new methods and solution to system security problems. Accordingly, Cybersecurity was at the forefront of the solutions that must be researched. The problem here is that the danger exists continuously and has different forms that can be altered, this can camouflage the risk and thus being able to deceive the protection program.

Since including the whole set of security problems and trying to find a solution that solves them collectively will be an impossible mission, since the security problems have a wide spectrum and covering them in one shot is impossible. So, taking each problem independently and covering it will be a better solution.

Malware is considered as one of the oldest security threats that had widely affected systems, malware variations are numerous, covering the threat of malware will mean covering a good percentage of the security problems, since many of the problems depends on malware to implant its malicious applications inside the system.

Three different strategies have been implemented; the goal was to detect malware. The start was with traditional Artificial Intelligence, aiming to test the influence of the header features that exists in the portable executable file on the accuracy of malware detection, two of the three strategies were implemented regarding this issue. The attained results have outperformed other similar implementations.

The third strategy that has been implemented was a different novel strategy that has been implemented for the first time. The main aim of this strategy was to overcome all the minor wholes that may lead to obtain a faked good result and this was an important issue. Dataset imbalance and curse of dimensionality were an obstacle that may affect the attained results. This new strategy has bypassed these obstacles and outperformed any previously implemented strategy. The strategy was the subspace learning. The techniques implemented from this strategy were Subspace Support Data Description (SSVDD) and Graph Embedded SSVDD, before implementing them, One-Class Classification and Support Vector Data Description were also tested, results attained from subspace learning have outperformed other results from other techniques implemented previously.

Keywords: Artificial Intelligence (AI), Cybersecurity, Decision Tree (DT), Neural Network Multi-Layer Perceptron Classifier (NN_MLPC), One-class Classification (OCC), Subspace Support Vector Data Description (SSVDD), Graph Embedded Subspace Support Vector Data Description (GESSVDD).

TABLE OF CONTENTS

	<u>Pages</u>
ABSTRACT	vi
LIST OF TABLES.....	xii
LIST OF FIGURES.....	xiii
ABBREVIATIONS.....	xv
LIST OF SYMBOLS	xiv
1. INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 DISSERTATION OBJECTIVE AND PURPOSES	4
1.3 WORK PHASES AND IMPLEMENTATION.....	5
1.3.1 Work Stages	5
1.3.2 Purpose of Each Stage	6
1.3.3 Datasets Used.....	7
1.3.4 Datasets Used.....	8
1.4 CONTENT.....	8
2. TERMINOLOGIES AND TOOLS.....	10
2.1 INTRODUCTION	10
2.2 MALWARE AND PORTABLE EXECUTABLE FILE.....	10
2.2.1 Malware	10
2.2.2 Portable Executable	12
2.2.3 Malware in Portable Executable File	13
2.3 NON-PORTABLE EXECUTABLE FILE	17
2.4 MALWARE DETECTION TECHNIQUES	18
2.5 ALGORITHMS AND TOOLS USED	20
2.5.1 NN_MLPC.....	20
2.5.2 Decision Tree.....	21
2.5.3 OCC	22

2.5.4 Support Vector One-Class Classification	23
2.5.5 OCC Based on Subspace Learning	24
2.5.5.1 Subspace SVDD	24
2.5.5.2 Graph-embedded SSVDD	25
2.5.5.3 Programming languages	25
2.6 HARDWARE	25
3. EFFECT OF HEADER FEATURES ON DETECTION ACCURACY	27
3.1 CHAPTER INTRODUCTION	27
3.2 PART ONE	28
3.2.1 Introduction	29
3.2.2 Criteria and Basis	31
3.2.2.1 Algorithms used	31
3.2.2.2 Feature selection	32
3.2.2.3 Feature numbering	34
3.2.2.4 Standards of Test_Size and Random_State	35
3.2.2.5 Implemented algorithms	36
3.2.3 Procedure	37
3.2.3.1 Algorithms testing	37
3.2.3.2 Implementation Process	39
3.2.4 Performance outcomes	39
3.2.5 Detailed Results Obtained	41
3.2.6 Analysis And Comparison of Results	47
3.2.6.1 NN_MLPC	48
3.2.6.2 DT	49
3.2.7 Observations	51
3.2.7 Part one's Conclusions	55
3.3 PART TWO	55

3.3.1 Part two Introduction.....	56
3.3.2 Research Literature Review	60
3.3.3 Header Features of Portable Executable.....	65
3.3.4 Algorithms.....	67
3.3.5 Work Constants	68
3.3.6 Methodology - Subgrouping.....	72
3.3.6.1 Runs for NN_MLPC_0.3_10.....	75
3.3.6.2 Runs for NN_MLPC_0.15_3.....	76
3.3.6.3 Runs for DT_0.3_10	77
3.3.6.4 Runs for DT_0.15_3	77
3.3.7 Results.....	78
3.3.7.1 NN_MLPC_0.3_10	78
3.3.7.2 NN_MLPC_0.15_3	79
3.3.7.3 DT_0.3_10.....	81
3.3.7.4 DT_0.15_3.....	82
3.3.8 Observations Discussion.....	86
3.3.8.1 Observations of the work.....	86
3.3.8.2 A Comparison study	90
3.3.9 Summary and Conclusions	95
4. ONE-CLASS CLASSIFICATION AND SUBSPACE LEARNING-BASED ONE-CLASS CLASSIFICATION IN MALWARE DETECTION.....	97
4.1 CHAPTER INTRODUCTION.....	97
4.2 INTRODUCTION	99
4.3 MALWARE.....	102
4.4 SV BASED OCC FOR MALWARE DETECTION	103
4.4.1 One-Class SVM	103
4.4.2 Support Vector Data Description.....	104
4.5 SUBSPACE LEARNING.....	105

4.5.1 SSVDD	105
4.5.2 GESSVDD	106
4.6 WORK IMPLEMENTATION	108
4.6.1 Preprocessing and the Datasets	108
4.6.2 Used Hyperparameters	109
4.6.3 Metrics: Evaluating Classifier Performance	110
4.7 EXPERIMENTAL RESULTS DISCUSSION	112
4.8 SUMMARY AND CONCLUSIONS OF THE WORK	121
5. SUMMARY AND CONCLUSIONS	122
5.1 SUMMARY	122
5.2 CONCLUSIONS	127
REFERENCES	128

LIST OF TABLES

	<u>Pages</u>
Table 1.1: Datasets Used	8
Table 1.2: Published Papers	8
Table 3.1: Excluded Features	34
Table 3.2: Features Used Sorted According To Their Importance.....	35
Table 3.3: Result Synopsis	40
Table 3.4: All The Runs And Accuracies Obtained Starting With Ten Features.....	43
Table 3.5: The Added Runs And Their Accuracies.....	47
Table 3.6: Summary Of Analysis: Maximum And Minimum Accuracy Attained	51
Table 3.7: Features And Their Numbers	67
Table 3.8: Low Accuracy Due to Use Only Features Twenty to Twenty-Nine	84
Table 3.9: Incorporating Half or Majority of Features From The Features Twenty To Twenty-Nine.....	84
Table 3.10: Maximum Accuracy Obtained	89
Table 3.11: Previous Work And Current Work.....	91
Table 3.12: Samples of The Runs Done	92
Table 4.1: Datasets Used in This Work.....	109
Table 4.2: Linear Results for Dataset A, Dataset_Malwares	115
Table 4.3: Non-Linear Results for Dataset A, Dataset_Malwares	116
Table 4.4: Linear Results for Dataset B, Clamp.....	117
Table 4.5: Non-Linear Results for Dataset B, Clamp	118
Table 4.6: Linear Results for Dataset C, Pe_Section_Headers	119
Table 4.7: Non-Linear Results for Dataset C, Pe_Section_Headers	120

LIST OF FIGURES

	<u>Pages</u>
Figure 1.1: General Structure for Portable Executable Header File.....	3
Figure 1.2: Malware Detection Techniques.	3
Figure 2.1: The Format Overview of a Portable Executable File.....	13
Figure 2.2: Original Software and Software with Code Cave Comprise the Software Structure.	15
Figure 2.3: Software Execution: a) Without Code Cave, b) With Code Cave, c) Complete Change to Execution Flow.	16
Figure 2.4: Malware Detection Strategies.	18
Figure 2.5: Malware Analysis Types.....	19
Figure 2.6: An MLP's Hidden Layer.	21
Figure 3.1: NN_MLPC.....	37
Figure 3.2: DT.	37
Figure 3.3: NN_0.3_10 Results.	49
Figure 3.4: NN_0.15_3 Results.	49
Figure 3.5: DT_0.3_10 Results.	50
Figure 3.6: DT_0.15_3 Results.	50
Figure 3.7: Algorithms General Flowchart.	72

Figure 3.8: Bar Chart for NN_MLPC_0.3_10.....	79
Figure 3.9: Bar Chart for NN_MLPC_0.15_3.....	80
Figure 3.10: Bar Chart for DT_0.3_10.....	82
Figure 3.11: Bar Chart for DT_0.15_3.....	83
Figure 3.12: The NN_MLPC_0.15_3 Combo Chart.	85
Figure 3.13: Combo Chart for the DT_0.15_3.	86
Figure 4.1: The Proposed Pipeline.	101

ABBREVIATIONS

Adware	:	Advertising-Supported Software
AI	:	Artificial Intelligence
AIS	:	Artificial Immune System
.bat	:	Batch File
BiLSTM	:	Bidirectional Long Short-Term Memory
BSA	:	Business Software Alliance
COFF	:	Common Object File Format
CPU	:	Central Processing Unit
.CSV	:	Comma-Separated Values
DDR RAM	:	Double Data Rate Synchronous Dynamic Random-Access Memory
DLAM	:	Document Level Analysis Model
DLL	:	Dynamic-Link Library
DNS	:	Domain Name Service
.docx	:	Microsoft Word Open XML Format Document File
DOS	:	Disk Operating System
DPI	:	Deep-Pack Inspection
DSL MGPT2	:	Domain Specific Language Model Gpt-2
DT	:	Decision Tree
.EXE	:	Executable
FN	:	False Negative

FP	:	False Positive
GB	:	Gigabyte
GE-SVDD	:	Graph Embedded Subspace Svdd
GHz	:	Gigahertz
GLM-GPT2	:	General Language Model Generative Pre-Trained 2
GE-OC-SVM	:	Graph Embedded One Class Support Vector Machine
GE-SVDD	:	Graph Embedded Support Vector Data Description
Gmean	:	Geometric Mean
GPT	:	Generative Pre-Trained
.HTA	:	HTML Applications
HBCIA	:	Host-Based Code Injection Attack
HTML	:	Hyper Text Markup Language
HTTP	:	Hypertext Transfer Protocol
IEEE	:	Institute of Electrical and Electronics Engineers
IoT	:	Internet of Things
ISSN	:	International Standard Serial Number
JAVA	:	Just Another Virtual Architecture
Keylogger		Keystroke Logger
KNN	:	K-Nearest Neighbour
LibSVM	:	Library for Support Vector Machine
LightGBM	:	Light Gradient-Boosting Machine

.LNK	:	LiNK
Malware	:	Malicious Software
MATLAB	:	Matrix Laboratory
ML	:	Machine Learning
MLP	:	Multi-Layer Perceptron
MS-DOS	:	Microsoft Disk Operating System
.msi	:	Microsoft Software Installer
MZ	:	Mark Zbikowski. [Hexadecimal: (4d 5a) ₁₆]
NN_MLPC	:	Neural Networks–Multi-Layer Perceptron
NPT	:	Non-Linear Projection Trick
NVIDIA	:	N A Mathematical Variable VIDIA Stands For Video
OCC	:	One-Class Classification
OCSVM	:	One Class Support Vector Machine
OS	:	Operating System
PCA	:	Principle Component Analysis
PE	:	Portable Executable
PEFAF	:	Portable Executable File Analysis Framework
Q	:	Quartile
RAM	:	Random Access Memory
RF	:	Radio Frequency
Rootkit	:	Compound Of "Root" Which Is The Traditional Name Of The Privileged Account On Unix-Like Operating Systems. And The

	:	Word "Kit" Which Refers To The Software Components That Implement The Tool.
SCIE	:	Science Citation Index Expanded
.SCP	:	Script Component Files
SLAMs	:	Sentence Level Analysis Models
SoM	:	Self-Organizing Map
Spyware	:	Spying Software
SVDD	:	Support Vector Data Description
SVM	:	Support Vector Machine
Ssdeep	:	Context Triggered Piecewise Hashing
SSVDD	:	Subspace Support Vector Data Description
.sys file	:	System File
TN	:	True Negative
TNR	:	True Negative Rate Or Specificity
TP	:	True Positive
TPR	:	Sensitivity, True Positive (Recognition) Rate (TPR) Or Recall
UNIX	:	Uniplexed Information Computing System
VRAM	:	Video Random Access Memory
WiFi	:	Wireless Fidelity
.ZIP	:	Zone Improvement Plan

LIST OF SYMBOLS

C	:	Hyperparameter Controls the Trade-Off Between Margin Maximization and Training Error Minimization
Φ	:	The Non-Linear Relationships in The Data Can Be Handled Using Φ
σ	:	The Distance Between Two Datapoints Can Be Scaled Using the Hyperparameter Σ , This Was in The Non-Linear Implementations
d	:	The D Hyperparameter Was Used to Represent the Dimensionality of The Data Or The Number Of Data Used In The Lower-Dimensional Space
η	:	Hyperparameter Represents the Learning Rate
β	:	Used To Control Regularization Term Importance
P	:	All The Positive Samples
N	:	All The Negative Samples
ψ	:	Regularization Term
ξ_i	:	A Slack Variable Which Allows for Some Training Errors
$\mathbb{R}^D$	:	D -Dimensional Feature Space
a	:	Denotes the Hypersphere Center
$\mathcal{L}$	:	Lagrange Multipliers
L_x	:	Graph Laplacian Is Represented
γ_i	:	Lagrange Multiplier
α_i	:	Lagrange Multiplier

1. INTRODUCTION

1.1 INTRODUCTION

The proliferation of Internet usage during the last twenty years and how most domains of applications have employed this technology to obtain more benefits and to make it easy to accomplish many paperwork without the need to go physically to the intended place, add to that the various daily life applications, social communication, emails and so on have become an important need to approximately all the people, not only the specialists or people working in computer fields. Although this proliferation included many advantages and made the work accomplishment easier, but on the other hand there were some challenges appeared, the main challenge was the cybersecurity. According to [1], United States, Japan, China and Germany who represents the biggest economies at that time had loss \$200 billion because of cybercrime.

Cyberization is a trend which constitute the new cyber world. The rapid spread of this terminology especially with the smart devices propagation made the organizations, private and governmental take into consideration how to protect their assets from cyber-attacks and enhance their cybersecurity [2]. Cybersecurity includes information securing, but recent advancement made cybersecurity include confidence and trust in digital systems, however, fundamental values of users like privacy, freedom, fairness, etc. are still respected and protected [3].

Cybersecurity includes an assortment of technologies, Internet of Things (IoT) may be the most popular and evolving technology nowadays. IoT adds more prosperity to life, accordingly, the demand on IoT is growing continuously, this results in many security challenges and a continuous optimization to security strategies will be required. IoT system represented by its layers which are Perception Layer, Network Layer, Support Layer and Application Layer, like any other networking model each layer can be exposed to many attacks, each attack must be taken into consideration in the cybersecurity strategy. For example, the perception layer may be exposed to spoofing attacks, eavesdropping attack, Tag Cloning attack, or RF Jamming attack. Network layer include attacks like Sinkhole, Sybil, Malicious code injection, Man-in-the-Middle or sleep deprivation. Support and Application layers include similar attacks like DNS flood, HTTP flood, Ethernet

IEEE802.11/ Wi-Fi attacks, Botnet, Identity theft or Malware attacks [4] – [6]. According to that, cybersecurity strategy must cover a wide range of security threats so that it can be successful.

Malicious Software or Malware can serve in many types of attacks and be the tool used by the attack to enter the system and achieves its intended goal, hence, malware detection can be used to protect the system from multiple attacks. Many researches including researches on IoT mentioned that there is an increase in malware variants which converts the protection process of networks and IoT devices to a challenging process [7], [8]. Add to that the same malware family may include malware variants with similar behaviour or signatures representing the malware purpose [8]. Malware has the capability to nestle inside the system and disable its features if it discovers that there is an attempt to detect its existence [7]; this may be a dangerous feature that the malware characterized with, since this feature may lead to that the security software be unsuccessful in detecting the malicious software.

The file format of a Portable Executable (PE) in Windows operating system include a list of information that is used to execute files. Portable Executable header part behaves like a metadata for different tables since the header contain all the necessary details for the stored data in the PE file [9]. Microsoft Windows use the PE file format so that files can be executed. The header part in the PE file format includes all needed information that describes the included data in the PE file. Hence, the header acts as descriptive data for the tables inside. DLLs, executables (.exe), object files and other types of files has the format of portable executable. Figure 1-1 gives a general idea about the structure of the portable executable file format, it is divided into parts like the data dictionaries, PE header, PE signature, DOS stub, sections, and image optional header. Since the PE header hosts all the information which depicts the body of the PE file, so, if a malware sticks to a file of the PE kind, it can easily harm the system and may corrupts the whole activity of the system.

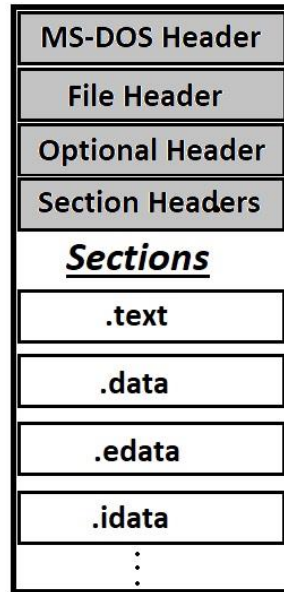


Figure 1.1: General Structure for Portable Executable Header File.

Since malware first appear, strategies and tactics have been evolved to cope with the malware dynamic and evolving nature, it is known that the malware structure and tactic are always modified, which means that the tool that can detect a certain malware structure cannot detect another variation of the malware. According to that, many strategies have been evolved. Two main malware detection strategies can be recognized: Behaviour or Anomaly-Based Detection and Signature-Based Detection [10], [11]. Both strategies can take the shape of static, dynamic or hybrid analysis. It is important to indicate the methodology of detecting the malware, whether the methodology is anomaly or signature based, Figure 1-2 illustrate the ecosystem for detecting malware [10], [11].

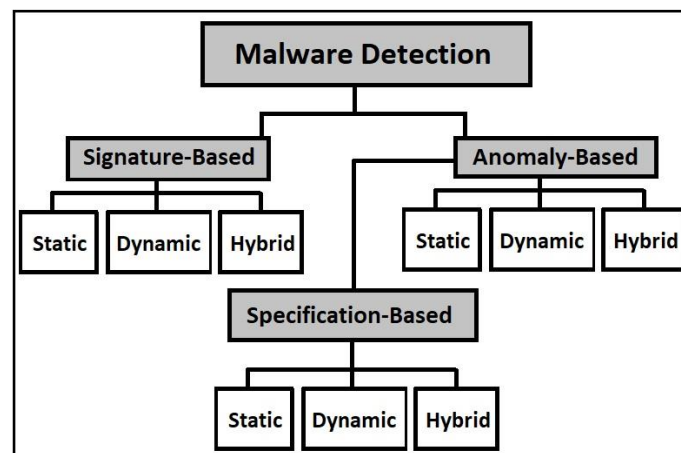


Figure 1.2: Malware Detection Techniques.

Behaviour or Anomaly-Based Detection specify the strength of the recognized malware through the naturalistic (inbred) behaviour. Malware consideration in the anomaly-based detection can take place in two situations; first, the training or learning phase, in which the inbred behaviour can be studied or learned by the detectors, the second phase is the monitoring or detection situation [10]. While in the Signature-based methodology, a predefined database or repository which includes all the updates of the malware like Trojans, viruses, worms, etc. which may be considered as a source of attack or threat to the system at any time must be included in the security system software, and the software must be updated continuously to not miss any newly evolved tactic of attack [10], [11].

Over the years, a technique which considered to be a promising one in malware detection represented by Machine Learning (ML) algorithms, unknown malicious software can be recognized using ML which plays a vital part regarding this issue [12]. Different strategies have been incorporated to detect malware using ML, it was notable that many of these strategies obtained good results, despite this, other problems like the data imbalance problem were obstructing their way to success. It was necessary to test different strategies that combat malware till reaching an optimized strategy which can bypass all the difficulties and problems that may arise using the traditional techniques.

According to the above, the candidate implemented three different techniques in a trial to reach an optimized technique in malware detection. First, implementing traditional ML technique using two different algorithms and testing other algorithms. Second, implementing the One-Class Classification (OCC) technique. Third, and finally implementing the Subspace Learning-based One-Class Classification technique using two different sub-techniques, which obtained the intended results and outperformed other strategies.

1.2 DISSERTATION OBJECTIVE AND PURPOSES

During the proposal stage, it was found that many researchers have studied and proposed different strategies that can detect malware, of course, their results were satisfying, but always there is a question which appears, is the strategy implemented is a comprehensive one? Or, can the implemented technique deal with different problems that can face it? Note that the talk is about the Anomaly-based detection only. More than one problem can face the implemented strategy, either regarding the dataset used or in more precise word, the collected positive and negative samples, or the algorithm used to detect the malware.

Regarding the collected samples, the negative samples always face the problem of being rare or has always a new tactic or shape (especially in malware detection issue) so, the implemented algorithm will be deceived and will not recognize the new mutated malware, or the shortage in negative samples will make an imbalanced set training suffer. Furthermore, the algorithm and what may limit it from performing its job, Curse of Dimensionality may represent the most popular obstacle that affect the algorithm accuracy. Hence, there is a need to find out a strategy that can bypass all types of problems and represent the promising solution for future researches and evolution regarding this topic.

Accordingly, the candidate started from scratch and tested different algorithms and employed three different datasets to reach a satisfying level that can answer all the questions and problems mentioned previously.

The prime objective of the candidate in his work was to detect a suitable and effective or at least a promising solution for detecting malware in the portable executable headers, by employing the various features as a mean for that detection. We must mention that this strategy does not stop at discovering malware in the PE file, but also it was noted during the experiments stage that the results obtained in this dissertation may widen the usage of this strategy to include other types of malware detection. This was concluded after going through different algorithms and touching the previously mentioned problems with each implementation.

1.3 WORK PHASES AND IMPLEMENTATION

1.3.1 Work Stages

This work has been gone through three main stages. It was necessary to obtain tangible evidence without just reading about them, especially in a topic related to cybersecurity, so that it will be possible to build our research and findings on a strong and reliable foundation. Accordingly, the start was from the early beginning, by implementing traditional Artificial Intelligence (AI) algorithms for malware detection inside the header of the PE, Multi-layer Perceptron Neural Networks (NN_MLPC) also, Decision Tree (DT) were served as primary algorithms used to test the detection accuracy of the malware in PE files. Other algorithms like “K-Means Clustering”, “Random Forests”, “Support Vector Machine (SVM)” and “Naïve Bayes” were also tested, note that some of them are clustering algorithms and others

are classification algorithms, according to what is mentioned in [13], to obtain more accuracy, classification algorithms can be used to more effectively detect malware, hence NN_MLPC and DT were the selected algorithms, the reason is discussed in chapter three in which a detail illustration to this stage will be presented. It is important to mention that this stage needed a detailed study, which means that the start was with a strategy and its outcomes were recorded in details, then these outcomes have been used to widen the study and do further experiments to reach more precise and detailed conclusion, accordingly, this stage has been done in two parts each part has been published in a separate work.

The second stage was the One-Class Classification (OCC) stage, wherein more accurate results were attained, this stage has been implemented using One-Class Support Vector Machine (OCSVM) [14] and Support Vector Data Description (SVDD) [15], both are implemented using the LibSVM library [16].

The third and final stage, was the One-Class Classification based on Subspace Learning, which was the dissertation main contribution, two algorithms were used during this stage which were Subspace Support Vector Data Description (SSVDD) [17] and the Graph Embedded Subspace Support Vector Data Description (GESSVDD) [18]. Stages Two and Three were published in one work.

1.3.2 Purpose of Each Stage

Cybersecurity solutions always need a continuous testing to the algorithms and strategies used in detecting the threat, attack or the malware, it will be useless to build a technique or methodology without covering all the possible wholes that it has and see the different points of views, which means that testing or searching for all the vulnerabilities that this technique may have, those vulnerabilities will lead to breach and hence achieve the purpose that this malware or attack has been created for.

It was important to start from an early beginning point, so that an improved vision to the next stages can be obtained, add to that recognizing the positive and negative aspects from each implementation. Accordingly, the start was with traditional AI algorithms. Actually, the number of tests that have been done were enormous, and each test case was documented and then moving to the next test case. This will be clarified more in chapter three.

After finishing all the tests and runs in stage one, it was noticed that the imbalance dataset problem may affect the attained results, so there is a need to enhance the strategy to overcome such problem, knowing that the obtained results were enhanced and the experiments had brought many observations regarding the portable executable header features, those observations were useful in case that stage one was completed to obtain more improved malware detection system and more observations and results, add to that, the observations were useful in the next stages. From this point stage two has been started, since the OCC can deal with the class imbalance issue, it was employed in two different algorithms OCSVM and SVDD, both has used the LibSVM library. Performance was evaluated using different metrics to what was used in stage one, stage two metrics included the FMeasure and Gmean in addition to more metrics. The attained results were enhanced and outperformed previous results and implementations.

Curse of Dimensionality was the main motive that pushed towards going to stage three of the work, which was dedicated to reach the optimized technique for malware detection, so, this stage included testing and then employing a novel strategy represented by the Subspace Learning-based One-Class Classification, two variants of this technique were employed, (SSVDD) and GESSVDD, at this level, results obtained were satisfying and promising, add to that, this technique has bypassed all the problems that any previous implementation can face like curse of dimensionality in OCC and class imbalance problem, so, this technique represents the comprehensive solution that may be the promising technique toward better malware detection not only in the case of the portable executable files but may include other types of malware, add to that, this strategy could also be a promising technique that can serve in cybersecurity several fields.

1.3.3 Datasets Used

All the dissertation experiments and studies were done on the datasets mentioned in Table 1.1, which lists the datasets. All listed datasets have the proper permissions to be used and work on them the way that the researchers like.

Table 1.1: Datasets Used.

	Dataset Name	Samples	Features	Classes
1	dataset_malwares	19611	77	2
2	CLaMP	5184	69	2
3	pe_section_headers	43293	4	2

1.3.4 Datasets Used

During the research period three works have been published, one of them is an IEEE conference paper, and the two other papers were a Journal papers, one of them located at Q2 Quartile, and the other one is located at Q1 Quartile. Table 1.2 include all the details for each paper and the place where the paper has been published in.

Table 1.2: Published Papers.

	Paper Title	Conference or Journal Name	Impact Factor	Quartile	Index
1	Effect of PE File Header Features on Accuracy	2020 IEEE Symposium Series on Computational Intelligence (SSCI), December 1-4, 2020, Canberra, Australia DOI: 10.1109/SSCI47803.2020.9308507			
2	Impact of Portable Executable Header Features on Malware Detection Accuracy	Computers, Materials & Continua, vol. 74, no.1, pp. 153–178, 2023.	3.86	Q2	SCIE
3	Malware Detection with Subspace Learning-based One-Class Classification	IEEE Access ISSN: 2169-3536	3.476	Q1	SCIE Under Review

1.4 CONTENT

This dissertation will continue as follow. Chapter Two will present the definitions of important terminologies included in the dissertation, also, it will include all the methodologies and programming languages employed. Chapter Three will present Stage One, which includes the traditional AI implementations. Chapter Four will include Stages

Two and Three, all the work related to implementing OCC and Subspace Learning-based One-Class Classification and the results obtained. Finally, Chapter five will summaries and conclude the whole work and results obtained.



2. TERMINOLOGIES AND TOOLS

2.1 INTRODUCTION

The subject of this dissertation consisted of heterogeneous components, each component has its own role, collectively they work as a system which intends to detect malware. To make this point clear, there is a need to illustrate each component whether the component is a software or hardware, note that hardware components is limited to computers that have been used to perform the experiments. For the software components the start will be on the subject that the experiments and tests has been done on, which is the Portable Executable file. Machine learning is the main title for the tools used to work as malware detection tools. NN_MLPC and DT primary tools that were used first in malware detection, then OCC represented by OCSVM and SVDD, both tools employed the LibSVM library as a base for their work. Finally, the last tool used is the OCC with Subspace Learning, two variants of this technique were employed, SSVDD and GESSVDD. Programming languages that have been used in this work are **Python** and **MATLAB**. The next sections in this chapter will give an introduction to each tool and its role in the work.

2.2 MALWARE AND PORTABLE EXECUTABLE FILE

2.2.1 Malware

Malicious software (Malware) is an illegal code that aims to damage computer or networking systems, by interrupting the system proper work. The key point in malware is to find a hole, a backdoor or a vulnerability point like code caves in Portable Executable (PE) file, that make the road easy to enter the system. This means that the badly intended people whose responsibility is to invent malware and implant it inside the computer, will always study the structure of the different files and programs that may exist in the computer, add to that they must know how the operating system (OS) deal with the files and programs and the form that the OS use to execute or present files.

Starting from the illustration above the war started between malicious people and cybersecurity people, malicious people always invent new techniques to deceive security people, hence the malware took different shapes, according to [9] modern malware has the characteristic of mutation and encryption which increases the different malware types daily, because of these characteristics, the malware is more reluctant to security application.

Malware is characterised by its gloomy behaviour, which typically contradicts the computer user's wishes. The intension always represented by gaining an unauthorized access to computer and network resources, compromising the network integrity, or ripping off personal data without the system owner's consent [9].

Malware can take many names or forms, all the names of malware have the same malignant target, Trojan horse, spyware, adware, keyloggers, spam, ransomware, sniffers, rootkit, worms and virus, etc., many of them can be found in more than one variation [19], [9].

Business Software Alliance (BSA), or the software alliance, established in 1998, the main role of this trade group is to fight copyright violation of the software produced by the members of BSA. Knowing that some of its members are Oracle, Adobe, Microsoft, Nikon and others. According to [20] in 2011, the ninth annual BSA study reported that pirated software has been used by 57% of computer users worldwide, also [20] mentioned that pirated software before the Internet era can be exchanged only through face-to-face or through media like disks. Unfortunately, Internet connections has made the exchange of pirated software easier. Finally, they stated that the Internet is the source of 45% of fraud software [20].

Malware as stated by [21] can grow in number and develop in severity each day, and represents a major danger to Internet security. Following are some types of malwares with a brief definition to each [21]:

- a. Spyware: It is type of malware that can perform spying activities like search history monitoring on behalf of a third party, and transmit custom advertising to that third party.
- b. Adware: Can be considered as a subset of spyware, the aim of adware is to make a profit to the developer, the only target of adware is to display computer advertisements.
- c. RootKit: A malware that give unauthorized user an administrative access. RootKit is usually hidden and unnoticed, sometimes, to find them and remove it is an extremely challenging task. It can give the intruder a permission higher than it is permitted.
- d. Keylogger: This type of malware records and stores all keys pressed by the user, including credit card number, passwords and other sensitive user information.

- e. Ransomware: This type of malware blackmails the victim by encrypting his data in the computer and requesting to pay an amount of money so that the user can get the decryption key and retrieve the data. Usually, the computer infected with ransomware will be Frozen, and the user will be unable to access any file, the screen can be used only to display the attacker's request.
- f. Virus: Simply, a piece of software that has been modified, launched or loaded without the permission of the user.
- g. Trojan: This malware appears as a legitimate software, people download such software since they thought that this application is legitimate, such malware can propagate through social engineering.
- h. Worm: Worm is similar to virus except that it can propagate to other devices via the network.
- i. Backdoor: This type of malware occurs usually before other type of malware attack occur; hence, it is not used individually. This malware opens the door to the attacker in different way to access the device.

2.2.2 Portable Executable

An overview of the PE file format is presented in Figure 2-1. The structure includes two main parts, the PE header and then a series of sections. This type of files is used by Windows executables like DLL, object and other files [9], [22]. The format of the PE file includes a group of information that the Windows operating system can use to execute the files. As illustrated in Figure 2-1 the header part represents the metadata that describe all the tables since this part contains all of the information about the data stored in the PE file [9], [22].

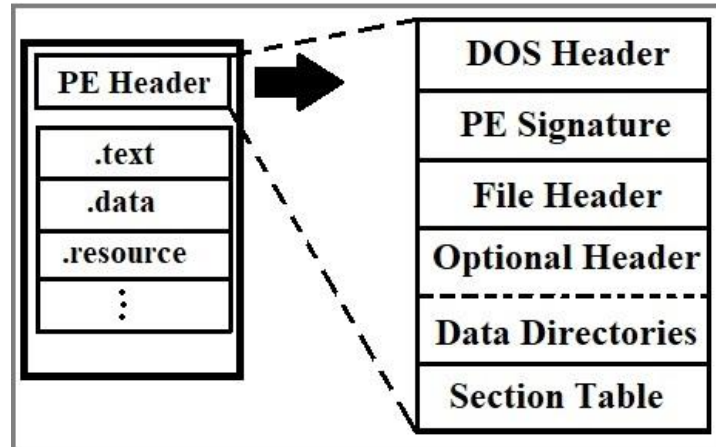


Figure 2.1: The Format Overview of a Portable Executable File.

Each of the components that make up the PE Header—DOS Header [22], PE Signature, File Header, Optional Header—which contains the Data Directories and Section Table—has a specific function. The DOS Header, in particular, is in charge of determining whether the file is legitimately an executable file. The PE Signature, whose value is always set to "PE \ 0 \ 0," designates the file as a PE file. The File Header and Optional Header provide some helpful information, such as compile-time, the size of the code section, the size of the file as it loads into memory, and the number of sections the program includes. The last field of the optional header contains an array of data directories whose purpose is to identify the locations of key executable information components of the file, such as the resource table (which manages resources like the embedded images in the portable executable file) and the import or export tables (which handle functions that are imported or exported). The size and location of the table that is allocated in each row are contained in the rows of the data directories [22]. The information about the program sections is finally located in the component Table, the final component in the PE header. A specific row can be used to specify each portion of the PE file. Section Header provides information about the section, such as its name, size, and address. The file's actual contents, which contain global data, code, and resources like images and icons, are located after the header [22].

2.2.3 Malware in Portable Executable File

The PE header is used by malware creators to propagate their malicious code and breach systems. The PE files will be used to store and spread the malicious contents. In general, the Windows systems use the portable executable file format to store and execute the .exe and other similar files whether these files were benign or malicious. As mentioned previously,

the portable executable header includes a description data which has the information about the structure of the portable executable file itself, if a malware resides in the portable executable file, it will affect the system and malfunction the activity of it. The most effective malware is the second-generation malware, this malware uses techniques like obfuscation and encryption, hence traditional malware detection software will not be able to detect this type of malware [9].

Going back to the portable executable file structure, operating system will use header to access the programs because it contains all of the necessary information. The MZ character is the start of all portable executable files. Compatibility is the responsibility of the DOS STUB. The actual signature employed in identification of that file will reside in portable executable signature. Entry point and subsystem of the file are residing in image optional header. Information about the way of loading the executable into memory can be found in the section table. Finally, the SECTIONS can be located at the conclusion of the format, which contains the information that the executables use. Sixteen data directories can possibly be defined in portable executable file format, but different files have different number of data directories, as an example, tracing the executable execution is the responsibility of debug-information which resides in debug directory. Accordingly, malware inventors avoid detection through removing their programs of debug information. Resources are located in the resource directory and include menus, dialog boxes, icons, and more, so, the developers of malware remove parts of data directories like debug information and resources, so that they can reduce the compiled executable size [22], [9].

Code Caves [23], [24], is a terminology that can be found in the field of cybersecurity and reverse engineering, The original software and a software with a code cave are both displayed in Figure 2-2. In portable executable file, Code caves comprises the space resulted from the Null Bytes (Empty bytes) whether they are ordered or random. In portable executable compiled file, Assembly language can be used to alter and modify the software execution flow through code caves, allowing for the injection of different codes.

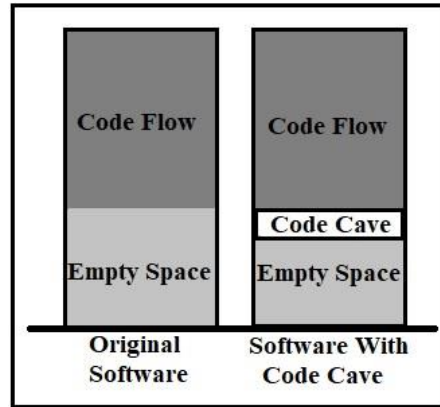


Figure 2.2: Original Software and Software with Code Cave Comprise the Software Structure.

Shellcode can be defined as the instructions set that can be injected in to the exploited program, then this program will execute them at compile time [23]. Hackers and malware inventors create shellcode that can be injected to code caves through malware, those injected shellcodes can create backdoor on the computer. Hackers can exploit the vulnerabilities that the injected shellcode program has been created to hack the computer system. There are several ways to insert shellcodes into an executable program, including application linking, binding, and embedding methods [24]. Shellcode has the ability to directly change the registers of the Central Processing Unit (CPU) and the functionality of the exploited program.

Detecting code caves in portable executable file needs to disassemble the program by the attacker, this can be done using a disassembler tool. The code cave's dimensions must match the size of the shellcode that is supposed to be injected, at the very least. Hence the attacker needs to search manually for the suitable code cave location [23], [24].

To perform operations with code caves, an unused space must exist in the software. Then, new code addition is performed to this free space. Assembly language instructions like jump or call can be used to achieve the transition to the code cave from the initial source code to run the recently added software. When the execution of the new code is done, the original normal code flow will be retrieved through jump or ret instructions. Figure 2-3 illustrate both the normal flow of code execution of a non-code cave software and the flow of code execution in a software that include a code cave. In Figure 2-3 -a-, points A, B and C constitute the normal flow of the execution process. While in Figure 2-3 -b- and after the execution process go through point A, it will be redirected using the jump or call instructions to the code cave to run the shellcode; after completing the instructions' execution in the code

cave, the execution will continue from point B using the required instructions before moving toward point B [23], [24].

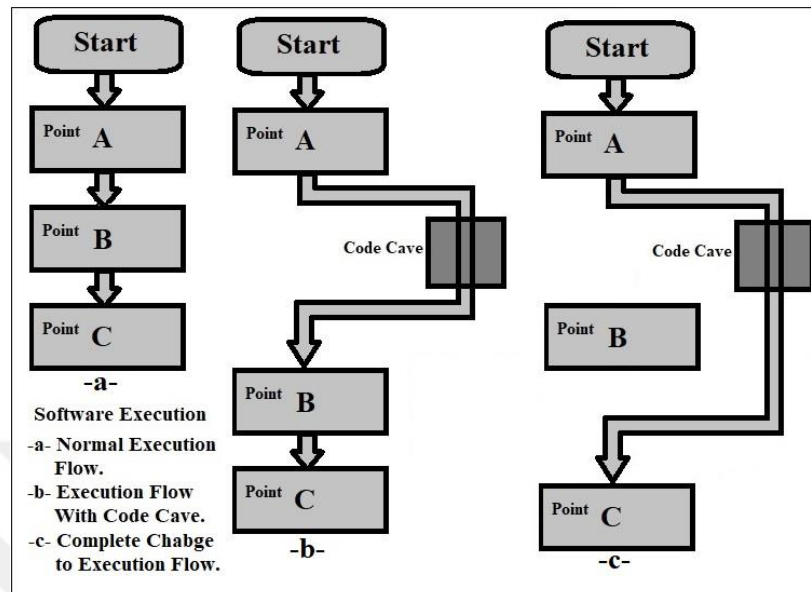


Figure 2.3: Software Execution: a) Without Code Cave, b) With Code Cave, c) Complete Change to Execution Flow.

The most dangerous aspect of code caves is that it can change the execution flow completely, Figure 2-3 -c-; this will mean that the execution flow will not return to B, rather, the execution flow will proceed with the required instructions starting at point C. Some researchers like [23] suggested a tool named CodeCaveFinder which can search the portable executable file to find the suitable size code cave that can fit the injected software. In this way, there will be no need for the manual search and finding the suitable size code cave will be easy. Additionally, the code cave's beginning address, the section they are living in, and their size can all be displayed [23].

Sometimes injecting a code into code caves will be useful and bring some benefits, so, it is important to mention those benefits. when writing a new software, code caves can be used to make some small changes to the compiled software. Software recompile will be not necessary since commands can be written directly into the portable executable file using hexadecimal number system. Actually, this is how assembly language commands can be added to a disjointed portable executable file [24].

Also, from the advantages of code caves, it is possible to modify the portable executable file without compiling it, if the portable executable file does not have source code. An additional benefit is that breakpoint can be placed at the start of the cave of the code, it is possible to easily make a test to the code which resides in the code cave. As for the disadvantages that code cave possesses, there is a need to bypass or fake the file if there is need to pass any physical CRC (Cyclic Redundancy Check). Another disadvantage, when there is no code cave with the required size in portable executable file, this may lead to corrupt the portable executable file if some of the old, original code is overwritten by the injected code, which exceeds the size of the code cave [24].

2.3 NON-PORTABLE EXECUTABLE FILE

It is accurate to say that the primary goal of this dissertation experiments is to identify an appropriate method for detecting malware; in this case, the method is to identify malware contained in a portable executable file. But there must be a point that can present the non-portable executable file just to show that malware can come from other types of files also as it can also come from other sources. In this subsection, a brief introduction will be presented to the non-portable executable file.

The term "non-PE file attack" refers to attacks that don't make use of a .exe or .dll (Dynamic-Link Library) files. Visual Basic Script, PowerShell and JavaScript are the most repeated examples of non-portable executable files. Those files can evade detection by pretending to be someone else. The script can be found in a PDF file or an office document [25].

Another popular type of files that can include malicious behaviour is the Zip file. Files with extensions like .SCT, .LNK or .HTA are a script files which can be activated. Those files may include malicious content and pretend to be a legitimate script. Once the file is unpacked the virus will reside inside the device. The malicious file may have the ability to move out to the cloud using the path of the system, and this will allow it to work undercover even if the user try to search for it. Although Google in 2017 started to block the .js files, this was an ineffectual step because it failed to detect the exact number of JavaScript malicious files which travels through Google's service. Social engineering tactics were combined with links that are malicious or a non-portable executable attachment with extensions like HTML or Word documents (.docx) to perform professional attacks that targets the emails. Main

characteristics which nominated the non-portable executable files to be used by the attackers are being polymorphic, easy to modify and obfuscate [25].

2.4 MALWARE DETECTION TECHNIQUES

Malware can be separated into two categories: first-generation malware and second-generation malware [21]. The functionality of the device and process decides the category of the malware. First generation malware structure remains the same as long as it is working, while the second generation has the property of mutation, which means that a new system is created after each iteration. This property of the second generation made the detection and isolation of the malware a complex task. Machine learning strategies has proven their effectiveness as being the solution to defeat the malware in recent years, Figure 2.4 illustrate the different strategies that can be used to detect malware [21], [26], [10], [11].

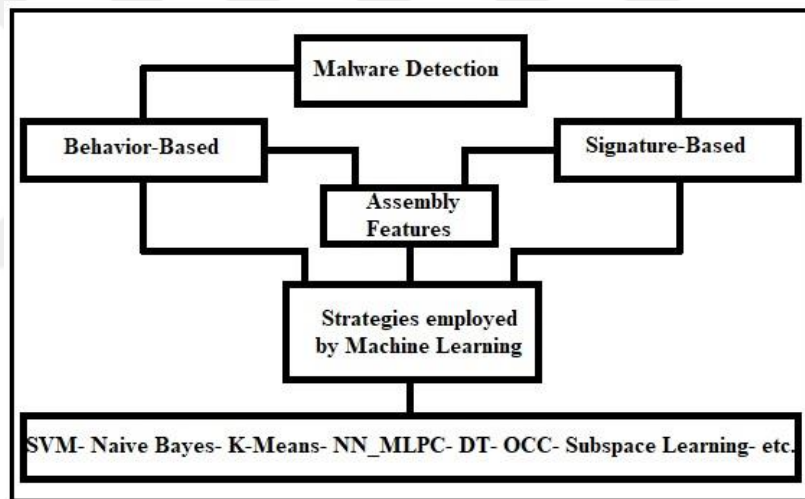


Figure 2.4: Malware Detection Strategies.

Three main strategies to analyse the malware are [21], [26] Static Analysis, Dynamic Analysis and Memory Analysis, see Figure 2.5 which illustrate the types of malware analysis.

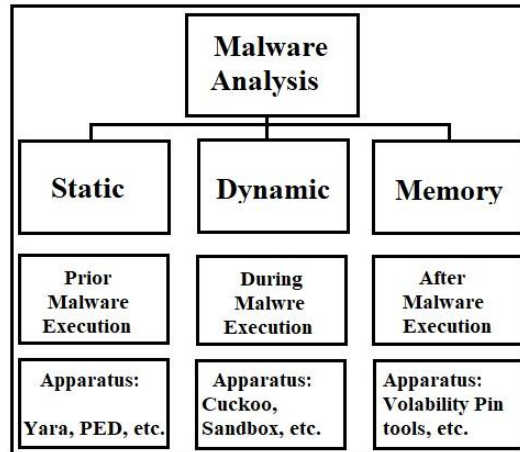


Figure 2.5: Malware Analysis Types.

Static Analysis is used for signature-based; this technique is done without executing the malicious code. Assembly code, DLL, EXE, byte code documents, etc. are the most likely types of files for static malware analysis. To analyse the files, extracting the static features from the files. Tools such as ssdeep, PEiD, strings, IDA pro, Olly-Dump, Olly-Dbg, Pafish and other tools are used to analyse the files statically [21], [26].

Dynamic Analysis is used for freezing the spread of the malware to other devices in the systems, The process of performing analysis dynamically involves examining the malware's operations while it is in operation on the system. Dynamic malware analysis deletes the dynamical features of the malicious code, this includes the Cat, TRACKTRAK, Anubis, CWSandbox, etc [21], [26].

As for the Memory-damaging Malware Analysis [21] the test for the maliciousness will be occurred after executing the malicious code. The features used for memory analysis are application programs, network services, rootkits link, injection code, shared resources, etc., the tools used in such analysis may include Valgrid, volatility, pin tools, etc. [21], [26].

Machine learning can be considered as a powerful tool that can deal with different types of malwares, and the variety in machine learning algorithms makes it easy to choose the suitable algorithm for each type of malware, or choosing the suitable algorithm to control a wide range of malware types.

Supervised, unsupervised and Reinforcement techniques are the main techniques used by Machine learning. In Supervised learning technique like DT, SVM, Bayesian, k-Nearest

Neighbour (kNN) and Artificial Neural Networks (ANN) are some of the examples that use Supervised learning where the algorithm attempt to infer a feature from a dataset that includes a labelled data. In the second technique the unsupervised learning an untagged data is used to find out its hidden structure or shape. Algorithms like SoM (Self-organizing map), hierarchical clustering, neighbourhood-based methods add to that K-Means. The Reinforcement technique differs from both supervised and unsupervised learning techniques. Since it does not need a labelled pairs as the supervised learning need, also, it does not need any action whether a sub-optimal action or any other action to be corrected. It does only need to find a balance between current knowledge and uncharted territory [21].

2.5 ALGORITHMS AND TOOLS USED

This section displays all the tools and algorithms participated in this dissertation work. This will include all the hardware and software used. This section will simply outline the definition or the prim characteristics of the terminologies without illustration to the reasons or the motivations of employing this terminology, such illustrations, like reasons and motivations will be presented later.

It is also important to mention here, that the algorithms used, for example, NN_MLPC and DT were not the only algorithms which were tested, other clustering and classifications algorithms were tested but NN_MLPC and DT were the only algorithms used in the first stage experiments of the dissertation.

2.5.1 NN_MLPC

Neural Networks Multi-Layer Perceptron (NN_MLPC) can replicate how the human brain learns [27]. It is a supervised learning algorithm. Using the training on a dataset the function in equation 2.1 is learned to the multi-layer perceptron algorithm:

$$f(.) : R^m \rightarrow R^o \quad (2.1)$$

Where the dimension of input number is represented by m and dimension of output number is represented by o. As an example, if a group of attributes (features) are given $X = x_1, x_2, x_3, \dots, x_m$ and the target is y, a classification function that is non-linear can be learned. It can include a one or more hidden layers or in other words non-linear layers in between the layers of input and output. Figure 2-6 illustrate the hidden layer in multi-layer perceptron (MLP) [28].

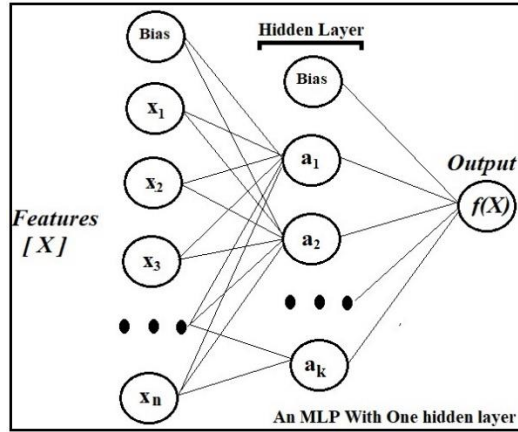


Figure 2.6: An MLP's Hidden Layer.

The layer on the left is the input layer, and it is made up of a collection of neurons that represent the input features. Values are presented to output layer by the final hidden layer, then those values are transformed in the output layer to output values [28].

Multi-layer Perceptron has some advantages like the ability to learn from models that are non-linear, also, it has the ability to on-line learning. From the disadvantages of the MLP is its sensitivity to feature scaling, it has to do hyperparameters tuning like the number of iterations, layers and hidden neurons [28].

2.5.2 Decision Tree

Decision Tree (DT) employ binary trees to process and analyse the data, using the binary trees DT can deduce its predictions. DT can use quantitative information and numerical values as input data. The predictions consider the numerical and categorical values [27].

It is an algorithm that employ conditional statements so that data can be classified. The start of decision tree is with single node, this node will be branched out into two or more branches. The resulted branches will present a different outcome, till reaching the final outcome. When sketching the outcomes and the branches it will appear like a tree. Within the machine learning domain and when there is a need to analyse data, Decision tree is considered to be an extremely important algorithm which accomplish the prediction analysis, classify the data and regression, this is due to that decision tree can break the complicated data to a several manageable divisions. All decision trees are composed of three main parts; the first part is the Decision node, which represents a decision. The second part is the Chance nodes, which

represents an uncertainty or we can call it a probability. The third and final part is the End nodes, which will represent the outcome [29].

2.5.3 OCC

The leading target of OCC is building a model for classification if the dataset is not balanced, in other words if the negative class is not defined well, absent or sampled poorly. Data imbalance will limit the classifiers to be learned efficiently [30]-[32].

In traditional multi-class classification, two or more classes are existing, each class is represented by the data samples. Most classifiers fail to work if the number of samples is not enough or completely absent. In 1975 Minter was the leading person who used the expression “Single-Class classification” when Bayes classifier needed only the class of interest. After that, in 1993 Moya and its group [31] presented for the first time the expression “One-Class Classification” in their work. Other names were created to such classification technique like Novelty Detection, Outlier Detection, Concept Learning or Single Class Classification [30].

Unfortunately, classification problems that may appear in traditional classification like curse of dimensionality, classification error estimation, solution complexity measuring started to appear in OCC and sometimes these problems in OCC become more notable [30]. Due to the fact that in OCC, classification boundary is determined using the positive class only, hence OCC problem become more complex than the problem in traditional classification of two-class. The defined classification boundary surrounding the positive class in the OCC strategy must accept the greatest number of objects that belong to the positive class while reducing the likelihood that outlier objects will be included in the boundary. It is hard to determine which features can be used to separate the positive and outlier class, also, it will be difficult to determine the tightness of the boundary around the data [30].

Another point should be mentioned regarding the OCC. It will be impossible to ascertain which outlier objects—the false positive, or F^+ —will be included or accepted by the OCC due to the lack of distribution and data examples of the outlier class. Also, another complication in OCC is that it is possible to determine only the False Negative, F^- , and True Positive, T^+ , and it is not possible to determine the False Positive, F^+ , and True Negative, T^- . Accordingly, there must be a restricted number of outlier classes. for performance

determination and popularize the accuracy of the classification of an OCC. If the proportion of the outlier is not reasonable, it's possible that the OCC's actual accuracy values are not an accurate or true representation of the metric [30], [31].

2.5.4 Support Vector One-Class Classification

When the negative class is not well-defined or absent, One-class classification algorithms can be employed to construct a model which can do the classification process in such situations. One-class classification can define a class boundary using the available information about the positive class. Concept learning and outlier detection are examples in which the one-class classification is used [30].

Density estimation, reconstruction and class boundary are the main strategies used to tackle the problem of the one-class classification [17]. Gaussian model and mixture of Gaussian are examples of the density estimation. K-means clustering and self-organization maps are examples of the reconstruction model, which employ assumptions on the process that generates the target class.

Examples on solving the OCC problem by utilizing techniques for boundary description are SVDD and OCSVM. Those two examples have been used widely as boundary description strategies, both techniques employ the boundary to isolate the target class from the other classes. SVDD defines a ***hypersphere*** that enclose the desired class, through employing kernel methods, more flexibility can be added to the hypersphere. In OCSVM strategy, a ***hyperplane*** is defined that can distinguish the origin from the data with ultimate margin [17].

Reference [30] mentioned that Tax and Duin constructed a hypersphere that surrounds the positive class in a trial to solve the problem of OCC. Hence using this hypersphere, they distinguished the positive class in the pattern space. Using minimum radius, the hypersphere contains approximately all the points in the data. The technique is called SVDD, this SVDD technique considered all points that were tested as outlier if they fall in regions outside the hypersphere, another case when shrinking the volume of the hypersphere all the positive points that will be outside the hypersphere will be rejected. More flexibility to the hypersphere can be given by employing kernel functions. By implementing two different kernel functions (Gaussian and Polynomial) on several datasets, Gaussian kernels performed better. When using different kernel width, it appears that fewer support vectors are selected

and more spherical description is obtained when the kernel has larger width. Add to that Gaussian kernel resulted in tighter descriptions and requires additional data so that it can result in more flexible boundary, Tax experiments [30] also concluded that if a high dimension is given to the data set the method will be inefficient, also, if the positive class objects have a large diversity in density the hypersphere technique will not work well, because it will reject the target points with low-density [30].

Instead of building a hypersphere, OCSVM constructs a hyperplane [30], [18], the positive class can be separated by extending the distance between the origin and the hyperplane.

Finally, it is important to mention that reference [34] stated that OC-SVM which is Graph Embedded (GE-OC-SVM) and SVDD which is Graph Embedded (GE-SVDD) techniques incorporated information about geometric class, the geometric class information has been expressed using the optimization of OCSVM and SVDD generic graph structures, which played the role of solution regularization [17].

2.5.5 OCC Based on Subspace Learning

If a low dimensional feature space optimized for OCC allows for the definition of a model for one of the classes (e.g., positive class), then the learning process is called subspace learning. This terminology is a common between several algorithms that use this technique for the process of classification. SSVDD [17], GESSVDD [18], Ellipsoidal SSVDD [33] and Multimodal SSVDD [34] are examples of subspace learning-based one-class classification.

In this dissertation, two subspace learning methods are integrated, SSVDD and GESSVDD. Both techniques are employed in a trial to attain better results in malware detection and to bypass all the problems that can face other implemented algorithms, like the Imbalance dataset problem and the curse of dimensionality problem. Actually, the results obtained are promising and outperformed other traditional and OCC strategies. Next two sub-subsections will introduce both SSVDD and GESSVDD algorithms briefly without going into details which can be found in Chapter Four.

2.5.5.1 Subspace SVDD

SSVDD [17], a common OCC method, this method can define a positive class pattern in an optimized low-dimensional feature space for OCC. It is possible to represent complex

models that were present in the original feature space via the establishment of basic class models in the low-dimensional feature space, enabling the non-linear data mappings will accomplish this. This strategy in non-linear problems can supply a good result. Add to that it can simplify the required information in describing the targeted class [17].

2.5.5.2 Graph-embedded ssvdd

GESSVDD [35], This method presents subspace learning as graph embedded problem for the OCC. Also, subspace learning has been merged with the SVDD which has been used in the subspace to produce the iterative technique known as Graph-Embedded Subspace Support Vector Data Description (GESSVDD) [18].

2.5.5.3 Programming languages

Python and MATLAB were the programming languages used in implementing all the experiments in this dissertation.

Taking the particular requirements of the issue into account, the start was with implementing the experiments of NN_MLPC and DT, all experiments were done using Python version 3.8, using SciKit-Learn package which was the exact package of specialization, of course other packages were also used. Variety of programming languages can also be used in Artificial Intelligence applications like Prolog, C++, Lisp and Java. Python was chosen since it incorporates simple syntax and involve truly little coding, also, testing is relatively easier in Python. The primary reason for using Python is its built-in libraries, which include NumPy, SciPy, SimpleAI, matplotlib, and nltk, for a wide range of AI applications.

As for MATLAB, this language was used in all OCC techniques and Subspace Learning-Based One-Class Classification techniques. The MATLAB version was 9.6.0.1072779 (R2019a). MATLAB was incorporated in this part of the experiments since several complex mathematical calculations and preparations such as the preprocessing and fine tuning to the related parameters were needed. Also, the Software libsvm-3.22 is compatible with MATLAB which was the basic software needed to do such implementations.

2.6 HARDWARE

The hardware used to implement the experiments of NN_MLPC and DT is an ACER Laptop computer equipped with an Intel(R) Core (TM) i7-7500U CPU operating at 2.70 GHz and

2.90 GHz, with an x64-based processor, 64-bit operating system, and 8.00 GB of installed RAM (7.88 GB usable). Windows 10 Pro is the Windows Edition. Experiments related to OCC techniques and Subspace Learning-Based One-Class Classification techniques were done in a more advanced computer which incorporates an Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz 2.30 GHz Processor and an installed RAM of 64GB.



3. EFFECT OF HEADER FEATURES ON DETECTION ACCURACY

3.1 CHAPTER INTRODUCTION

Malware programmers always try to find an entry point to the system so that they can implant their malicious software and take the advantage of the malware implantation process. Portable Executable as mentioned in chapter two provides a good environment and residence for the malicious software to be implanted. It was stated in the implementations of [35] and [36] that the native executable of Win32 is portable executable. Portable executable derived its specifications from the UNIX Common Object File Format (COFF). Windows loader important information is encapsulated within the portable executable which considered as a data structure, through these necessary information managements to the executable code can be done. Additionally, there are two empty spaces in the PE file's layout that are used to conceal the watermark. The idea is to build a backdoor that can be utilized in emergency situations, such as when a password is forgotten. Regretfully, since hackers' job is to find gaps in systems to sneak inside, they won't miss such a backdoor.

Accordingly, the candidate decided to select the portable executable file as an example in a trial to reach an optimized strategy that can be employed in detecting malware. The specifications of such strategy should include as minor problems as possible, problems here not only related to the strategy as a whole, but even the components and algorithms used in this strategy, those algorithms and components should be clean and free of any problems that may result in obstructing the work of detecting malware. Since any problem related to any part of the strategy may affect the detection efficiency or may lead to misclassification problem, such problems will open the way to the malware to breach the detection system and work freely but this time with the approval of the system and without any doubts that there is a danger may breach the system or be already existing in the system.

Studying the features of the portable executable file header was the first step that must be done, those features are numerous and play a good role in deciding whether the file is benign or malicious. Also, to reach a strategy that is problem free, there is a need to start from the early beginning, which means to start with an algorithm that employ the features in detecting malware without concentrating on the expected contraindications. Then see the

results and go to other strategy which is characterized by fewer vulnerabilities that may affect the classification process. This is what actually has been done, Accordingly, stage one included the first two papers published [35] and [36] which are presented in this chapter, those papers represent part one and part two of stage one respectively. The experiments and implementation in these two papers started from testing then selecting the machine learning algorithms used then testing and selecting the required features, add to that the experiments took into consideration selecting the suitable number of features and the suitable features to be incorporated in the detection process. After that, a new strategy and algorithms in stages two and three were incorporated which achieved the desired goal in presenting an optimized and promising malware detection system, the strategy is presented in Chapter Four.

Next two sections will present the part one [35] and part two [36] of stage one respectively with the obtained results from each part. Section 3.2 will present part one [35] of stage one and section 3.3 will present part two [36] of stage one.

3.2 PART ONE

This was the candidate's first publication [35]; The purpose of this work was estimating how those features can affect the accuracy level and to use the portable executable headers features as a malware detection facility. The work employed various numbers of features. Two algorithms have been incorporated to do the same experiments, NN_MLPC and DT so that effectiveness evaluation can be done. Of course, many other algorithms including classification and clustering algorithms were tested to reach the highest precision in algorithm selection as possible. The relative performance was evaluated using a specified control data. The standards used in evaluation was the obtained level of accuracy through variation and large number of experimental studies.

Each study began with a specific set or collection of features, to which more features were progressively added. Accuracy level was registered so that at the end the highest level of accuracy obtained can present the quantity of features and the specific features that were utilized. This tactic demonstrated that not every feature affects the accuracy level in the same way. Next subsections will introduce the detailed steps of doing the experiments. Also, it indicated that including a large number of features will not necessarily result in a high accuracy level or improve the accuracy level.

3.2.1 Introduction

The job of the malware is to perform illegal job. The spread and evolution of malware made the antivirus developers search and invent more signatures and be sure to keep their antivirus program up to date. Actually, antivirus developers consider the problem of malware detection as a major one [37].

Obfuscation can be considered as the common feature in malware, this feature may lead to increase the complexity, also, this feature can hide the malicious purposes; this will lead to raise the hardness of discovering the malware using the reverse engineering. Recently, one of the most important strategies that uses obfuscation is packing, after evolving the compression and encryption that were used previously [38].

As mentioned previously, the native executable of Win32 is portable executable. Portable executable derived its specifications from UNIX COFF. Windows loader important information is encapsulated within the portable executable which considered as a data structure, through these necessary information managements to the executable code can be done. Additionally, there are two empty spaces in the PE file's layout that are used to conceal the watermark. Beyond that, the idea is to build a backdoor that can be utilized in emergency situations, such as when a password is forgotten. Regretfully, since hackers' job is to find gaps in systems to sneak inside, they won't miss such a backdoor [39], [40].

Analysing the characteristics of the portable executable file's header was one way to comprehend how it might be used maliciously, of course, different studies tried different ways also. Many researchers have suggested to use various features to be employed in differentiating between benign and malicious files, the suggested tool was the artificial intelligence which was incorporated in most of the trials.

This paper offers an analysis of the characteristics through adding the features and see how the accuracy will be affected. Two classification techniques were incorporated in this work NN_MLPC and DT.

Several features were proposed and tested in this work to be used in the process of differentiating the benign from the malware files. The characteristics were gathered from earlier works of different researchers. Those collected features have been used to study their

effect on accuracy. The primary goal of these implementations is to prevent the misuse danger of portable executable file.

The following paragraphs show some related work and implementations that used the PE header features.

The implementation in [41] examined the Windows PE header's properties and made use of the structural information that are standardized in Windows to extract the distinguishing features.

Packed executables can be detected through the features of portable executable, this is what the work of [42] discussed, they mentioned that during packing a change in the internal structure of the portable executable file, this change will make the process of reverse engineering of the file difficult.

The work of [39] proposed the malware detection through mining the portable executable file format information. The first step was extracting 197 features from the portable executable format information, then they applied the feature selection, and classification algorithm was used to train the chosen features.

People in [38] made a trial to detect packed files and malware using fuzzy classification algorithms and static heuristic features. The implementation employed several obfuscation techniques. The work showed that the obfuscation affected the portable executable through making some anomalies in them.

Precise decision for intrusion attacks and malware depends on the process of selecting features and extracting them, this was discussed by [43]. The goal was to create a system that could recognize, comprehend, and stop malicious connections.

In [44], various algorithms such as Random Forest, Naïve Bayes, Logistic Regression, Linear Discriminator analysis, kNN, and Decision tree were employed to distinguish between malware and benign files with a high degree of precision and minimal computational overhead.

People in [45] have studied malware detection and features learn, the study has extracted portion of the portable executable via employing least amount of domain knowledge.

For malware detection, deep learning and artificial immune systems (AIS) were recommended by [46]. The work included two stages, the first was features extraction, the second was employing the AIS system to construct malware detectors.

In order to identify malware, individuals in [37] have suggested a technique that combines characteristics taken from the portable executable section table and header using various classification algorithms. The features were selected after extraction using forward selection strategy.

Finally, it is noteworthy to mention that the findings of this study, which examined how portable executable header features affected the accuracy of malware detection were as follow: NN_MLPC attained 0.979x and DT attained 0.987x, knowing that other high accuracy levels have been obtained but what is mentioned above is the highest records.

3.2.2 Criteria and Basis

This subsection will present the incorporated algorithms, the strategy of selecting and numbering the features, some criterion used and finally a discussion of the algorithms that have been tested before taking the decision of selecting the two incorporated algorithms.

3.2.2.1 Algorithms used

The algorithm selection was proceeded by testing several various algorithms so that a decision can be correctly made in choosing the appropriate algorithm. Examples of the tested algorithms are Naïve Bayes, K-Means Clustering, Support Vector Machine (SVM) and Random Forests. Both clustering and classification techniques were tested. But a general decision that were first taken was to select a classification technique, this was due mainly to what other people recommended like [13] who stated that dealing with malware is better using classification technique. The second decision was to use two different classification techniques not only one, the aim of this decision was to compare results and see how algorithms will vary in the obtained accuracy depending on the number of features used, additional reason was to study the behaviour of each feature in each algorithm and its impact on accuracy, and this what was studied in details in part two. Accordingly, the selected algorithms were NN_MLPC and DT.

The reasons behind selecting those particular two algorithms were reasonable, for the NN_MPLC, this algorithm can build an AI machine which needs a human intelligence, so, this algorithm can mimic the brain of the human, which can always learn new things [27]. Accordingly, it was supposed to use this classification technique to build a malware detection machine which will work on the portable executable file and this machine can obtain better performance. Returning to the DT algorithm, this algorithm incorporates a distinctive approach to process and analyse data, this approach is the binary trees, using this approach representing the values using numerical and categorical forms can be taken into consideration in the prediction process. Hence, both numerical values and quantitative information represents the forms of the input data [13]. Two main reasons led to make DT a good candidate in this work which are, between the various tested algorithms, DT yielded the highest accuracy and the second reason is the popularity characteristic that this algorithm has, where it is recommended and incorporated by several previous implementations, of course with acceptable levels of accuracy and performance.

3.2.2.2 Feature selection

The reference [47] was the origin of the control dataset that was used in this paper. PE defines the form of the executable and the section layout, A binary form data structure called a portable executable is present in the Windows operating system and uses files with extensions like .dll, .exe and .sys. The portable executable file also keeps the description's information, this information includes program code and used by the loader of the Windows [48].

Portable executable file header was the study material of many researchers, those researches have employed different strategies to identify the most influential features. According to each researcher's conclusions, various numbers of features were extracted, since each researcher had concluded different group of features which has the best influence and the most sensitive. As a result, different number of features were elected by each researcher. What was striking was that some of the features were commonly used by the researchers like DllCharacteristics, MajorImageVersion, MajorOperatingSystemVersion and others. The conclusion is that, based on experiments and implementations of prior works, a set of features can be deemed as the most effective features to achieve good results and diagnose the portable executable more accurately.

This paper employs a collection of features, those features were collected from previous studies and implementations that other researchers did. The aim is to include as wide features range as possible.

Parisi [13] incorporated eight features only for malware detection, such as MajorLinkerVersion, AddressOfEntryPoint and MajorImageVersion. While Zatloukal [48] mentioned in his paper that Yibin Liao stated in his paper in 2012 that their work has analysed the portable executable headers using the Header-parser, after that a comparison between the malware and benign files occurred, based on the header parts used. Three rules were stated in Yibin Liao paper:

- a. If the values of DLLCharacteristics, CheckSum and MajorImageVersion are equal to Zero, it can be said that this executable is malware with 90% accuracy.
- b. In the event that SizeOfInitializedData equals Zero, the program might be malicious.
- c. File can be expected to be malware if the section's name is unknown e.g., 6dnn4fh4, while the section's name of a benign file must have a meaningful name.

Twenty-five features were listed by [46], those 25 features were the nominated features by the work for obtaining a better malware detection accuracy, the selection of those 25 features were done using statistical mean after the extraction of 53 header features. Features like SizeOfHeaders and SizeOfUninitializedData were employed by Kumara et. Al. [44], some of the used features are raw features and others as binary features. Five features were selected by [41] as being the top five features, those features were concluded to include a special difference between malware executables and benign ones. Those features are: MajorImageVersion, and Checksum, UnknownSectionName, DLLCharacteristics and SizeOfInitializedData.

Part one and two of stage one has incorporated all the features that were mentioned above in [13], [48], [46], [44] and [41]. Six features were excluded, four of the six features were not included or does not exist in the control dataset used in this work, two features are not applicable to this work; the other does not affect accuracy, this was approved by the tests that have been made.

Table 3.1 list the Features excluded from the set of features used by the implementations.

Table 3.1: Excluded Features.

No.	Name of Excluded Feature	Reason of excluding
1	ResourceSize	No effect when removed
2	UnknownSectionName	Not found in DataSet
3	Entropy	Used for Packed File
4	SizeOfRawData	Not Found
5	CreationYear	Not Found
6	BaseOfData	Not Found

3.2.2.3 Feature numbering

As mentioned in before, features incorporated in this work are taken from five different previous implementations, the total number of incorporated features was **Twenty-Nine** features (29 features), those features are listed in Table 3.2. According to their significance, the features are arranged in the table, that is, shared features across multiple implementations are mentioned first. This table 3.2 will be the reference table; features can be identified during the implementation according to their sequence number in the table instead of using their actual names.

Table 3.2: Features Used Sorted According to Their Importance.

No.	Feature Name	References used the feature
1	DllCharacteristics	[13], [46], [44], [41]
2	MajorImageVersion	[13], [46], [41]
3	MajorOperatingSystemVersion	[13], [46], [44]
4	SizeOfStackReserve	[13], [44]
5	AddressOfEntryPoint	[13], [46]
6	Characteristics	[46], [44]
7	SizeOfHeaders	[46], [42]
8	SizeOfInitializedData	[46], [41]
9	SizeOfUninitializedData	[46], [42]
10	MajorSubsystemVersion	[46], [44]
11	MinorSubsystemVersion	[46], [44]
12	Checksum	[46], [42]
13	ImageBase	[46], [44]
14	MajorLinkerVersion	[13]
15	NumberOfSections	[13]
16	SubSystem	[46]
17	MinorImageVersion	[46]
18	SizeOfStackCommit	[46]
19	e_ifanew	[44]
20	e_minalloc	[46]
21	e_ovno	[46]
22	Machine	[46]
23	PointerToSymbolTable	[46]
24	NumberOfSymbols	[46]
25	Magic	[46]
26	SizeOfCode	[46]
27	BaseOfCode	[46]
28	SectionAlignment	[46]
29	FileAlignment	[46]

3.2.2.4 Standards of test_size and random_state

It is important to define some criterion before doing the experiments, Test_Size and Random_State are two important criterion that needed to finetune, so that a better performance can be obtained.

- a. Test_size: dividing the dataset as training and test sets is the role of Test_size, selecting the Test_size was according to what is recommended in previous implementations 70-30 ratio which means 70 percent training and 30 percent test is recommended by [44]. The selected values for Test_size was 0.3 and 0.15 since many implementations have recommended those two ratios, add to that using these values in the testing phase, some cases have obtained a better result.

- b. `Random_state`: the selected values for this criterion are 10 and 3. Before predicting any data, there is a need to train the model. Accordingly, it is crucial to separate the dataset into testing and training sets, rearranging the data by specifying `Random_state` will be an important issue to not bring the biases in the process of data prediction. The biases can be illustrated by the following example; if a dataset that includes the prices of certain item, the first eight rows of the dataset include the prices of the cheap part of this item, while the lower eight rows include the normal part of this item, Let's say the set needs to be split into a 70% training set and a 30% test set. Biases will arise if the shuffling procedure is not used before splitting the dataset into training and testing sets, the 70 percent training set will always include a majority from the cheap part of this item, and the test set will include majority or all of the normal part, this will yield a problem in the prediction process. Hence applying the shuffling through selecting values to the `Random_state` will be an important and necessary issue

In this work, the dataset is shuffled before data splitting is applied; the shuffling control is defined by the `Random_state`. Following a series of experimental tests, it was determined to repeat all runs with a `Random_state` of 3 with a `test_size` of 0.15 and a `Random_state` of 10 when the `test_size` was 0.3.

It is important to emphasize that according to the previous runs implemented during the testing phase, Notably, the best results were obtained with `test_size` of 0.3 and 0.15 and `random_state` of 3 and 10, respectively. Because of that during the experimental runs same values have been used.

3.2.2.5 Implemented algorithms

According to the reasons mentioned above, supervised learning (classification) technique was chosen to implement this work. Based on that and in order to establish a reliable platform for this work, several classification algorithms were tested as a start. Random Forest, Naïve Bayes, and SVM were the supervised algorithms that were tested in addition to the `NN_MLPC` and `DT` algorithms that were selected for use in this work.

There were two classification algorithms developed: one for the `NN_MLPC` and the other for the `DT` algorithm. For every algorithm, two runs were performed: one with a `test_size`

and random state pair of 0.3 and 10, and the other with test_size and random state pair of 0.15 and 3.

Every time a.csv file was created for a set of runs, this set of four runs included that specific.csv file. The Scikit-Learn package was utilized in Python version 3.8, the programming language used to implement the algorithms for NN_MLPC and DT.

Figures 3-1 and 3-2, which depict the NN_MLPC and DT, respectively, provide examples of the algorithms that were used.

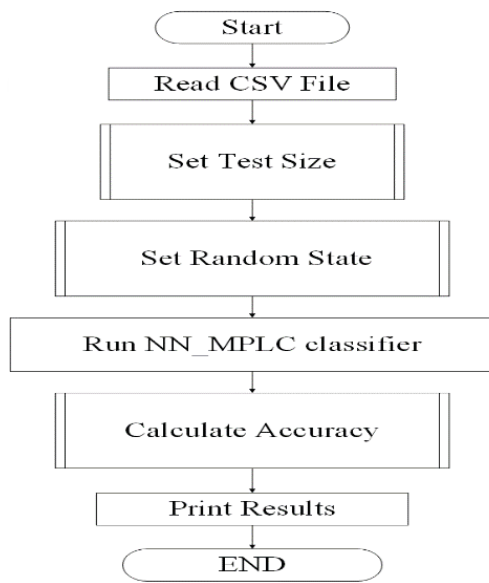


Figure 3.1: NN_MLPC.

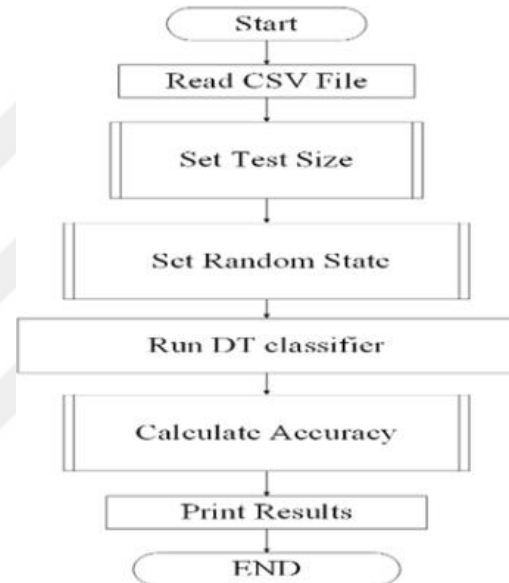


Figure 3.2: DT.

3.2.3 Procedure

The work procedure included the testing phase for the algorithms used, then the phase of implementing the actual experiments after making all the preparations which included the .csv files that will be used in the experiments and the programming language preparation and coding, note that with each run some parameters must be changed in the program to accommodate the experiment required to be done.

3.2.3.1 Algorithms testing

As mentioned previously, several algorithms have been tested, the aim was to avoid random selection to the algorithms, the testing process was complemented by choosing two

algorithms instead of one, so that it is possible to avoid any mistake that may ruin the experiments and their results.

In order to test the algorithms incorporated in this paper, a combination of features was selected from the control dataset of [47] then performing a set of four runs were on this collection of features. Results obtained were contrasted with those of [13], features selected in this testing process were (5, 14, 17, 3, 1, 4, 15) according to Table 3.2. The following are the resulted accuracy from the testing process:

- a. When test_size and Random_state was set to 0.3 and 10, respectively, NN_MLPC yielded an accuracy of 0.968982 and DT produced 0.982151.
- b. The NN_MLPC yielded an accuracy of 0.970849 with a test_size of 0.15 and a Random_state of 3, whereas the DT produced an accuracy of 0.982393.

Comparing what was obtained with the results presented by [13]. Regarding the Decision tree, obtaining 0.982x is better than the 0.962x which was presented by [13]. Taking into consideration that the feature ResourceSize was included in the implementation of [13], but it was not included in the testing process, to make things fair the feature ResourceSize was removed from the dataset of [13] before executing the run (note that the dataset that [13] used is different from the dataset that experiments were done on). As for the NN_MLPC the obtained accuracy level was good and near to what is obtained in DT, taking into consideration that [13] did not take the NN_MLPC into consideration.

After the algorithm testing process, all the features will be taken into consideration and will be studied using both algorithms. Test_size and Random_state will be paired off twice in the study: once with Test_size of 0.3 and Random_state of 10, and again with Test_size of 0.15 and Random_state of 3. This implies that four runs will cover each set of features under study.

To make things easier, descriptive names were given to each run of the four runs, the names were as follow:

- a. NN_MLPC with (test_size= 0.3, Random_state=10), the name will be 'NN_0.3_10'.
- b. NN_MLPC with (test_size= 0.15, Random_state=3), the name will be 'NN_0.15_3'.

- c. DT with (test_size= 0.3, Random_state=10), the name will be 'DT_0.3_10'.
- d. DT with (test_size= 0.15, Random_state=3), the name will be 'DT_0.15_3'.

3.2.3.2 Implementation process

Initially, the runs were to begin with the first three features (refer to table 3.2). After that, single feature was added, a new set of features will be created in a new separate .csv file and got ready for the next run, this procedure was carried out until all of the features indicated in Table 3-1 were included.

Justification for choosing the initial three features as a starting point was because those features were common in several implementations. The prominent results obtained are summarized in Table 3.3.

The five steps below are included in a single group of runs:

- a. Establishing a CSV file for the execution and copying the designated characteristics to that CSV file.
- b. The Python code that has been set up for the experiment now imports the.csv file.
- c. At this point, set the necessary parameters in the Python program, such as test_size and Random_state.
- d. After preparing parameters in the program for both algorithms NN_MLPC and DT, now a run to the program was carried out for both algorithms.
- e. The results obtained from the current run are recorded with the name of the run and its parameters.

For each algorithm, a total of 54 (Fifty-Four) pairs of runs were performed to cover the addition of the features listed in Table 3.2 and also, to study the effect of each feature.

3.2.4 Performance Outcomes

Table 3.3 introduce the result summary; important milestones are given in this table. The following points define the header of each column on the table, since putting them in the table will take some space, also, this will be done in the next some tables.

- a. The run type which is defined previously.
- b. Lowest number of features that made the run produce 0.97x accuracy. Features to 0.97x.
- c. Lowest number of features that made the run produce 0.98 accuracy. Features to 0.98.
- d. The first accuracy value produced in the 0.97x range. First Accuracy in 0.97.
- e. Within the 0.98x range, the initial accuracy value generated. First Accuracy in 0.98.
- f. Maximum or highest accuracy obtained.
- g. How many features are present to get the highest level of accuracy. Features for maximum accuracy.

Table 3.3: Result Synopsis.

1	2	3	4	5	6	7
NN_0.3_10	11	NO	0.971618	No Value	0.978076	21, 28
NN_0.15_3	13	NO	0.971788	No Value	0.979606	23
DT_0.3_10	6	14	0.975527	0.981645	0.984534	25
DT_0.15_3	5	8	0.971448	0.981645	0.987424	19, 27

It is interesting to note that Table 3.3 lists some details regarding accuracy values in the ranges of 0.97x and 0.98x, like the first accuracy value produced in the 0.97x range and the first accuracy value produced in the 0.98x range, minimal feature count that resulted in 0.97x accuracy and minimum feature count that resulted in 0.98 accuracy for the run. To be clear, 0.978x was the maximum accuracy that can be achieved in NN_MLPC, while the maximum accuracy obtained in DT is 0.987x in DT_0.15_3 and 0.984x in DT_0.3_10. The NN_MLPC during the experiments of this paper did not produce any accuracy in the range

of 0.98x. all the high records obtained including the maximum accuracy in NN_MLPC were in the range of 0.97x, on the other hand the majority obtained accuracies in DT algorithm including the maximum accuracy were in the range of 0.98x, because of that the discussion will take both 0.97x and 0.98x into consideration.

The highest accuracy attained in NN_0.3_10 is shown in Table 3-3. which was achieved using 21 features and 28 features and NN_0.15_3 also has achieved maximum accuracy using 23 features, while DT_ 0.3_10 has achieved maximum accuracy using 25 features, DT_ 0.15_3, on the other hand, has used 19 and 27 features to achieve maximum accuracy.

Next subsection will present all the obtained results from the experiments of this work, it will present the run sets and will include all the obtained results in Tables 3-4 and 3-5, those tables will present all the runs and the obtained accuracy from each run, the tables were taken from the documentation of this work as it is without any additions or rearrangements to give the reader an idea on what was done regarding this part one of stage one of the dissertation.

3.2.5 Detailed Results Obtained

This subsection is taken from the documentation of the work which was created during the experimental phase of the dissertation preparations. It is important to emphasize that there will be no major rearrangements or paraphrasing to give the reader a picture on how this work has been accomplished at first. Of course, this subsection represents a part of the documentation, it was not reasonable to put all the documentation in the dissertation since this will take too much space.

All the runs that have been done in the experimental phase are given in Tables 3-4 and 3-5, which provides us with all of the outcomes. from running different features together to get the best accuracy. Note that feature's name was taken from Table 3.2, the number of features is written instead of the name.

According to the previous testing runs implemented, noteworthy that the best results were obtained with Test_size of 0.3 and 0.15 and Random_State of 3 and 10, respectively had obtained the best results, so in the following runs same values will be used.

The first three features were used as the default features found in each run, since DllCharacteristics, MajorImageVersion, and MajorOperatingSystemVersion considered as major features that most of the implementations have incorporated.

Run Set 1: in this Set of Runs, a start with ten (10) Features, and ending with the whole twenty-nine (29) features mentioned in Table 3-2. A record to each run's accuracy was created in Table 3-4, Also number of Features used in each run, names of the used features in the run (Actually, the name was recorded as numbers taken from Table 3-2). Also, there were records for test_size and random_state; test_size was 0.15 and 0.3, and random_state was 3 and 10.

NN_MLPC and DT were employed, each run was as follow: importing the.csv file into the program, setting the Test_size to 0.15 and the random_state to 3, running the program for both NN_MLPC and Decision tree, and recording the obtained accuracies from each run. copying the necessary features to the.csv file that was specifically prepared for that run. Next, run both programs again while adjusting test_size to 0.3 and random_state to 10, recording the accuracy once more, all the data were written to Table 3-4, this procedure has started with 10 features in the first run, then a new feature is added, by adding the feature to a new .csv file specially created for this particular new run, and the procedure was done, this procedure has ended after adding all the 29 features given in Table 3.2.

Table 3.4: All the Runs and Accuracies Obtained Starting with Ten Features.

RUN No.	No. of Sub-Run	No. of Features Used	Features Used. Table-3-	Test_size	Random_	Accuracy	
						NN_MLPC	Decision Tree
Run 1	1	10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	0.15	3	0.9544527532290958	0.9819850441876274
	2	10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	0.3	10	0.9673691366417403	0.9777362338545207
Run 2	3	13	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	0.15	3	0.9717878993881713	0.9830047586675731
	4	13	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	0.3	10	0.97093813732155	0.9796057104010877
Run 3	5	14	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	0.15	3	0.9738273283480625	0.9860639021074099
	6	14	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	0.3	10	0.9765465669612509	0.981645139360979
Run 4	7	15	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	0.15	3	0.9758667573079538	0.9836845683208701
	8	15	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	0.3	10	<u>0.9777362338545207</u>	0.978585995921142
Run 5	9	16	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16	0.15	3	0.9755268524813052	0.9833446634942217
	10	16	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16	0.3	10	0.9775662814411965	0.9826648538409245
Run 6	11	17	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17	0.15	3	<u>0.978585995921142</u>	0.982324949014276
	12	17	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17	0.3	10	0.9773963290278722	0.9813052345343304
Run 7	13	18	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18	0.15	3	0.9772263766145479	0.9860639021074099
	14	18	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18	0.3	10	0.9770564242012236	0.9821549966009517
Run 8	15	19	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	0.15	3	0.9765465669612509	<u>0.987423521414004</u>
	16	19	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	0.3	10	0.9768864717878993	0.9828348062542488
Run 9	17	20	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20	0.15	3	<u>0.977906186267845</u>	0.9847042828008158

Table 3.4: All the Runs and Accuracies Obtained Starting With Ten Features “Table Continued.”

	18	20	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20	0.3	10	0.9773963290278 722	0.9838545207341 944
Ru n1 0	19	21	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21	0.15	3	0.9768864717878 993	0.9850441876274 643
	20	21	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21	0.3	10	<u>0.9780761386811</u> 693	0.9843643779741 672
Ru n1 1	21	22	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22	0.15	3	0.9765465669612 509	0.9847042828008 158
	22	22	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22	0.3	10	0.9758667573079 538	0.9836845683208 701
Ru n1 2	23	23	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23	0.15	3	<u>0.9796057104010</u> 877	0.9853840924541 128
	24	23	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23	0.3	10	0.9758667573079 538	0.9833446634942 217
Ru n1 3	25	24	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24	0.15	3	0.9792658055744 392	0.9850441876274 643
	26	24	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24	0.3	10	0.9768864717878 993	0.9838545207341 944
Ru n1 4	27	25	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25	0.15	3	0.9785859959211 42	0.9853840924541 128
	28	25	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25	0.3	10	0.9756968048946 295	<u>0.9845343303874</u> 915

Table 3.4: All the runs and accuracies obtained starting with ten features “Table Continued.”

Run 15	29	26	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26	0.15	3	0.9779061862678 45	0.98504418762746 43
	30	26	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26	0.3	10	0.9765465669612 509	0.98198504418762 74
Run 16	31	27	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27	0.15	3	0.9755268524813 052	<u>0.98742352141400</u> <u>4</u>
	32	27	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27	0.3	10	0.9777362338545 207	0.98232494901427 6
Run 17	33	28	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28	0.15	3	0.9772263766145 479	0.98470428280081 58
	34	28	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28	0.3	10	<u>0.9780761386811</u> <u>693</u>	0.98402447314751 87
Run 18	35	29	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29	0.15	3	0.9779061862678 45	0.98538409245411 28
	36	29	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29	0.3	10	0.9779061862678 45	0.98266485384092 45
Run 11	37	11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	0.15	3	0.9687287559483 345	0.98334466349422 17
	38	11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	0.3	10	0.9716179469748 47	0.97790618626784 5
Run 12	39	12	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	0.15	3	0.9694085656016 316	0.97926580557443 92
	40	12	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	0.3	10	0.9704282800815 771	0.97603670972127 81

Twenty pairs of runs were performed to finish the run set in order to cover adding all the features and observing the impact of the added feature, in total 40 runs for NN_MLPC and

another 40 for Decision Tree. At the beginning adding feature number 11 and adding feature number 12 separately were not recorded, the start was with ten features then jumping to 13 features, but after finishing, Run 1_11 and Run 1_12 were added at the end of the table to record the effect of adding those features (11 and 12). Note that the .csv file corresponding to each run has the same name of the Run, for example, Run 1 correspond to the .csv file named “Malware_Hassan_Features_Run1”, Run 2 corresponds to “Malware_Hassan_Features_Run2” and Run 1_11 corresponds to .CSV file “Malware_Hassan_Features_Run1_11” and so on.

Table 3.5 which includes the runs with three features till nine features, Run 21 through Run 27, in this table a complementary Runs to Table 3.4, since in these runs, the start was with three features (Run 21) and ended with nine features (Run 27), the features added has the same sequence number in Table 3-4. At the beginning it was not decided to make these runs, since the number of features were considered relatively low, after implementing all the runs in Table 3-4 and since this study looked at how the number and characteristics of a feature affect accuracy, it was important to complete the runs, but this time the least number of features was decided to be not less than three since this will be considered irrational. So, the three features chosen to be the starting point are the most common features as mentioned above which were DllCharacteristics, MajorImageVersion, and MajorOperatingSystemVersion.

Table 3.5: The Added runs and Their Accuracies.

RUN No.	No. of Sub-Run	No. of Features Used	Features Used. Table-3-	Test_size	Random_State	Accuracy	
						NN_MLPC	Decision Tree
Ru n21	41	3	1, 2, 3	0.15	3	0.8871515975526852	0.959551325628824
	42	3	1, 2, 3	0.3	10	0.8864717878993882	0.9578518014955812
Ru n22	43	4	1, 2, 3, 4	0.15	3	0.946974847042828	0.9585316111488783
	44	4	1, 2, 3, 4	0.3	10	0.9525832766825288	0.9609109449354181
Ru n23	45	5	1, 2, 3, 4, 5	0.15	3	0.9333786539768865	0.9714479945615228
	46	5	1, 2, 3, 4, 5	0.3	10	0.9350781781101292	0.9656696125084976
Ru n24	47	6	1, 2, 3, 4, 5, 6	0.15	3	0.9490142760027193	0.9765465669612509
	48	6	1, 2, 3, 4, 5, 6	0.3	10	0.9398368456832087	0.9755268524813052
Ru n25	49	7	1, 2, 3, 4, 5, 6, 7	0.15	3	0.969068660774983	0.978585995921142
	50	7	1, 2, 3, 4, 5, 6, 7	0.3	10	0.9486743711760707	0.9753569000679809
Ru n26	51	8	1, 2, 3, 4, 5, 6, 7, 8	0.15	3	<u>0.9619306594153637</u>	<u>0.981645139360979</u>
	52	8	1, 2, 3, 4, 5, 6, 7, 8	0.3	10	0.9456152277362339	0.9746770904146839
Ru n27	53	9	1, 2, 3, 4, 5, 6, 7, 8, 9	0.15	3	0.9656696125084976	0.9796057104010877
	54	9	1, 2, 3, 4, 5, 6, 7, 8, 9	0.3	10	0.951393609789259	0.9758667573079538

3.2.6 Analysis And Comparison of Results

In this subsection, a detailed and comprehensive analysis to the results obtained. The analysis process included the use of the graph, since the graph will give a better idea and vision on the results obtained.

It was important to use the same scale in drawing all the figures, NN_MLPC and DT figures. For both algorithms, the accuracies were rearranged starting from the lowest accuracy till the highest accuracy in a new separate sheet, an independent sheet for each pair of test_size and Random_state for each algorithm. A total of four independent sheets were created. The

scale started with 0.88 which was the lowest accuracy range or level (0.88x) till 1.0 since the highest accuracy was in the range of 0.98xx.

3.2.6.1 NN_MLPC

Figures 3-3 and 3-4 presents the graphical form of the obtained results in NN_0.3_10 and NN_0.15_3 respectively. The accuracy is plotted against the total number of features used in each run in these figures.

Results obtained from including three features only were low results comparing with other high level obtained results, especially in NN_MLPC, while in DT a better level of accuracy were obtained using three features only. Also, including relatively low number of features like four, five, six till nine or even ten features brought a better level of accuracy but not in the range of high accuracies or levels that have produced the highest accuracy. While in DT, the view is different, since higher accuracies were obtained till 0.97x or the 0.98x levels using the four till nine features.

It is notable that relatively low level of accuracy was obtained if three, four till ten features were included in the run. While employing eleven and more features made the accuracy to be increased to 0.97x range which is the level of maximum accuracy in NN_MLPC, for instance, an accuracy of 0.97x was achieved using eleven features, test_size = 0.3, and Random_state = 10. Conversely, at test_size = 0.15 and Random_state = 3, the accuracy did not approach the 0.97x threshold until thirteen (13) features were included. Using Eleven or Twelve features produced a lower level of accuracy, see the end of Table 3-4 for the eleven and twelve features and the beginning of Table 3-4 for the thirteen or above features. Note that, at this point the importance of doing all the runs with all the number of features appeared, because the eleven and twelve features' runs were not done at the beginning after doing the ten features run, they were done after finishing the twenty-nine features run, and before starting again with the runs of Table 3-5.

Now, returning to the maximum accuracy obtained using the NN_MLPC algorithm. This accuracy has been obtained with NN_0.15_3 which was 0.979606 using Twenty-three (23) features. In general, after including Fifteen (15) features in the runs of NN_MLPC, a more stable graph has been obtained, see Figures 3-3 and 3-4, of course some fluctuation can be noticed within the same range of 0.97x accuracy which is the range of maximum accuracy.

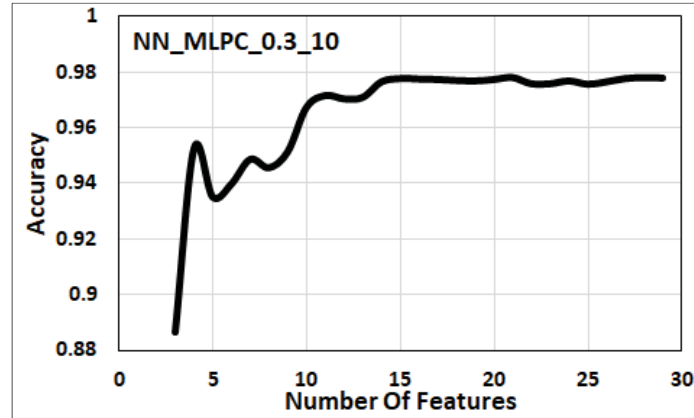


Figure 3.3: NN_0.3_10 Results.

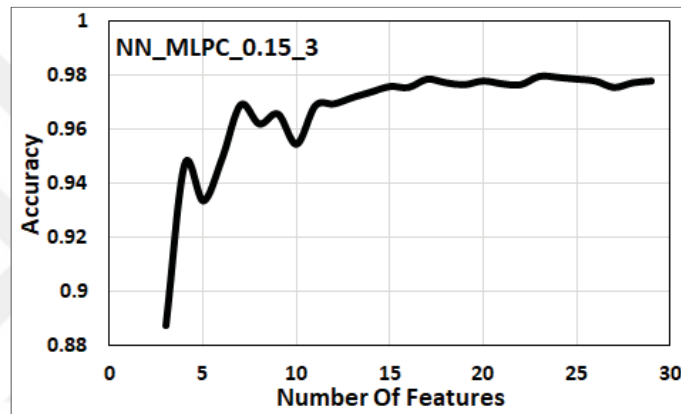


Figure 3.4: NN_0.15_3 Results.

3.2.6.2 DT

Figures 3-5 and 3-6 presents the graphical form of the obtained results in NN_0.3_10 and NN_0.15_3 respectively. The accuracy is plotted against the total number of features used in each run in these figures.

Till four features, an accuracy between 0.95x and 0.96x were obtained, after that between five features and ten features an accuracy level of 0.97x were obtained except in the run 26 sub-run 51 particularly in DT_0.15_3 the obtained accuracy was 0.981645139360979, note that this run used only eight features. Also, this level of accuracy which is 0.98x is the same level of maximum accuracy in Decision Tree. It is significant to note that the 0.97x accuracy level obtained using the number of features between six and ten is the level of highest accuracy obtained in NN_MLPC using number of features of eleven and more and this level was not obtained with number of features less than eleven in NN_MLPC.

Starting from DT_0.3_10 an accuracy of 0.97x was obtained and as mentioned above the first 0.98x accuracy was obtained using eight features but in DT_0.15_3, add to that, starting from employing fourteen features the 0.98x accuracy was appearing continuously.

In a conclusion to what is mentioned regarding DT, using six features in DT_0.3_10 an accuracy of 0.97x has been obtained, while the accuracy of 0.98x has been obtained using fourteen features onwards. On the other hand, for DT_0.15_3 an accuracy of 0.97x can be obtained by using five features only and above, and 0.98x has been achieved with eight features only then ten then thirteen and after that continuously.

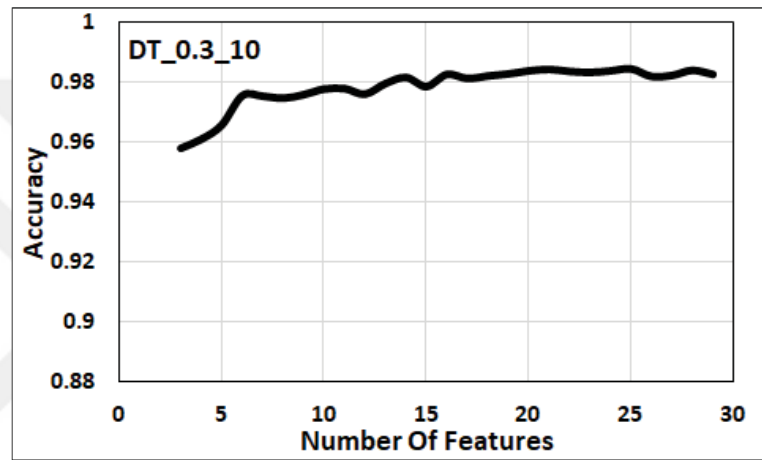


Figure 3.5: DT_0.3_10 Results.

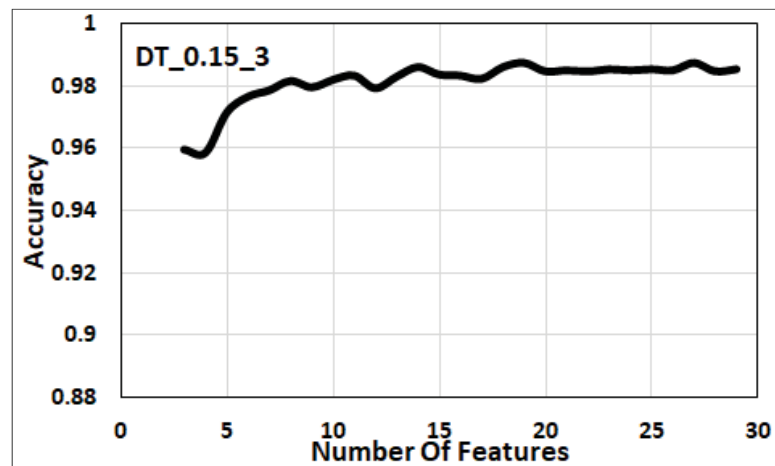


Figure 3.6: DT_0.15_3 Results.

3.2.7 Observations

The start will be with Table 3.6 which will give the analysis summary. The names of the columns will be as follow:

- a. Run Type.
- b. Highest accuracy or maximum accuracy obtained.
- c. How many features are needed to get the best accuracy?
- d. Features that have been used when highest accuracy obtained. Feature's list. This column will use the format 1-x, this means that the features starting from feature 1 in table 3.2 till the feature x have been used, e.g., 1-5 means features 1, 2, 3, 4, 5 have been used.
- e. The lowest or the minimum accuracy obtained.
- f. The number of features used when minimum accuracy has been obtained.
- g. The features that have been used when highest accuracy obtained. Feature's list. This column will use the format 1-x, this means that the features starting from feature 1 in table 3.2 till the feature x have been used, e.g., 1-5 means features 1, 2, 3, 4, 5 have been used.

Table 3.6: Summary of Analysis: Maximum and Minimum Accuracy Attained

1	2	3	4	5	6	7
NN_0.3_10	0.97808	21	1 - 21	0.88647	3	1 - 21
		28	1 - 28			
NN_0.15_3	0.97961	23	1 - 23	0.88715	3	1 - 3
DT_0.3_10	0.98453	25	1 - 25	0.95785	3	1 - 3
DT_0.15_3	0.98742	19	1 - 19	0.95853	4	1 - 4
		27	1 - 27			

Some observation points can be noticed from Table 3.6:

- a. Decision Tree algorithm produced the highest accuracy.
- b. Test_size = 0.15 and Random_state = 3 were the pair that yielded the highest accuracy between test_size and Random_state.
- c. The maximum accuracy was achieved by utilizing twenty-seven features.

Table 3.3 and 3.6 can give some observation points like:

- a. A reduced feature count will lead to a lower accuracy. This could imply that increasing the number of features can increase accuracy. For instance, the maximum accuracy obtained in NN_MLPC was in the range of 0.97x, using NN_0.3_10 the 0.97x accuracy was obtained using eleven features or more, while in NN_0.15_3 the 0.97x accuracy was obtained using thirteen features, any number of features less than Eleven in NN_0.3_10, and less than 13 in NN_0.15_3 will result in accuracy of less than 0.97x accuracy.
- b. According to point (1) mentioned above, the phrase "low feature count or number" indicates that there are fewer features than that can produce the range of highest accuracy which means 0.97x in NN_MLPC and 0.98x in DT. Hence, in DT less than fourteen features in DT_0.3_10 and less than eight features in DT_0.15_3 will be considered as low number of features since they produce an accuracy of less than 0.98x.
- c. It is notable that each algorithm has its own range of highest accuracy, the maximum accuracy in NN_MLPC is different than the Maximum accuracy in DT, also, the quantity of features required for each algorithm in order to achieve the highest level of accuracy.
- d. Every algorithm has a specific number of features that produce the range of maximum accuracy, so obtaining the range of highest accuracy can be done with the fewest features possible, this does not imply that we should stop here; in fact, features addition more could improve the accuracy even more. The graphs showed that the accuracy reached its maximum value when more features were added, In DT_0.15_3,

for instance, eight features yielded an accuracy of 0.98x, whereas the highest accuracy possible was achieved with Nineteen or Twenty-Seven features, returning to the Tables 3-4 and 3-5 same observation can be notices in NN_MLPC.

It is evident from the aforementioned points that different kinds of algorithms will use different numbers of features in order to achieve the accuracy upper limit or the range of the upper limit accuracy. Additionally, every algorithm will be impacted differently from the others in terms of the quantity of features required to achieve each milestone or even the ranges that each quantity of features will yield. This was the primary motivation behind the inclusion of two distinct algorithms in this work, which allowed us to examine both the algorithms' behavior and how the features affected accuracy.

Estimating the number of features and their impact on obtained accuracy was the work primary goal. Because of that the progressive addition of the features was an important issue regarding the observations of the produced accuracy.

The results obtained show that accuracy tends to increase as the number of features increases, beginning with a small number of features. Add to that, it appears that each feature has an impact on accuracy different from the impact of other features, this point was clear from the graphs of each type.

Adding some specific features will lead to a positive impact on accuracy which is different than other features, on the other hand, adding some features will lead to a small negative effect in accuracy. As an example, when sixteen features were included in NN_0.3_10, the features are 1 to 16, the result was an accuracy of 0.9776, but when the feature 17 (Table 3.2) was added the accuracy was 0.9774. It is correct that the change is small value, but this change pours into the interest of the study. Another example was when twenty-one features have been used in DT_0.15_3, an accuracy of 0.985 was obtained, but when the feature number 22 (Table 3.2) was added, the accuracy decreased to 0.984, this topic requires more thorough investigation.

Additional signals suggested that accuracy would plateau after adding a certain number of features, or that this quantity of features would be sufficient to achieve a satisfactory degree of accuracy. As an example, ten features obtained an accuracy of 0.954, while thirteen features obtained 0.971 in NN_0.15_3. This point was outside the purview of this part of the

stage one work and requires additional research and work to be proven. But all the observations and outcomes from this work have been more analyzed in part two of stage one which included an in-depth analysis and testing to determine how the features affected the accuracy of malware detection.

During preparations of the work, confusion matrix was taken into consideration, for example, in NN_MLPC algorithm the readings showed that TN=47071, FP=1241, FN=839, and TP=19571 were confusion matrix values, note that taken samples were 68722 samples. Actually, this point was taken into consideration sometimes since it can help in taking the correct decision of to what extent we can depend on the algorithm.

Another important issue that was taken, is the comparison with previous work. In this part, the maximum accuracy measured in DT was 0.987 and in NN_MLPC was 0.979. Different strategies have been exploited in previous implementations. The implementations that have a similarity with this work obtained the following:

- a. In the work of [44], accuracies of 0.98 and 0.97 have been obtained, no additional fractions have been mentioned. In this work the accuracies obtained were 0.987x and 0.979x. Being aware that NN_MLPC was not included in the implemented algorithms in [47], which included DT in addition to other algorithms, this part implemented both NN_MLPC and DT.
- b. People in [37] obtained an accuracy of 0.9826 using the DT algorithm. In this work DT has attained 0.9874 accuracy. The implementation in [37] is different than the presented work in that forward selection method has been implemented after feature extraction, then classification was employed to discriminate between the malware and benign files.

There is a final noteworthy point which can be mentioned, that high accuracy levels i.e., the levels of maximum accuracy, was attained early using certain number of features, like the 0.97x using eleven features in NN_MLPC and the 0.98x using the eight features in DT. The accuracy will benefit from and be improved by the gradual addition of features, but this improvement will not last forever, as an example, accuracy of 0.972x was obtained in NN_0.3_10 using eleven features. When Twenty-One features have been reached, maximum accuracy of 0.978x was obtained, and this same maximum accuracy appeared again with

Twenty-Eight features. This indicates that the feature addition enhances the accuracy to some value on the same range which considered as the highest and the accuracy will not exceed this value even if additional features were added. The same idea applied to DT algorithm. Finally, the observations and findings in addition to the various implementations gave an importance to do more deeper studies regarding malware detection using portable executable file headers.

3.2.7 Part one's Conclusions

A study on the effect of adding portable executable header features so that a better accuracy in malware detection can be obtained, has been done in this part.

Python programming language was used to implement the work, after that algorithm testing and selection is done, two algorithms have been selected, NN_MLPC and DT. Good accuracy levels have been attained from both algorithms, but the DT produced a better accuracy level.

The increase in features number will result in enhancing accuracy level, according to the results. Furthermore, indications showed that accuracy does not always increase with feature count. There have occasionally been noticeable instances where an increase in feature numbers results in a decrease in accuracy.

To attain the high accuracy level, each algorithm needed a different quantity of features. In both algorithms, NNMLPC and DT, required higher number of features to reach the highest accuracy.

Incorporating the graphs in presenting the results was useful, it gave an advantage in observing the accuracy variation resulted from adding more features.

3.3 PART TWO

This section represents the candidate's second publication [36]; this work (part two) includes a detailed study which can be considered as both a creative and complementary study motivated by the results and findings obtained in part one [35]. In this work features subgrouping method was implemented, this implementation aimed to answer some questions. Which feature, when added, can have a positive impact on the accuracy value? Is it feasible to identify a set of characteristics that reliably differentiates between malicious

and benign files? Does the accuracy of malware detection change when the features are combined?

Obtained results were promising and improved, also, the results included some observations. The outcomes' primary features were their practical and numerical components. The practical finding part included that which features included and the number of the features considered as the main factors that impacting the accuracy calculated. Add to that, features combination can be considered as a critical issue in the calculations. The numerical part includes the improved accuracies obtained, such as in NN_MLPC accuracies of 0.979x and 0.98x were produced; while in DT the obtained accuracies were 0.9825 and 0.986. the attained accuracies were improved and outperformed what was attained in part one which is [35].

3.3.1 Part two Introduction

Several researchers have employed the features of the portable executable header files to build a strategy that can find the malware that may reside in such files. It has been demonstrated that some or most of the features can be used to distinguish between benign and malicious files.

Several methodologies were employed to examine and identify the features that could impact the accuracy of malware identification. Some observations were recognized in previous implementations, those observations need to be studied in details and being proved, the degree of accuracy may be impacted by factors like the quantity of features used, feature set combinations, and particular features included in the set. Those observations have been studied in this part.

It was assumed in this work that all the features selected to be used possess the positive influence on the accuracy. This assumption came from the facts and findings observed previously. The process of feature selection was not based on random feature set collection.

It was also assumed that accuracy could be increased by using fewer features, by using specific features, and that the combination of features could have a positive or negative impact on accuracy. It is imperative to reiterate that none of the assumptions were arbitrary or derived from the candidate's wild speculation.

Malware is described as intentionally introducing, deleting, or altering code into software in order to cause harm and change how the software functions. Although there are numerous studies on malware detection have been conducted, some considerable limitations can exist in the technology of malware detection [39].

Various techniques have been employed via incorporating modern detectors for malware [41]. The detectors first reveal the file type, then items embedded in the file were discovered and/or parsing the file after extracting its content. The anti-virus scanner will have to parse a diversity of file format, this will lead to complicate the anti-virus scanner. However, in some certain cases, with the incorporation of a complicated anti-virus programs some simple malware can penetrate the program.

Both the malware and the benign files' structural and behavioral characteristics are used when developing a machine learning malware classifier model. Model will decide whether the sample is infected or not [44]. Various features of the portable executable header have contributed when used in the process of discriminating the good files from the infected files. People in [44] had observed that the values of the features NumberOfSections, NumberOfSymbol and SizeOfInitializedData controls the larger differences to malware and good files, this was stated by [44].

Portable executable is the format in which executable files were originally used by the Windows operating system. The portable executable specifications were originally derived from the UNIX COFF, the format structure of the data which represent the portable executable encapsulates the important information. The loader of the Windows operating system uses those encapsulated information, hence the executable code management can be carried out [35]. Resistance to code modifications was not from the characteristics of the portable executable when designed, this drawback made the process of injecting malware easy. Naturally, if the injected code is malicious, then the portable executable file will be infected. At this point and according to the type of the injected code, threats like Trojans, ransomware and worms will start to do their job. The problem of such threats is that they will run after their injection without any signs or indications given to the user.

The work of [49], stated that computer networks suffer from the threatening of various Cyber-attacks, different tools like Deep-Pack Inspection (DPI), Encryption and Intrusion

Detection tools are utilized by the administrators of the networks to guard their systems from those threats. Recently, security can be further improved using machine learning techniques.

Differences between the traditional signature-based technique and machine-learning techniques in malware detection were discussed by several previous implementations. People in [50] and [51] stated that fresh and new malware will not be recognized by the static signature-based technique, while already recognized and known malware will be detected easily.

Dynamic technique used by many Anti-malware programs, this technique considered as a second malware detection technique, its foundation lies in using virtual environments and running the executable within them, the primary disadvantage of this method is that the scanning process takes long duration and this technique considered as a high resource consumer.

Heuristic approach that has demonstrated its value in a number of fields, is considered as a third technique which incorporates machine learning [50], [51]. It is important to compare between machine-learning technique and signature-based technique, people in [52] and [23] concluded that the machine-learning based technique can incorporate the suitable parameters for the model used in detection, also, malware and benign samples can be deduced.

People in [52] stated that introducing machine learning made the process of malware detection an active process and will be no longer passive.

The work of [53] mentioned that the attack of zero-day cannot be faced using signature-based techniques, and the networks and computers are vulnerable to be damaged by an attack if the signature-based technique did not get its update in the suitable time. Another problem stated by [54] with signature-based is that the virus can bypass the operation of examination using obfuscation.

Hybrid technique has been implemented by [55], this technique includes signature-based and machine learning, The goal was to defend Internet of Things (IoT) systems from home WiFi assaults that were directed at the gadgets. This system succeeded to protect the system against five big attacks included in a list of the top ten risks during 2018.

Executable packer tool can be used to protect the software, this was stated by [54], originally, this tool was used to conserve the information from reverse engineering. Unfortunately, this tool was exploited to be a tool for obfuscation, this was done by changing the shape of the malware in order to confuse the detection mechanism. Packed malware constituted 80% of malware, not only to confuse the detection system, but also to make the compact inside the malware propagate.

Exploiting the image page of an .exe file to hide information is presented by [56], in the manner of this system, the .exe will not be detected as being a virus, add to that the .exe will continue to work normally, knowing that the functionality of the executable file may be changed if any data is hidden inside it.

MS-Windows operating system has made a portable executable file format standardization [41]. This standardization included executable files, object files and for dynamically linked libraries (DLL). The work proposed a PE-Miner real-time and accurate framework. Distinguishing features that are inside the portable executable can be extracted automatically so that malware can be inspected. The obtained result was a detection rate more than 99%, but the time consumed so that the whole content of the portable executable can be scanned, was more than one hour.

The work of [35] used two distinct classification algorithms: Neural Network Multi-Layer Perceptron (NN_MLPC) and Decision Tree (DT), where twenty-nine portable executable header features were collected after being recommended from previous implementations, and then those features have been studied to be the contrivance with which malware can be recognized, also, to estimate the effect of various combinations from those features on accuracy.

Based on observations and results from earlier implementations, particularly [35], which demonstrated that characteristics of the portable executable header file can influence malware detection accuracy in a positive or negative way; Numerous factors that could impact the accuracy of malware detection have been researched, including the features that are used, the quantity of features that can yield the highest accuracy, Are the features different in terms of how they affect accuracy? That is, does a particular feature or combination of features have a greater impact on the accuracy value? Is it feasible to identify

a set of features that is optimal and can be used consistently to differentiate between malicious and benign files? A unique approach must be used to choose the features and consider them for each run, in order to answer all of these statements and questions. The process of achieving the intended goals of this work was largely aided by the technique of subgrouping the features based on the observations and findings made earlier, as well as by classifying the accuracy by establishing criteria based on defining the high, good, and low accuracies.

3.3.2 Research Literature Review

Misuse of the portable executable file will lead to hazardous problems, so, it is important to take caution when dealing with them, or try to detect the malicious aspects inside them, this was the main reason which made several implementations study how to avoid the misuse of this kind of files by malicious people, through detecting the files with contain the malicious code inside them.

Nine features only were employed by [23] to get an accuracy of 95.59%. The used algorithms were SVM, Random Forest and kNN, the advantage of this model was that it has a faster training and the overall model's speed was high.

Using the tool named VirusTotal, people of [57] were able to achieve an accuracy of 97%. Additionally, by regularly updating their model, the main goal was to decrease the number of attributes needed to detect malware. Thus, instead of using thousands of attributes they incorporated a single model. This will be useful in reducing the overhead of the processing as they stated, reducing the attributes was represented by decreasing the number of the used files in the process of extracting those abundant number of attributes.

Three steps were done in the work of [41], the steps are summarized as: (1) large dataset was collected from two different websites, this dataset define the malware and legitimate exe files. (2) a comparison was made between the malware and legitimate files, following the removal of every feature from each file's header and (3) After being extracted from the PE file, the most popular tokens were identified. The aim of doing the three mentioned steps was to discriminate benign files from the malicious files using the structural information.

The work of [42] illustrated the important role of the portable executable header features, they stated that the process of packing the file will make it difficult to be recognized as

malware or good file, hence their implementation was to incorporate the header features in detecting the packed exe files. Their study took only the packed exe files, after that feature were extracted, then analyzed so that a decision can be made on whether this binary file is packed or not.

Less than two-hundred features were used in [39], the work proposed the information mining of the features, after extracting the features, a learning process has been carried out using classification technique. The obtained accuracy was 99.1% and the percentage of detecting new malware was 97.6%.

Malware can evade from detection if numerous obfuscation techniques have been employed, this is what [38] stated in their paper. They mentioned also, that malware detection approaches used like the signature-based scanning will be ineffective. Fuzzy classification algorithms have been studied to be implemented on static heuristic features, this was the technique proposed to detect packed files and malware.

The work of [43] confirmed that selecting then extracting features will play a good role in reaching the right decisions regarding malware classification and the attacks by intrusion. The constructed system was able to prevents, understand and indicate the connections that could be malicious.

High accuracy and low overhead in computation were the main targets of [44] who employed machine learning to classify the files to be either benign or malware. DT, kNN, Random Forest, Logistic Regression, Linear Discriminator Analysis and Naïve Bayes were hired to work on several PE header fields. Sets of integrated features were merged to create a derived and raw values. The obtained accuracy was 98.4% while the cross-validation was ten. Note that employing only the top fifteen features, an accuracy of 97% to 98% was obtained.

The people in [45] suggested the extraction of portion of the portable executable header by using the domain knowledge with minimal amount. While people in [46] suggested the use of both Artificial Immune System (AIS) together with Deep learning to determine whether the file is malicious or benign, the main aim was to achieve a high rate of accuracy. The role of AIS was to build a generation of malware detectors in the second stage while feature extraction to construct a feature set was done in the first stage. Obtained accuracy on the unknown virus was 99.4%.

The prior implementations have demonstrated that the feature extraction process is the initial stage, and this is also the case with [37], where after the feature extraction process, a method of forward selection was employed targeting the malware detection. After that file classification can decide whether the file is benign or malware. Highest accuracy obtained was 98.26% using Decision Tree (DT) algorithm.

The candidate works of [35] has collected twenty-nine features, those features were recommended from previous implementations, NN_MLPC and Decision Tree (DT) were used in the work, given that other classification and clustering techniques were tested. A feature was added at a time before the run and recording the obtained accuracy. The obtained results can be presented as follow: using twenty-one or twenty-eight features, the algorithm was NN_MLPC, and the accuracy was 0.9781 when test_size was 0.3 and Random_state was 10. Once more, NN_MLPC was used with twenty-three features set, test_size = 0.15, Random_state = 3, and the accuracy was 0.9796. The accuracy was 0.9845, however, when DT was applied with test_size=0.3, Random_state=10, and a feature set made up of twenty-five features. In contrast, DT's test_size of 0.15 and random_state of 27 features resulted in an accuracy of 0.9874.

Classification and detection of malware using the similarity of n-grams attribute has been used by [58]. In addition to the proposed technique, SVM, Naïve Bayes, C4.5 DT and Bayesian Networks have been used. People in [52] used the Deep Q-learning Network (DQN), this was done through feature analysis by neural networks, after that the Q-Value corresponded to the action space was extracted. A decision strategy action will be selected by incorporating Q-Learning and depending on the obtained Q-Value. To this point the malware detection is completed.

Another technique is proposed by [59], which is the fuzzy hashing, in which file similarities were detected by testing specific hashes. In addition to test some techniques combinations in order to improve detection rates using the method of hashing. Imphash, Ssdeep and PeHash hashing techniques were used in this work. The focus in the work was on the techniques that figures out the portable executable file similarities.

The technique of PEFAF which is the Portable Executable File Analysis Framework was used by [60], on the other hand the incorporation of static analysis was implemented by [19],

the static analysis technique employed seven algorithms so that malware can be classified, the work also used metrics like accuracy, F-measure and precision which will be used later in another implementation of the candidate. Obtained results favored the integrated features on the raw features on all metrics used. Integrated features obtained accuracies between 91% and 99%, while 71% to 97% were the range of results obtained by the raw features. 70/30 split ratio used and Random Forest attained 99.23% accuracy.

Six group tree-based machine learning systems are used. the Gradient Boosting Classifier (GBC), the Ada-Boost Classifier (ABC), the Random Forest Classifier (RFC), the Extra Trees Classifier (ETC), the Bagging Decision Tree Classifier (BDT), and the Voting Ensemble Classifier (VEC), has been also tested by [61], an accuracy of 95% was obtained, calculated and raw metadata has been extracted from the file which is portable executable, the used features were fifty-four features.

Classification the metamorphic family of virus variants was done by [62] using the opcodes histogram of the instructions. On the other hand, a classification algorithm which is the Support vector machine (SVM) was employed in [53], feature selection and the process of training the classifier were done simultaneously so that the performance of the classifier will be cost-effective and number of attributes will be reduced.

Packing has occupied an important space in research of the people working in the field of malware detection in portable executable files. For example, researchers of [63] have worked on improving the process of packing detection, where they have confirmed that numerous properties of the portable executable file will be modified because of packing. While people in [54] have proposed the packing detection framework (PDF), the work stated that attributes which are valued ones were extracted first, after that a training process is carried out using those extracted attributes, the process will train a two-class SVM classifier so that the executable can be recognized to be packed or not.

Building an intrusion detection system which is based on anomaly was studied by [5], the work emphasizes how various machine-learning techniques have been applied to the KDD'99 dataset on a large scale.

A Logistic Regression, XGBoost, voting classifier and LightGBM have been employed as an algorithm for hybrid machine learning in [64]. The RECALL metric was used to measure the performance and attained 99.5%.

Auto-encoder after implementing static analysis to portable executable files were the main steps that [65] has applied in order to detect the codes of malware. Five-hundred-forty-nine features (594) were incorporated in this research, all those features contain information related to DLL/APL.

Pairing LightGBM with Long Short-Term Memory (LSTM) [66] the pairing process aimed to classify malware using portable executable. Sophos-Reversing Labs 20 million dataset was used in the training process, then testing has been carried out using the same dataset. Obtained result was 91.73%. Another work has employed the LSTM in detecting Ransomware, the work is [67], taking the portable executable file header as a reference, the work stated that the portable executable file header can include important information, the information particularly related to the structure of the program, which indicates that by altering the order in which the bytes that make up the header information are sequenced, the program structure can be changed. The procedure involved extracting the ransomware-infected samples and using the LSTM network to process the series of bytes that made up the header. Using this method yielded an accuracy of 93.25%.

The work of [68] took both point of views of the defender and attacker. The 2018 FFRI dataset was the reference of the surface information of the portable executable so that the model can be evaluated. The suggested work made use of machine learning and dimension reduction. The study suggested an adversarial attack called the IM-AGE_RESOURCE attack in addition to a malware detection model.

This time Deep learning model has been incorporated by [69] so that malicious code can be detected, the GPT-2 “the Generative Pre-Trained Transformer based” and the Stacked BiLSTM that is “Stacked Bidirectional Long Short-Term Memory” were primary proposals of this work. It is known that assembly instructions are contained in .text sections of both benign and malicious portable executable file. The work treated the assembly instructions as sentence, and treating each .text section as document. Every sentence and document were classified as either benign or malevolent based on the file source. Then creating three

datasets, the first is document, this dataset is fed to the DLAM which is the Document Level Analysis Model, second dataset that SLAMs use is made up of sentences. which is the Sentence Level Analysis Models, this was established on GLM-GPT2 (General Language Model GPT-2), DistilBERT and Stacked BiLSTM also, DSLMGPT2 (Domain Specific Language Model GPT-2). The third dataset, which was created by combining all assembly instructions without any labels, will be used to train a specially designed model beforehand. After that the performance of detecting malware is compared. The metric use to measure the performance was F1_score.

Finally, looking to the work of [70], in which accuracy of classification was investigated, this was accomplished using LightGBM and the custom log loss function. was the foundation for the work. The false negative side of the loss function was equipped with the coefficient of α , on the other hand the β coefficient was installed to the loss function false positive, this was done to control the process of learning through the LightGBM. The installation of the α and β coefficients created a lopsided classifier. Non-public and public malware datasets were employed. The header of portable executable (PEheader) and analyzing of PE-file surface were the main sources that the dataset features were obtained from.

3.3.3 Header Features of Portable Executable

The executable form and layout section are defined by the portable executable. The portable executable is incorporated in every .sys, .exe and .dll files. A binary form data structure is the shape of that the portable executable looks like in Windows.

The Windows operating system's loader uses the description's information contained in the portable executables. In addition, the code of the program is hosted in the portable executable [48].

Some description to the portable executable file has been presented in chapter two; Here, it's crucial to emphasize that the file header in a portable executable file is what's being meant, both the DOS header and the optional header have a collection of structures in them. According to earlier implementations, certain structures can affect accuracy; features are the forms that these structures are used on. The features that this work's experiments used were gathered from earlier implementations.

Along with the importance of the features (structures) in the portable executable file there is a drawback, that some of these features can host a malware which will represents a threat to the operating system. Because of that many implementations have studied the features of the portable executable file header, also, various strategies were used so that the most effective and sensitive features can be extracted. Each study has nominated a set or more of features to study, the special thing about that is that some features like DllCharacteristics for example were a common suggestion that more than one study suggested them.

AddressOfEntryPoint, MajorImageVersion and MajorLinkerVersion are examples of the features that [13] has used in the experiments, where eight features in total were incorporated. While [48] mentioned that in previous implementations PE-Header-Parser were employed to analyze part of the portable executable header, the outcomes of the work indicated that with an accuracy of 90% the file will be malicious if the features MajorImageVersion, checksum and DLLCharacteristics have the value of zero, another indication that the file is malicious is that the SizeOfInitializedData feature equals to zero.

Twenty-five features were listed by [46], the people of [46] believe that a better accuracy can be obtained if those twenty-five features were incorporated in the process of malware detection. SizeOfUninitializedData and SizeOfHeaders were examples of the raw features used in [44], where raw and binary features combination were utilized in the work so that a better accuracy can be obtained.

Five features were suggested by [41] to be the features that contain the considerable differences between malicious executables and benign ones. The five features are Checksum, MajorImageVersion, DLLCharacteristics, SizeOfInitializedData and UnknownSectionName.

A wide collection of features has been incorporated in this work, majority of those features were collected from previous implementations, Table 3-1 lists the six excluded features. While Table 3-7 list the twenty-nine features incorporated in this work. It is possible to refer to Table 3-2 for more details about the references of the features knowing that Table 3-7 and 3-2 has similar feature order. Please take note that while illustrating the work experiments and outcomes, the feature number will be used in place of the feature name in this work.

It is important to emphasize that this work is part two of stage one and it represents an independent work which complements the work of part one [35]. As previously stated, the indications and conclusions from [35] have mandated that more work be done to address a few crucial questions about the quantity and combination of features used, As mentioned earlier, further research is required to address a few important questions regarding the number and combination of features used. This is because of the indications and conclusions from [35], in addition to other findings, this was important, since the obtained results from this work were enhanced and achieved an accuracy level better than what was achieved in [35]. Also, all the questions obtained a reasonable answer. Accordingly, the feature set used in this work, the dataset and the algorithms were the same to prove what was needed to be proved regarding the findings of the work in [35].

Table 3.7: Features and Their Numbers.

Number	Feature name	Number	Feature name
1	DllCharacteristics	16	SubSystem
2	MajorImageVersion	17	MinorImageVersion
3	MajorOperatingSystemVersion	18	SizeOfStackCommit
4	SizeOfStackReserve	19	e_Ifanew
5	AddressOfEntryPoint	20	e_minalloc
6	Characteristics	21	e_ovno
7	SizeOfHeaders	22	Machine
8	SizeOfInitializedData	23	PointerToSymbolTable
9	SizeOfUninitializedData	24	NumberOfSymbols
10	MajorSubsystemVersion	25	Magic
11	MinorSubsystemVersion	26	SizeOfCode
12	Checksum	27	BaseOfCode
13	ImageBase	28	SectionAlignment
14	MajorLinkerVersion	29	FileAlignment
15	NumberOfSections		

3.3.4 Algorithms

Different algorithms have been tested in order to take the correct decision regarding which algorithm can be the most suitable algorithm for this implementation, clustering and classification algorithms were included in the process of testing. K-Mean was one of the clustering algorithms tested, in addition to Naïve Bayes, Random Forest and SVM which were a classification algorithm. Some recommendations stated that classification can better deal with the problem of malware detection [13]. Eventually, the choice fell on two classification algorithms, NN_MLPC and DT.

Selecting those two algorithms came for several reasons, the most important was the results were enhanced, in addition to the reasons related to each algorithm. The NN_MLPC can simulate the human intelligence [51], this will result in constructing a classifier which can recognize malware which resides in the portable executable file in a way that might be better than others. The shapes of the values that will be taken into account when creating the predictions for the decision tree are numerical and categorical forms [13]. The primary reasons that led to select the Decision Tree algorithm is that the DT has produced the highest levels of accuracy between all the tested algorithms including NN_MLPC add to that in the preparation stage for the work of [35] then this work, it was necessary to collect a good number of related researches to study them and extract useful information from them, as it was found that many implementations had used this algorithm, DT, over others or made it part of the set of algorithms used in their implementation. The majority of research that uses one or two methods, the DT should be among them.

Equation 3.1 was used to test the algorithms' performance using the accuracy metric.

$$Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \quad (3.1)$$

Where:

- a. TP: “True Positive”. A positive label sample that is in fact classified as positive
- b. TN: “True Negative”. A sample of a negative label that is in fact classified as negative.
- c. FP: “False Positive”. A sample with a negative label was inadvertently identified as positive.
- d. FN: “False Negative”. A sample with an indeed positive label was incorrectly identified as negative.

3.3.5 Work Constants

The accuracy of malware detection is affected differently by the features contained in the portable executable header, along with several other issues, this work primarily aims to determine and investigate this premise, all of these came from the outcomes that were obtained in the previous work [35]. Also, different standards and strategies were

implemented previously by other researchers to study and extract the features which they thought that they are sensitive and effective.

Various numbers features have been extracted, each implementation suggested after testing and studying a number of features which as they said the effective features that improve the accuracy of malware detection. Some of these extracted features were common between more than one implementation, for example `DllCharacteristics`. The work of [41] can present an example, where Python programming language is used to write a header parser and the work used *pefile* library. The work [41] stated that many features are included in the file header, features like `NumberOfSymbols`, `Machine` and `TimeStamp` in both the malicious and benign files have no difference in value, so, including such features will not be useful. On the other hand, Features such as `SizeOfInitializedData` in the optional header have a zero value in the malicious file, but a value other than zero in the benign file; this also holds true for `MajorImageVersion`, `checksum`, and `DLLCharacteristics`; accordingly, the work of [41] incorporated those features, also this paper included those features (`SizeOfInitializedData`, `MajorImageVersion`, `checksum` and `DLLCharacteristics`) in the implementation beside the other features used. It is important to emphasize that every feature used here has its own special role that will positively affect the process of distinguishing between the malicious and benign files, and those features are taken from previous implementations that mentioned the importance of these features after examining and verifying them.

This work has incorporated twenty-nine features, Table 3.6. Control dataset from which all the employed features were extracted obtained from [47].

DT and NN_MLPC were the algorithms that have been used in this work. `Test_size` and `Random_state` combinations used were (0.3, 10) and (0.15, 3) for both algorithms. Tests and previous work recommendations were the reasons behind employing those values [44], [35], [19]. Add to that the results obtained using those values reached to 99.23% according to [19].

Based on how (`test_size`, `Random_state`) is combined for every case study, four runs were done, a name is given to each run, and the names are as follow:

- a. NN_MLPC_0.3_10: This algorithm, which uses test_size of 0.3 and random_state of 10, is called Neural Network Multi-Layer Perceptron.
- b. NN_MLPC_0.15_3: With test_size of 0.15 and random_state of 3, the algorithm Neural Network Multi-Layer Perceptron is run under this name.
- c. DT_0.3_10: Given a test_size of 0.3 and a random_state of 10, the algorithm Decision Tree is run under this name.
- d. DT_0.15_3: The algorithm run using this name, Decision Tree, has a random state of three and a test size of 0.15.

Portable executable features can do a good job in malware detection [42], [46]. The form of the executable file and the layout of its sections [48]. The layout of the portable executable has two unused spaces, can be used in hiding watermarks, this will be a backdoor which can be used in necessary situations like password forgetting, on the other hand this backdoor gives hackers access to change the file's security. [40]. It is evident from the above that there is a need to more study and explore the portable executable header features.

This study used a novel approach to examine the impact of features—whether positive or negative—on accuracy. Each previous implementation has nominated a set of features that their experiments proved that they are the best to distinguish between benign and malware files. Some of these selected features were common between more than one work. This work has collected all the used features in previous implementations and studied them. The aim was to find out their effectiveness and if there is a way to group them all or some of them in a way so that optimal accuracy can be obtained. Add to that, If the greatest accuracy can be achieved with the fewest features possible. As a result, a huge number of runs were chosen for each kind. The run selection was according to previous indications obtained from the outcomes [35].

More than one reason was behind selecting this method of subgrouping, The primary goal was to determine which feature, if added, would improve accuracy? Prior research has indicated that incorporating features will enhance the accuracy, does all the added features have this effect? Finding a dependable set of features that can be used to distinguish between

malicious and benign files was the second motivation. The third reason was to figure out whether the feature combination will have an impact on malware detection accuracy?

It's crucial to note that a staggering number of runs were required to test every possible combination of the twenty-nine features, it was not possible to compute this. So, there had to be a way that will reduce the number of runs, but at the same time will take into consideration all the important milestones that will surely lead to new and improved results. For example, to start from a previously achieved result.

The programming language used to implement the algorithms and do the experiments of this work was Python, particularly Python 3.8. The employed package from Python was the SciKit package. Figure 3-7, show a general flowchart of the used algorithms. It was possible to use other programming language, such as Java, C++, Lisp or Prolog; but Python was selected since it has achieved great popularity recently, the reason is that Python include simple syntax, the process of testing can be relatively easier and the coding process can be truly little. A more important reason which led to choose Python over other programming languages was the several types of AI projects libraries that are built-in inside Python. For example, nltk, matplotlib, NumPy, SciPy and SimpleAI [71].

In this work, different tools were grouped together to build a system that can discriminate between benign and malicious files, hence the work incorporated a heterogeneous tool to work under the security title.

Finally, the hardware employed in this work was an ACER computer, which incorporates an Intel CORE™ i7-7500 U, 2.7 GHz with TURBO Boost up to 3500. The RAM was 8 GB DDR3 L Memory and the video card was NVIDIA GeForceMX130 with 2 GB VRAM.

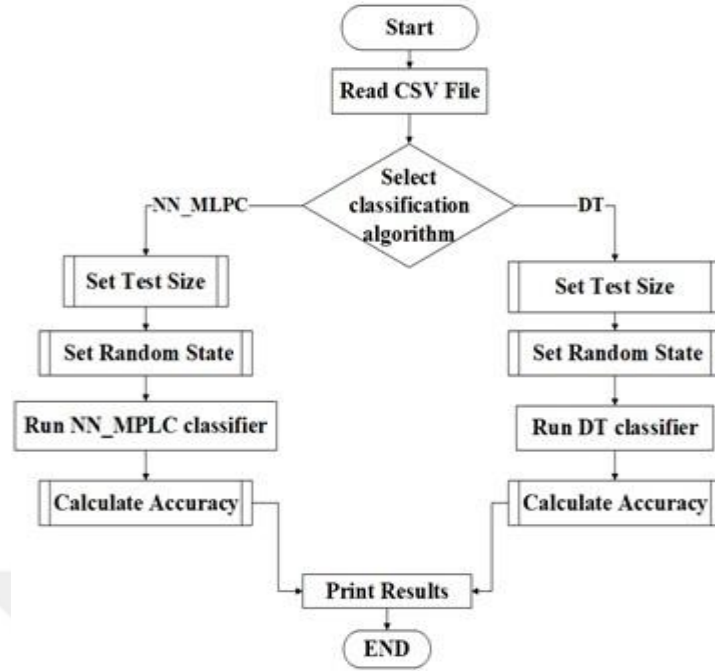


Figure 3.7: Algorithms General Flowchart.

3.3.6 Methodology - Subgrouping

Differentiating between the useful data and other data was important as a start, so, the previous work data have been studied to do this job.

Based on that, to differentiate between the number of features that produced high accuracy, good accuracy, and low accuracy, a distinguishing process was put into place. This was done to define the indicators for the method of subgrouping. Three levels of accuracy can be defined according to the outcomes and observations of previous results obtained in [35]. Note that the following standards were taken based on the attained results, maybe in other implementations the obtained results be in levels less than what were obtained, at that time the levels will be different.

- a. High Accuracy: the high accuracy for the NN_MLPC was the 0.97x range, note that the maximum accuracy obtained was in that range. In DT, 0.97x was obtained in many cases, but also DT attained a 0.98x accuracy in many other cases. So, for DT both 0.97x and 0.98x will be taken into consideration as the high accuracy level in this work. Note that, taking the 0.97x in DT into consideration since this is the high-level accuracy in NN_MLPC.

- b. Good Accuracy: According to the accuracies obtained previously, 0.95x and 0.96x will be considered as good accuracy levels.
- c. Low Accuracy: low accuracy will be indicated if the accuracy is in the range of 0.94x and below. Considering this range of accuracy as low-level accuracy because fewer runs achieved such accuracies compared with the runs that attained accuracies in the range of 0.97x and 0.98x.

To implement the subgrouping method, it was necessary to study the data collected from previous work and discriminate between useful data and other data; also, to exclude the data with low values.

It was noted that obtaining good accuracy level related to including different number of features, this applies also to obtaining high accuracy; maximum accuracy is obtained with numbers of features which were different in each case. In light of this, it was crucial to focus on the quantity of features that yields high, good, or low accuracy in order to define indicators for the subgrouping method. As an illustration, in the NN_MLPC, 0.97 was previously considered high accuracy, and the maximum accuracy obtained was within this range. Additionally, obtaining 0.95x and 0.96x can be regarded as good accuracy; however, starting at 0.94x and below can be regarded as low accuracy because there were few runs that produced such accuracy or less. Things are different with DT; in many cases, DT provided accuracy within 0.97x, but in many other cases, it provided accuracy within 0.98. Therefore, in DT, accuracies of 0.97 and 0.98x will be regarded as high; both accuracies will be considered in this work. In addition, accuracies of 0.94 or less will be regarded as low, relative to the number of runs that produced higher accuracies.

As mentioned previously, a tremendous number of runs will be needed to do all the combinations of the twenty-nine features, so, for each run of the four runs, a lower number of features were selected, this was done by subgrouping the twenty-nine features based on what was obtained previously. The following algorithm is employed for each type:

- a. A definition to the High, Good and low ranges of accuracies.
- b. According to previous results observations for both algorithms -NN_MLPC and DT-, the following were done:

- i. Selecting the minimum number of features that produced a high accuracy level. For example, eleven features in NN_MLPC attained an accuracy in the range of 0.97x which is the level of highest accuracy in NN_MLPC, so, the start will be with eleven features. A new set was created in a separated sheet, let's give it the name 'set x'.
- ii. Several new features were nominated to be incorporated in the experiments; the nomination was according to the previous observations.
- iii. Numerous runs were performed on the set x, the runs were done by replacing features with other features that were nominated previously and maintaining the quantity of features of set x fixed without changing it. Only one feature will be replaced during each run. Each run accuracy was recorded. Note that the number of features will not change, the change was with the features included.
- iv. Next step is to use more features, in this step more features will be added to the set x in steps, and new set will be created, the added features were brought from the nominated set of features.
- v. Now, from the nominated features from step b, a replacement to the features will be done and the run was repeated with each replacement, only one features was replaced, but the number of features was fixed.
- vi. Until the number of features reaches the number of features that achieved the highest accuracy in [35], steps c, d, and e will be repeated. This was the end of the process.

Organizing the work runs is essential to prevent completing all possible combinations of the 29 features, which will help to clarify the above points with additional details. In order to achieve this, fewer features were chosen for each of the four types of runs, i.e., the 29 features were sub-grouped based on earlier results. The following procedure is used for each kind. Based on the results that were available, the fewest features that yielded an accuracy that was within the high accuracy range were chosen. Several runs were made by replacing some features by other features from outside the initial set to establish the high accuracy with this number of features. This process is repeated several times with sets having more features. Thus, the set that produces the maximum accuracy is established. The number of features in

these sets was determined by looking at the prior outcomes, i.e., the feature sets that produced better accuracy than the others. The same concept guides the choice of exchange features.

According to previous work results [35], The quantity of features included in these sets was chosen. In other words, the selected sets of features attained an accuracy level which is higher than what was attained with other sets. The decision of selecting the features to be exchanged in each set was based on the same principal, those features were employed when higher levels of accuracy was attained.

When the set's feature count reached the number of features that achieved the highest accuracy in [35], the feature addition process was terminated. Given that this quantity of features was adequate to yield the highest level of accuracy. This choice was made in response to tests showing that adding more features than the number needed to achieve the highest accuracy in [35], but unfortunately, for all the algorithms the accuracy attained always lower than the maximum accuracy.

So, what was done is to test multiple conditions at the same time, first the quantity of features required to yield the highest level of accuracy, second, the combination of features used, and thirdly, if the feature will positively or negatively affect the accuracy.

The following steps were done to implement the experiments:

- a. Creating a new independent .csv file which includes the selected feature set elected for this particular set of runs (four runs).
- b. The Python program with the selected algorithm will import the created .csv file.
- c. Executing the runs for NN_MLPC and DT algorithms, each algorithm using the two established test_size and Random_state combinations. For every.csv file, four runs were made.

3.3.6.1 Runs for NN_MLPC_0.3_10

As mentioned before all the feature numbers, exchange process of features and runs done were based on what was obtained in [35]. In the case of NN_MLPC_0.3_10 same maximum accuracy obtained twice when employing twenty-one features and twenty-eight features. The

minimum number of features which attained an accuracy level of 0.97x which is the level of maximum accuracy was eleven features. So, the start was with eleven features. When performing the steps given in section 3.3.6 the maximum accuracy was attained using only fifteen features, while as mentioned, either twenty-one features or twenty-eight features attained the maximum accuracy in [35].

In NN_MLPC runs the feature combinations included were Eleven, Fifteen, Twenty-One and Twenty-Eight features, add to those combinations which includes more than Twenty-eight features and less than Eleven features were also tested to check the impact of such combinations.

Note that each set of features used in each step was taken according to certain observation from the work of [35]. For the eleven features step, this number was the first set of features reached an accuracy of 0.97x, which is within the range of the maximum accuracy. While employing Fifteen features as a second step came from the observations of the graph which became more stable when fifteen features were employed. Finally, the twenty-one and twenty-eight features were selected since those numbers of features had attained the maximum accuracy. Adding the step of twenty-five features was because this number considered as a middle point between the twenty-one and twenty-eight but the most important reason is that other algorithms obtained the maximum accuracy using twenty-five features. This criterion that was used to choose the number of features in each step were also applied to all the algorithms used.

3.3.6.2 Runs for NN_MLPC_0.15_3

In this case Twenty-three features attained the maximum accuracy in [35], based on that at most twenty-three features will be included in the experiments.

The starting point was with Thirteen features, since this number of features attained the range of 0.97x which considered as a high accuracy. The used subsequent sets were Fifteen, Eighteen, Twenty, Twenty-one, Twenty-Two, and Twenty-Three features, each number was selected according to certain observations, either related to what was obtained in NN_MLPC_0.3_10 or to what was obtained to DT or according to graph observations.

3.3.6.3 Runs for DT_0.3_10

The 0.97x accuracy level was obtained in [35] using Six features only, 0.98x accuracy was obtained using Fourteen features and twenty-five features to obtain the maximum accuracy. So, the steps were as follow: starting from six features, then Nine features, then subsequent combinations of Fourteen, Eighteen, up to maximum of Twenty-Five features were employed.

It is important to mention again that, taking the 0.97x level into account in DT while the maximum accuracy was in the level of 0.98x was done since this level of 0.97x was the level of maximum accuracy in NN_MLPC, and our experiments are interested in each indication from the employed algorithms, maybe the numbers used in one algorithm will affect the other algorithm positively, so, it was important to take the numbers and values in each algorithm into consideration in the other algorithm to attain the records and compare what will each number or value behave in the other algorithm.

3.3.6.4 Runs For DT_0.15_3

In previous observations [35] the accuracy level of 0.97x was attained using five features only, the 0.98x level attained with eight features only, you can note the big difference between NN_MLPC and DT. Also, maximum accuracy obtained previously was using Twenty-seven features.

Accordingly, the start was with five features, then Six, Eight, Nine, Ten, Fifteen, Eighteen, Twenty, Twenty-two and finally Twenty-seven features. The runs sets were carried out.

Seventy-Seven runs were the total number of runs that were carried out in NN_MLPC. While for DT the total number of runs were Seventy runs only.

It is notable that each algorithm has different number of runs, also, different number of added features. The reason behind that is the various observations and indications from [35], those observations were the pilot who guided the researchers to choose the features and their number, as well as the steps that were followed in choosing the number of features in each subsequent step, arriving to the quantity of features that yielded the highest level of accuracy. Each type produced different observations than the other, the employed steps and features

in each type differ from those in the other because those observations had varying numbers of features.

3.3.7 Results

To give a better understanding to the attained results, bar charts were used. Bar charts are self-explanatory charts. The charts illustrate the obtained range of accuracies for the runs of each type, also, the charts will show the change in accuracy with each case study.

All the attained accuracies from each run were sorted starting from the lowest value up to the highest value. In each chart, the variation in the attained accuracies with the various case studies will be shown. The set of features and the number of features will be represented in each case study. For all the charts drawn, the maximum accuracy and the related number of features used to obtain that accuracy will be shown in the last case study.

Table 3-7 include the features names, so, in the following sections the feature's number will be mentioned instead of the name of the feature according to Table 3-7. Note that it is possible to refer to Table 3-2 for more information about the reference from which the feature was nominated.

3.3.7.1 NN_MLPC_0.3_10

Figure 3-8 illustrate the NN_MLPC_0.3_10 bar chart. It is possible to see following points:

- a. The features Nineteen to Twenty-nine (see Table 3-7) attained the lowest accuracy. The obtained accuracy was in the level of 0.79x. Within 0.95x was a good accuracy level and high accuracy level was within 0.97x. The incorporation of the features between Twenty and Twenty-nine always deteriorate the level of accuracy to levels like 0.91x and 0.92x (low level), this will be clear. This will be clear during the discussion of the results later.
- b. Maximum accuracy attained when only Fifteen features were incorporated, the maximum accuracy was 0.97875595. Taking into account that the accuracy level of 0.978x was obtained also using Eleven, Twenty-One, Twenty-Five and Twenty-Eight features. But all the obtained accuracy were less than the Maximim accuracy attained with fifteen features.

- c. The level of maximum accuracy which is 0.97x was attained first using Eleven features, but not all the feature sets of eleven features attained this level of accuracy, some of these combinations obtained the minimum accuracy, this also applies to combination of features with more than eleven features.
- d. In the previous work of [35], an accuracy of 0.97790619 was obtained using Twenty-nine features, while in the current work, higher accuracies were obtained using subgroups of the Twenty-nine features. Sometimes a high accuracy level can be achieved with the same number of features, yet at other moments, when different features used the attained accuracy was low. As an example, it is mentioned in point B that Fifteen features were used to achieve the highest accuracy, while in other case when other combination of fifteen features were used, the attained accuracy was 0.95292318.

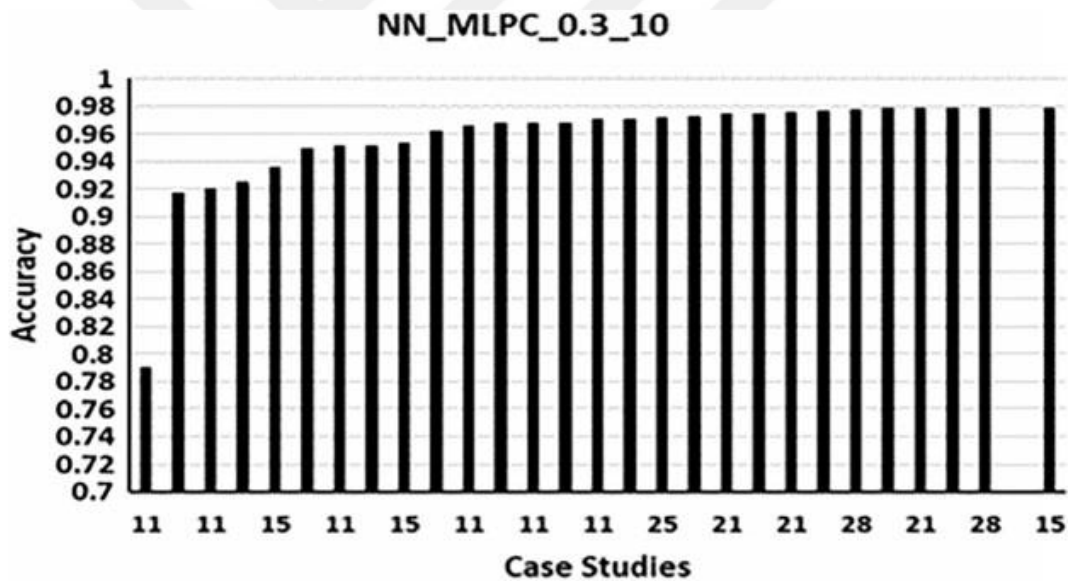


Figure 3.8: Bar Chart for NN_MLPC_0.3_10.

3.3.7.2 NN_MLPC_0.15_3

Figure 3-9 displays a bar chart that displays the outcomes of NN_0.15_3 analysis. The subsequent observations are indicative of:

- a. Fifteen features attained the lowest accuracy, the fifteen features were feature Twenty-nine down to feature Fifteen (see Table 3-7), the lowest accuracy was 0.91128484. It is notable that the features Twenty to Twenty-nine were also included

in this combination, so, the accuracy level was decreased. This situation is not the only situation which included those features (Twenty to Twenty-nine) and got a low accuracy level.

- b. Twenty-two features obtained an accuracy of 0.98130523, which was the maximum accuracy obtained. Note that this run was the only run in NN_MLPC which attained the level of 0.98xx, also, note that this level was not found previously in [35], and the maximum accuracy obtained was 0.976 using Twenty-three features.
- c. The accuracy level of 0.97xx was first attained using Thirteen features. Only one set of thirteen features attained the 0.97xx level the other combinations of thirteen features attained a lower accuracy.
- d. It was noticeable that if the features Twenty to Twenty-nine constitutes the majority of the combination employed in the run. A lower accuracy level was attained. For example, the features Fourteen to Twenty-six (Thirteen features) an accuracy of 0.93847723 was attained, another example, Eighteen features (One to Five, and Seventeen to Twenty-nine) the attained accuracy was 0.94323589. Another example, Twenty-Three features (Nine to Thirteen, One to Five and Seventeen to Twenty-nine) were employed, the accuracy obtained was 0.96057104.

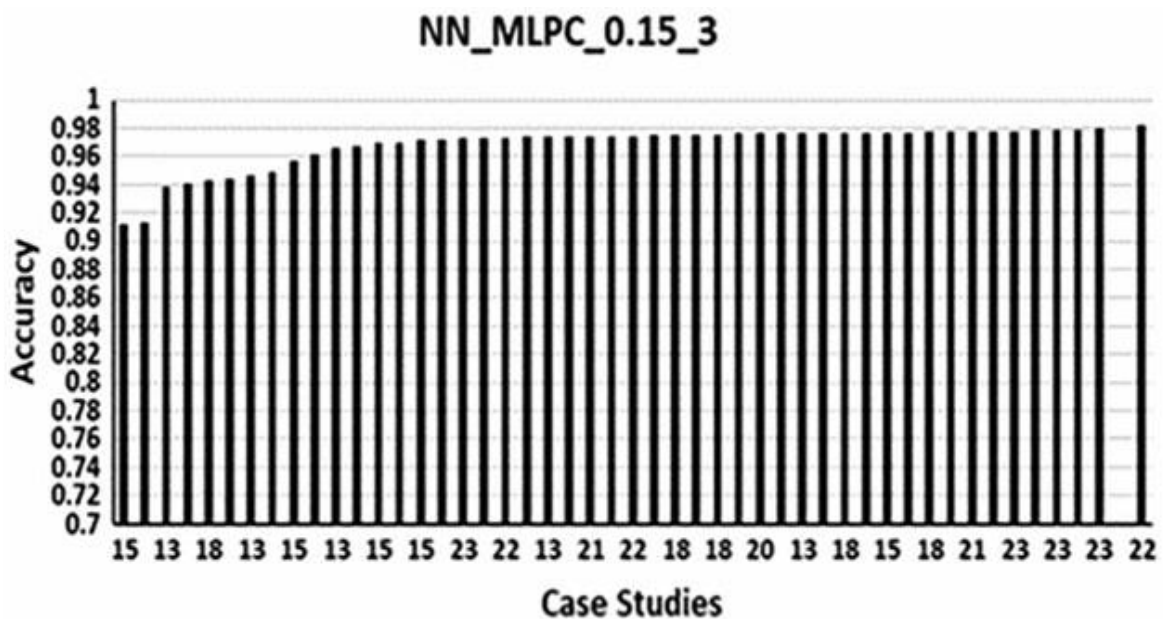


Figure 3.9: Bar Chart for NN_MLPC_0.15_3.

3.3.7.3 DT_0.3_10

Bar chart provided in Figure 3-10 displays the outcomes of the DT_0.3_10 runs. One can note the following observations:

- a. The result with the least accuracy was 0.75985724, the features that were incorporated in the run that produced this accuracy were Six features (Twenty to Twenty-five – Table 3-7). it is noticeable that the features used were also within the group of features that always lead to a lower accuracy level (features Twenty up to Twenty-nine). In order to conclusively support this hypothesis of the negative effect of those features, a special run was carried out using the features Twenty-one up to Twenty-nine (Nine features only), the accuracy obtained was 0.82545887. This also was noticeable in the runs of NN_MLPC.
- b. The accuracy of 0.98249490 was the maximum accuracy obtained in DT_0.3_10 using Twenty-five features, add to that, another combination of Twenty-five features had attained the second highest accuracy of 0.98164514. This give some indications on the number of features that should be used in DT_0.3_10.
- c. For the high-level accuracy 0.97x and above, the first 0.97x was attained using only Six features. Note that Six features previously attained the lowest accuracy, and now it attained a high-level accuracy, this means that the features used or the combination of features used have an impact on the accuracy maybe more than the number of features used. Since the lowest accuracy attained using the features Twenty to Twenty-nine, but another combination got a high-level accuracy.
- d. The level of 0.98x accuracy was repeated many times in DT_0.3_10 using different number of features the first accuracy obtained with this level was using Fourteen features which was 0.98113528, another combination of Fourteen features also attained an accuracy of 0.98028552.
- e. A 0.97824609 accuracy was obtained using Twenty-five features. On the other hand, and before three runs of this Twenty-five features run, a run with Eighteen features and two runs with Twenty-two features had attained an accuracy of 0.98x.

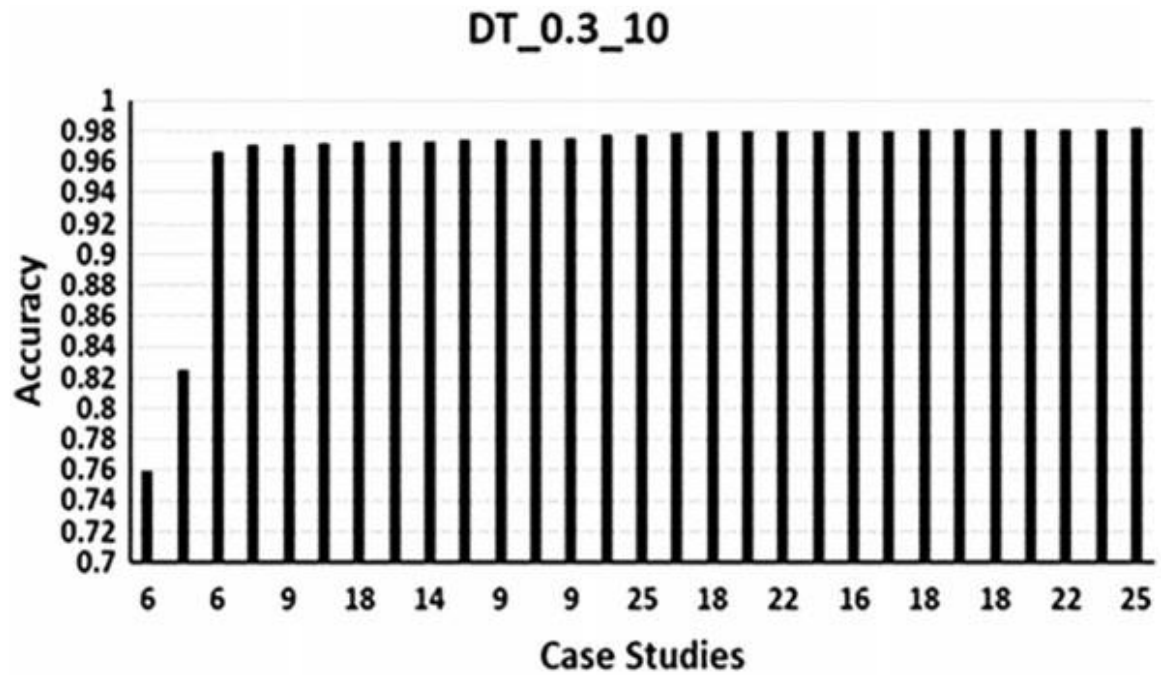


Figure 3.10: Bar Chart for DT_0.3_10.

3.3.7.4 DT_0.15_3

Bar chart presented in Figure 3-11 illustrates the outcomes of the DT_0.15_3 runs. One can note the following observations:

- Using Five features only, feature Twenty-One up to Twenty-five (Table 3-7), the obtained accuracy was 0.76614548, this was the lowest accuracy obtained in DT_0.15_3, note that the used features were also within the group of features that always lead to a lower accuracy level (features Twenty up to Twenty-nine), this low-level accuracy was expected. In Table 3-8, some examples are given which show the low-level accuracy obtained due to incorporate the whole combination of features from the features Twenty to Twenty-nine. In addition, it was notable that if the majority of features incorporated were from that range of features (Twenty to Twenty-nine) the resulted accuracy will be always lower than the normal, Table 3-9 show some examples on this aspect.
- When the features (One to Seven, Eleven to Nineteen and Twenty-one to Twenty-four Table 3-7) in total Twenty features, the greatest accuracy possible was attained, which was 0.98640381, note that this was the maximum accuracy ever attained, and no other algorithm or metrics within an algorithm got such accuracy.

- c. Five features attained the 0.97x level accuracy, 0.98x attained using Eight features, add to that Ten, Fifteen and Eighteen features attained an accuracy level of 0.98x and the maximum accuracy ever attained was using Twenty features.
- d. It was evident that a high level of accuracy was not always ensured by adding more features, for instance, when a specific arrangement of the eight features was used, 0.98x accuracy level was attained, however, the accuracy that was produced was frequently lower when the features were increased to a higher number. Also, many examples even from previous cases of NN_MLPC or the DT_0.3_10 can reflect or prove this issue. This means that the combination of features may be more influenceable on accuracy than the number of features, putting into consideration that there must be a minimum number of features incorporated to attain a higher accuracy. Note that this conclusion is stated for the second time, the first time was in DT_0.3_10 the point C,

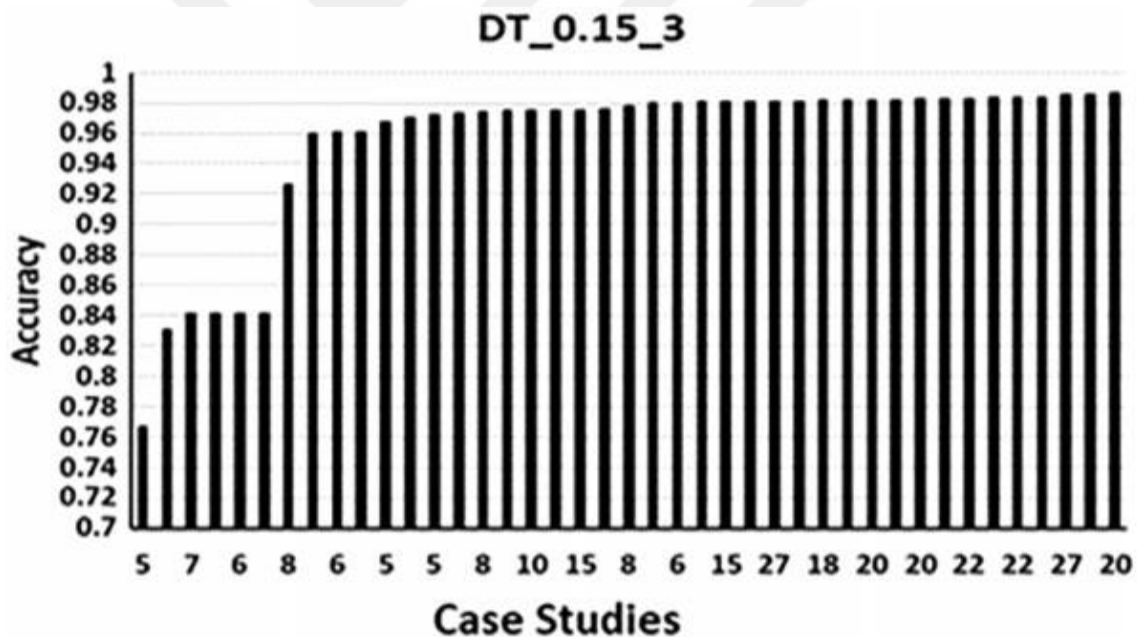


Figure 3.11: Bar Chart for DT_0.15_3.

Table 3.8: Low Accuracy Due to Use Only Features Twenty to Twenty-Nine.

Run no.	No. of features	Features used	Accuracy
Run132	5	21–25	0.76614548
Run133	5	25–29	0.83072739
Run139	7	21–27	0.84058464
Run151	10	20–29	0.84058464
Run135	6	21–23, 26–28	0.840924541
Run147	9	21–29	0.84092454

Table 3.9: Incorporating Half or Majority of Features from The Features Twenty to Twenty-Nine.

Run no.	No. of features	Features used	Accuracy
Run144	8	17–24	0.92624065
Run146	8	1–4, 21–24	0.95955133
Run136	6	1–3, 21–23	0.96057104
Run140	8	1–3, 21–25	0.96057104

In addition to the bar charts presented previously, two Combo charts shown in Figure 3-12 and Figure 3-13 were used also, this was done to give a better understanding and to give an example on how the result's analysis process had been done.

In Figure 3-12, all NN_MLPC_0.15_3 runs are shown, each run using the quantity of features and the accuracy that was attained. The accuracy is shown on the left axis, and features quantity is represented on the right axis.

A comparable Combo chart for the DT_0.15_3 is shown in Figure 3-13. It displays the accuracy as well as the quantity of features used for each run.

So, why choosing only NN_MLPC_0.15_3 and DT_0.15_3 to be presented in the Combo charts, this was because both of them got the highest accuracy between their classification algorithm. In Figure 3-12 and Figure 3-13, different obtained accuracies are shown, those accuracy levels obtained with various feature combinations. Those figures also show how features amount and employed features affect the accuracy. Each axis values suits the attained results for that algorithm. for the NN_MLPC_0.15_3, Since the lowest accuracy this algorithm could achieve was 0.91x, the starting accuracy value is 0.8. whereas for the DT_0.15_3, Since the lowest accuracy of 0.76 was recorded, the initial value was 0.7.

There is an interesting point that these charts had indicated, that the features used in the set have more impact on the obtained accuracy than the number of features. As an example, several runs with Thirteen features have been done, but each run has its own accuracy value which is different than the others, Because of the particular features that were employed, there was a discrepancy in accuracy. The same finding may be shown using different combinations. This point adds to the discussion of the features that are combined and how they affect accuracy. This can be supported by the observations about the features Twenty to Twenty-nine, if these features work together alone, a subset of them work alone or if a set is created with the majority of the features from these features, the obtained accuracy will be affected negatively or even the lowest accuracy will be attained. On the other hand, if certain combination of features includes one or two features from the features Twenty to twenty-nine, the obtained accuracy may be an enhanced accuracy or in the range of high-level accuracy, this was clear during the experiments that were carried out. Such cases will be presented next section.

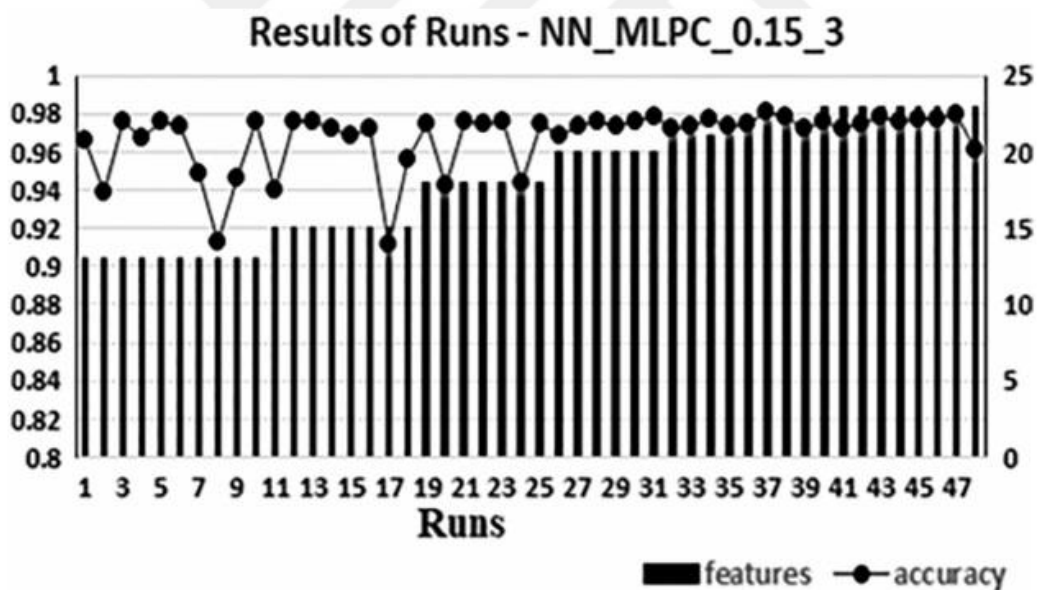


Figure 3.12: The NN_MLPC_0.15_3 Combo Chart.

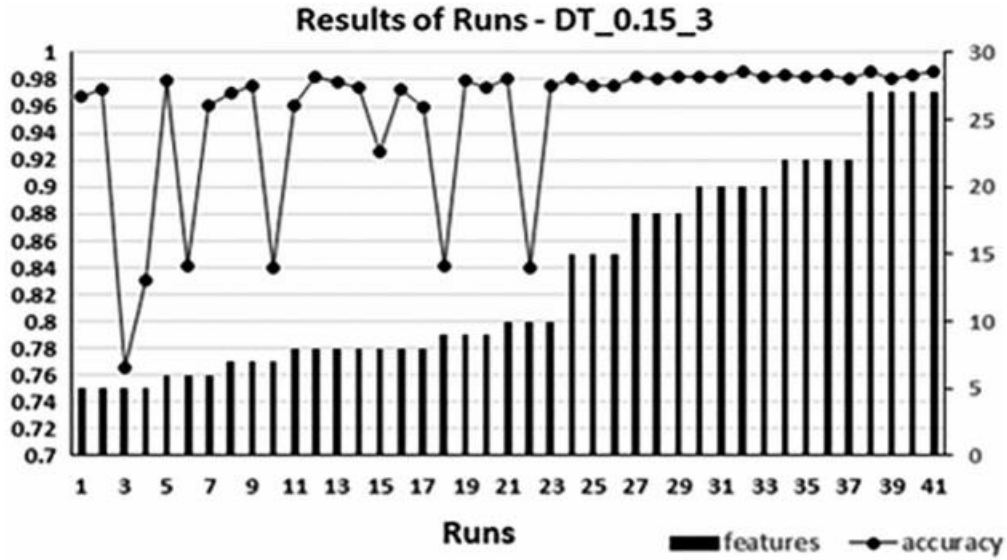


Figure 3.13: Combo Chart for the DT_0.15_3.

3.3.8 Observations Discussion

3.3.8.1 Observations of the work

NN_MLPC

The accuracies obtained in NN_MLPC_0.15_3 gave a different observation than what was obtained previously in [35], for example, particular combination of Eighteen features attained an accuracy of 0.9765, accuracy of 0.976 and 0.978 had produced using two different combinations of Twenty features. Also, an accuracy of 0.9769 had been obtained using Twenty-One features, Twenty-Two features attained an accuracy of 0.9786 and finally, different combinations of Twenty-three features had produced 0.9765, 0.978, 0.976 and 0.979 accuracy. Most of the mentioned accuracies are higher than what was obtained previously, in particular most of those values are higher than the maximum accuracy obtained previously. On the other hand, many feature combinations with the same numbers mentioned above attained an accuracy with lower values or even lower than the levels of good accuracy. It indicates that enhancing the accuracy level is significantly influenced by the feature combination.

It was stated in Results Section, in Point C of subsections of both NN_MLPC_0.3_10 and NN_MLPC_0.15_3, also from the observations of the previous implementations, that the first 0.97x in NN_MLPC_0.3_10 had been obtained using Eleven features, while NN_MLPC_0.15_3 required Thirteen features to produce that level of accuracy. This means

that a particular number of features were needed in each algorithm to reach certain level of high accuracy, for example Eleven for NN_MLPC_0.3_10 and Thirteen for NN_MLPC_0.15_3, putting into consideration the other conclusion which was what is the features that must be included in the combination of the Eleven or Thirteen features used. This applies to all the types used in this work or the algorithms types that were not included, each type will require certain features quantity to attain a high degree of accuracy. The conclusion regarding the number of features will primarily aid in or support the investigation into how the quantity of features affects the level of accuracy attained.

Using number of features less than Eleven or Thirteen will attain an accuracy level less than 0.97x, this will support the hypothesis of the quantity of features utilized is a crucial consideration. which affect the accuracy level. In many cases, Three, Four till Ten features were incorporated, the obtained levels of accuracy were sometimes lowest other times in the low-level range this reflects that the number of features must be a suitable enough to attain the required accuracies. However, it was observed that not every feature improves accuracy in the same way. It's necessary to recognize the features which have the positive impact and incorporating them will lead to increase the accuracy, even when using the same number of features.

Of all the NN_MLPC types, a specific combination of twenty-two features used in NN_MLPC_0.15_3 produced highest accuracy, the accuracy was 0.9813. The peculiar thing is that this set of twenty-two features was the only one with this quantity of features and attained the highest accuracy, Alternative combinations of the twenty-two features produced accuracies with lower values. This highlights the need to choose the appropriate feature combinations in order to improve malware detection accuracy. Also, it is significant to remember that this particular set of twenty-two features was the only one to achieve an accuracy level of 0.98x, this level of accuracy was not reached by any of the other runs in this work, including the other NN_MLPC_0.15_3 runs and the entire NN_MLPC_0.3_10 runs, or the runs of the earlier implementations [35]. The highest accuracy was attained at the level of 0.97x.

DT

In run 130 which was DT_0.15_3, only five features were incorporate, Features Six up to Feature Ten, the obtained accuracy was 0.96703, this accuracy was lower than the good level. Add to that several testing runs were done using three, four till ten features in both algorithms “DT and NN_MLPC”, the obtained accuracy in many cases were low and does not reflect the accuracy that make this number of features the correct choice. In other cases, five features produced an accuracy of 0.97x level, putting into consideration that the 0.97x was not the level of maximum accuracy, but in comparison with the NN_MLPC, 0.97x was the level of the maximum accuracy. Anyway, the key takeaway is that, as with NN_MLPC, employing few features will not yield the necessary degree of accuracy.

General remarks

Accuracy results obtained indicated that DT received the 0.97x accuracy using certain combinations of Five features and also Six features, on the other hand NN_MLPC attained an accuracy of 0.97x when certain combinations of Eleven features with NN_MLPC_0.3_10 and Thirteen features with NN_MLPC_0.15_3 were employed. DT obtained statistics were not the same as the statistics obtained in NN_MLPC. The common facts between the DT and the NN_MLPC are those facts that answers the raised questions, beginning with the quantity of features effect on accuracy, the positive effect of specific features and the features combination used.

The implemented runs for both algorithms proved that the features Twenty up to Twenty-nine have a negative effect on accuracy, in other words, if they were employed alone (some of them or all of them see Table 3-8) or if a bigger set of features includes more than half of the features from those features (Twenty up to Twenty-nine) see Table 3-9, the obtained accuracy will be the lowest or at least lower than the normal ranges. On the other hand, in several situations, when one or a small set from those features were incorporated in a bigger set, a positive impact to accuracy will be achieved and the level of accuracy will be improved, provided that the rest of the features are brought from other ranges, i.e., features One to Ten or Eleven to Nineteen. An example which illustrates this situation, the feature number Twenty-Six was employed in one of the runs, the accuracy obtained was the maximum accuracy, other examples can present the positive impact of individual or small

set of the features between Twenty up to Twenty-nine. But experiments proved that employing them alone or including majority of them in the set will spoil the obtained accuracy.

It was noticeable that the features Twenty up to Twenty-nine behaved variously if a broader range of features contained them; several runs yielded the highest level accuracy and part of the features used in these runs were some or maybe all the features between Twenty and twenty-nine, putting into consideration that those runs include a high quantity of features also, the features includes bigger quantity of features from One to Ten and Eleven to Nineteen, since it was notable from Table 3-8 and Table 3-9 the negative effect of features between Twenty to Twenty-nine if majority or all the features were between Twenty to twenty-nine.

Observations on the acquired results showed that numerous individual features can increase accuracy. In addition, certain feature sets can increase accuracy in NN_MLPC and DT if they are included within a larger feature set. Furthermore, certain features or feature combinations have an effect on NN_MLPC that is different from their effect on DT. The two algorithms differ in the additional features required to achieve the highest accuracy, but overall, the greatest impact can be obtained from these sets.

According to what is mentioned, regardless of the algorithm, it may be believed that some features or feature sets can improve accuracy.

A specific number of features were used in both algorithms to enter the region of high accuracy; additional features were added to achieve the highest accuracy. The maximum accuracy from the two algorithms is shown in Table 3-10.

Table 3.10: Maximum Accuracy Obtained.

Run type	No. of features	Test_size	Random_state	Accuracy
NN_MLPC_0.3_10	15	0.3	10	0.978756
NN_MLPC_0.15_3	22	0.15	3	0.981305
DT_0.3_10	25	0.3	10	0.982495
DT_0.15_3	20	0.15	3	0.986404

3.3.8.2 A Comparison study

The start will be with the implemented methodology, this work had employed the features subgrouping methodology so that testing process can be carried out, and to discover when and how to attain the maximum accuracy and answer some questions meanwhile, less time and fewer runs will be needed to complete everything.

The methodology used in this work demonstrated greater effectiveness in obtaining the answers to the issues brought up in this part. Obtained results were more, the same or slightly less than what was obtained in previous implementation of [35], this was from the numerical part point of view, for example, an enhanced and new level of accuracies were obtained in NN_MLPC, while in DT, the obtained results were also enhanced but the maximum accuracy attained was slightly less than what had obtained previously. The interesting and most important thing is that outcomes from this work answered the questions and achieved the desired goals of the work.

The incorporated portable executable features of this work were collected from previous implementations which used different strategies in studying and employing those collected feature and attained good level of accuracies, those features were used also in previous implementation [35].

The findings from this study were contrasted with those from [35]. in Table 3-11. After that, numerical results of this work are compared to what is attained in other previous works.

An enhanced outcomes and lower quantity of features have been used to obtain the maximum accuracy in NN_MLPC. Only Fifteen features were employed in NN_MLPC_0.3_10 to obtain the maximum accuracy, while in [35] Twenty-one or Twenty-eight features were used to get the maximum accuracy, putting into consideration that the maximum accuracy obtained in this work is higher than that of the previous work. This contributed to the fact that the combination of features used will have an impact on the accuracy, the evidence is that in this work other combinations of Fifteen features were employed or even more features till Twenty-eight features were employed, but the attained accuracies were lower or in some runs the accuracies were the minimum. From the above it can be said that the features employed in the run, add to that, both the quantity and combination of features all of them are considered among the most important factors that

affect the level of accuracy, and using a large number of features does not guarantee achieving high accuracy.

In DT_0.15_3, a slightly better accuracies were obtained in [35]. The quantity of features was same in DT_0.3_10 in both implementations, this work and [35]. Putting into consideration that feature number Twenty-five was employed in [35], while in this work the feature number Twenty-six was used instead of feature Twenty-five. It is notable that features Twenty-six has a negative effect on accuracy or it has no effect at all in the run of this work, or maybe it can be said with this combination of features it has no effect, since in other situations feature Twenty-six had improved the accuracy to the maximum accuracy as mentioned previously. Accordingly, this combination of features would have a better effect if feature Twenty-five was included instead of feature Twenty-six.

In the situation of DT_0.15_3, Twenty features attained maximum accuracy which is 0.001 less than the maximum accuracy attained in previous work [35], noting that the maximum accuracy attained in [35] was using Nineteen features and Twenty-seven features.

Table 3.11: Previous Work and Current Work.

Run type	Current work		Previous work	
	Accuracy	Features	Accuracy	No. of features
NN_MLPC_0.3_10	0.979	15	0.978	21 and 28
NN_MLPC_0.15_3	0.981	22	0.979	23
DT_0.3_10	0.982	25	0.984	25
DT_0.15_3	0.986	20	0.987	19 and 27

If the incorporated Twenty-nine features, see Table 3-7, three categories are used in this work: one for items one through ten, two for items eleven through twenty, and a third for items twenty-one through twenty-nine. The use of features from the three categories was used for all high-level accuracy calculations, as will be noted. Additionally, low-level accuracy was achieved with features with varying percentages from the three sets.

Table 3-12 lists several cases of the runs that were done including accuracies and used features in this work. The table will present many runs which attained an accuracy near the maximum accuracy in both DT and NN_MLPC, but the target was to find a way to produce the highest accuracy using the suitable combination of features and quantity of features.

Putting into consideration that attained maximum accuracy using this methodology was using a smaller set of features.

In the work of [44], the attained accuracies were 97% and 98% (fractions were not announced), putting into consideration that the work had implemented on DT, in this work the same levels of accuracies has been obtained but the maximum accuracy in this work was 98.6%. This accuracy is also higher than the 98.26% accuracy obtained in [37] which employed also the DT algorithm.

Table 3-12 Columb names:

- a. Algorithm and Run number (Algorithm_Run NO.: where the algorithm name is mentioned as either NN (NN_MLPC) or DT (Decision Tree).
- b. The utilized quantity of features
- c. Which features are used (See Table 3-6).
- d. Test_size
- e. Random_state.
- f. Attained accuracy in that Algorithm_Run

Table 3.12: Samples of the Runs Done.

1	2	3	4	5	6
NN_Run 33	11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 28	0.3	10	0.978 1
NN_Run 35_1	11	1,2, 3, 4, 5, 10, 11, 12, 13, 21, 28	0.3	10	0.950 7
NN_Run 36_1	11	1, 2, 3, 4, 5, 20, 21, 22, 23, 24, 28	0.3	10	0.919 8
NN_Run 37	15	1, 2, 3, 4, 5, 15, 16, 17, 18, 19, 21, 23, 27, 28, 29	0.3	10	0.961 8
NN_Run 39	15	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 21	0.3	10	0.978 8
NN_Run 40	15	15,16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29	0.3	10	0.917 2
NN_Run 42	21	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15,16, 17, 18, 19, 20, 21, 22, 23, 24, 28	0.3	10	0.972 8
NN_Run 44	21	20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 1, 2, 3, 5, 13,14, 15, 16, 17, 18	0.3	10	0.978 2

Table 3.12: Samples of the Runs Done “Table Continued”.

NN_ Run 46	21	1, 2, 3, 5, 6, 7, 9, 10, 11, 13,14, 15, 17, 18, 19, 12, 22, 23, 25, 26, 27	0.3	10	0. 97 58
NN_ Run 48	25	1, 2, 3, 5, 21, 22, 23, 24, 25, 26, 27, 28, 29, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16	0.3	10	0. 97 86
NN_ Run 50	25	11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 1, 2, 3, 5, 6, 7, 8, 9	0.3	10	0. 97 20
NN_ Run 52	28	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29	0.3	10	0. 97 77
NN_ Run 56	13	25, 26, 27, 28, 29, 10, 11, 12, 13, 14, 1, 2, 3	0.1 5	3	0. 96 70
NN_ Run 61	13	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25	0.1 5	3	0. 94 63
NN_ Run 65	15	1, 2, 3, 4, 5, 10, 11, 12, 13, 14, 25, 26, 27, 28, 29	0.1 5	3	0. 97 55
NN_ Run 68	15	21, 22, 23, 24, 25, 11, 12, 13, 14, 15, 1, 2, 3, 4, 5	0.1 5	3	0. 97 18
NN_ Run 71	18	1, 2, 3, 4, 5, 10, 11, 12, 13, 14, 21, 22, 23, 24, 25, 9, 17, 29	0.1 5	3	0. 97 45
NN_ Run 76	18	1, 2, 3, 4, 5, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29	0.1 5	3	0. 94 32
NN_ Run 79	20	11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1, 2, 3, 4, 5, 25, 26, 27, 28, 29	0.1 5	3	0. 97 38
NN_ Run 83	20	3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22	0.1 5	3	0. 97 82
NN_ Run 85	21	11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1, 2, 3, 4, 5, 25, 26, 27, 28, 29, 9	0.1 5	3	0. 97 35
NN_ Run 88	22	10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 1, 2	0.1 5	3	0. 97 42
NN_ Run 93	23	1, 2, 3, 4, 5, 6, 7, 8, 9, 22, 23, 24, 25, 26, 11, 13, 28, 15, 16, 17, 18, 19, 20	0.1 5	3	0. 97 25
NN_ Run 100	23	9, 10, 11, 12, 13, 1, 2, 3, 4, 5, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,	0.1 5	3	0. 96 06

Table 3.12: Samples of the Runs Done “Table Continued”.

DT_R un 102	6	10, 11, 12, 13, 14, 15	0.3	10	0. 97 86
DT_R un 105	9	1, 2, 3, 7, 8, 9, 15, 20, 25	0.3	10	0. 97 18
DT_R un 108	9	24, 25, 26, 14, 15, 16, 4, 5, 6	0.3	10	0. 97 62
DT_R un 111	14	1, 2, 3, 7, 8, 9, 10, 27, 28, 29, 17, 18, 19, 20	0.3	10	0. 97 40
DT_R un 112	14	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18	0.3	10	0. 98 03
DT_R un 117	16	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20	0.3	10	0. 98 05
DT_R un 118	16	1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 11	0.3	10	0.97 26
DT_R un 120	18	12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29	0.3	10	0.98 11
DT_R un 122	18	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 2, 14, 24	0.3	10	0.98 10
DT_R un 123	22	1, 2, 3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29	0.3	10	0.98 13
DT_R un 124	22	1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24	0.3	10	0.98 01
DT_R un 128	25	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29	0.3	10	0.98 05
DT_R un 131	5	11, 12, 13, 14, 15	0.15	3	0.97 21
DT_R un 134	6	1, 2, 3, 6, 7, 8	0.15	3	0.97 96
DT_R un 137	7	11, 12, 13, 15, 16, 17, 25	0.15	3	0.97 01
DT_R un 141	8	1, 2, 3, 11, 12, 13, 14, 15	0.15	3	0.96 06
DT_R un 149	9	1, 2, 3, 11, 12, 13, 27, 28, 29	0.15	3	0.97 45
DT_R un 152	10	5, 6, 7, 8, 9, 10, 11, 12, 13, 14	0.15	3	0.97 48
DT_R un 155	15	5, 6, 7, 8, 9, 15, 16, 17, 18, 19, 25, 26, 27, 28, 29	0.15	3	0.97 48

Table 3.12: Samples of the Runs Done. “Table Continued”.

DT_R un 157	18	1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 11, 7, 29	0.15	3	0.98 06
DT_R un 160	20	1, 2, 3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27	0.15	3	0.98 23
DT_R un 162	20	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 25, 26, 27, 28, 29	0.15	3	0.98 20
DT_R un 166	22	1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24	0.15	3	0.98 10
DT_R un 169	27	1,2, 3, 4, 5, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 7, 9, 11, 13, 14, 6, 7	0.15	3	0.98 37

3.3.9 Summary and Conclusions

Various prior implementations used various approaches to investigate PE files and how to detect malware which may resides in their headers, they had selected the features to be incorporated, some studies have been made on packed files, in all cases the portable executable header features will take the important role in identifying malicious files from the begnine files.

The features used in this work are Twenty-nine features, those features were collected previously in other previous work, the study has been made using subgrouping methodology and the resulted accuracy were observed after each run. Each group of runs had imported an independent dataset, this imported dataset were prepared specially for that group of runs.

The only two classification algorithms chosen to carry out the work's experiments were DT and NN_MLPC, the selection came after doing all the required tests and experiments to other classification and clustering algorithms. The group of runs includes four run types, NN_MLPC_0.3_10, NN_MLPC_0.15_3, DT_0.3_10 and DT_0.15_3, representing the test_size and Random_state, respectively, the included number in each type.

The attained outcomes of this work have two sections, Practical and Numerical sections. Starting with the practical section, this section includes all the deductions and observations which can be briefed by the factors that influence the malware detection accuracy, such as the used quantity of features, included features in the set and features combination used. On the other hand, the numerical section of the results were the accuracies obtained, which were

for NN_MLPC, the obtained maximum accuracies were 0.979 and 0.981 and for the DT the maximum accuracies obtained were 0.9825 and 0.986.

Outcomes of this work which represented by the obtained results were sometimes enhanced or higher than previous implementation, other times were equal and sometimes were somewhat lower than the findings of the earlier work. This was one of the work's objectives, in case that the obtained results were slightly lower accuracy this means that the features have kind of impact on accuracy, hence, a detailed studies must be carried on to those features and they must be taken into consideration in the future so that they can be excluded to get rid of their negative impact.

In this work, the subgrouping strategy has shown its high effectiveness when compared to previous implementations; previous observations were the main guide to do the experiments of this work, so, certain subsets of features were employed in steps and the reference was the previous work outcomes and observations, this process continued till reaching the maximum accuracy, this strategy reduced the time and the results were an enhanced and good.

As the work has demonstrated, the quantity of features used, the specific features included, and feature combinations are the key factors that contribute to achieving high accuracy levels. Other observations include high accuracy does not need high number of features, including a large number of features can occasionally result in the "curse of dimensionality" issue, or even can lead to obtain lower levels of accuracy since some of the incorporated features may have negative effect on accuracy.

The improved results achieved by this work necessitated a deeper study of the features in future work using different strategy, also, new features can be added to study their influence on accuracy. The idea also includes incorporating more new metrics to measure the performance, those new added metrics may include Specificity, Sensitivity, Precision, GMean and/or FMeasure.

4. ONE-CLASS CLASSIFICATION AND SUBSPACE LEARNING- BASED ONE-CLASS CLASSIFICATION IN MALWARE DETECTION

4.1 CHAPTER INTRODUCTION

In chapter three, stage one of the dissertation has been introduced, a set of twenty-nine features, which were collected from more than one work, as these works have nominated these features as having the most influence on malware detection accuracy. A detailed study has been conducted on these features, and the study of chapter three has been divided on two separate works part one and part two. All the conclusions from the conducted studies were the quantity, the particular combination, and the variety of features utilized, this means taking features from all the categories, the categories were Features One to Ten, Eleven to Nineteen and Twenty to twenty-Nine (according to Table 3-2). So, incorporating a variety of features from all the defined categories taking into consideration that the number of features from the third category must be the least, add to that, the impact of each feature will play the main role in obtaining the highest level of accuracy.

All what was mentioned above is good. But still building a security system that can detect malware need a more comprehensive study which can see the problems and the solutions from all points of views, there is another point of view that must be taken into consideration, since not only the features will play the main role in the process of detecting malware accurately, the specifications that the algorithm possess in ensuring that the detection process is operating properly as well as in the detection accuracy, in other words, the outcomes of the algorithm are real and not only theoretical, from these specifications are the employed metrics used in measuring the performance and if the algorithm can avoid the problems that may be an obstacle in achieving the right results.

First, let's discuss the measures -metrics- that can be applied to accurately gauge the efficacy and results. of the algorithm. Previously the accuracy metric was the only adopted metric in measuring the outcomes. It is known that other metrics can be incorporated and may give a better picture to the performance. From these metrics Sensitivity, Precision, Specificity, FMeasure and Gmean.

As for the problems that may be an obstacle to achieving the right results, the Curse of Dimensionality and the dataset Imbalance problems, those are the main problems that may mislead the classification process.

Other issue that must be mentioned, is the algorithm itself, the behavior of it, how it can deal with the classification problem. Previously in chapter three, the process that has been carried out in features studying was an important step to check the validity of the portable executable header features, the process of selecting the algorithms that had been used took long time of testing and experimenting each nominated algorithm till reaching the correct choice, this choice was built on the outcomes and the characteristic of each algorithm, so the selected algorithms were suitable for that job, the job of studying the features and see their influence on accuracy. Now, things are different, the principal objective is to construct a system or prototype with a holistic vision., this vision take into consideration how to optimize the detection process. This will need a solution that take all kind of problems away from the way that this strategy follows. This solution must take Curse of dimensionality, Imbalance of the dataset, other features problems into consideration and use a clear methodology which bypass these problems and concentrate on one target, how to detect malware in a reliable and accurate strategy.

So, in this work things will be different, the needed algorithm must do the whole job accurately and the features study which had been conducted will not be part of the problem, since the important role of the features have been proved, now, they will be a player in the new implementation. The work of chapter three can take another path to reach other goals represented with the set of features that can be used in certain situations to obtain a better detection accuracy, but this target now is postponed, since the main aim is to generalize a strategy using known algorithms that can face the problem of malware detection.

It is important to mention that computer systems security considers malware detection an important issue. Several traditional machine learning models may face some problems in malware detection, these problems may lead to not effectively detect the malware. From these problems is the class imbalance, in which one of the classes usually the malware class samples are smaller than the number of benign samples. Another problem is that the evolving and dynamic nature of the malicious software, the structure and tactics of the malware can continuously change, this constitutes a big challenge to the traditional

algorithms of artificial intelligence, and can lead to security vulnerabilities through misclassifying the benign as malware or vice versa.

Stage two of the dissertation employed the One-Class classification (OCC) methods are a promising technique that can face these problems and improve the detection process of unknown malware or manipulated file features, this will be done by recognizing the samples that vary from the majority of the known samples.

Unfortunately, the challenge of Curse of Dimensionality faces the most conventional approaches of one-class classification.

Accordingly, the candidate has moved to stage three, this stage has adopted the subspace learning-based OCC methods to face the curse of dimensionality problem and meanwhile can manage the problem of class imbalance.

This work has proposed a malware detection pipeline which utilizes techniques that jointly can optimize a data description and subspace for OCC.

Differently from what was used in the chapter three, in this application three datasets were chosen to participate in this work, while stage one presented in chapter three has employed only one dataset. incorporating three different datasets were to measure the performance of different One-Class classification used techniques. The employed techniques of the subspace-learning-based data description showed an enhanced performance and outperformed other conventional One-Class classification malware detection techniques. The pipeline proposed can be considered as a worthy tool for improving computer system security through detecting malware accurately. Add to that, the proposed pipeline can protect the computer system against unknown prospected attacks.

4.2 INTRODUCTION

Various threats can face computer systems, one of the oldest threats is the malicious software; malware founded even before the era of Internet and the wide spread of computer networks. Damaging the system software or altering the work of the computer are from the effects that the malware can do to the computer. Also, after the spread of networking, the whole network can be affected. The companies developing antivirus has a considerable challenge represented by malware detection. Between the various operating systems that can

be affected more with malware is the Windows operating system, it can easily be hacked in order to spread malware; hence, exploiting Windows vulnerabilities will be more prone. Accordingly, malware detection has always been a serious challenge in data privacy protection [72].

Securing the networking systems importance has been increased since the propagation of Internet of Things (IoT) devices. Botnets and other malicious software activities can exploit the vulnerabilities of IoT devices [73], [74]. In particular, the Mirai attack has opened the horizon to do more research in the IoT context [73], [75]. But still, malicious software constitutes a big threat to computers and information systems, in spite of the existence of different antivirus software [53]. Techniques used by hackers for evading from malware detection varied between mutation and evasion, this was coincided with the evolution of malware to become more complex [76], [77]. The devices that may face the risk of malware infection are the electronic devices which stores a large amount of data, examples on the malware types that may infect such devices are Trojans, viruses, worms and Ransomware [78]. The Internet easy access and the fake URLs that may exist are some of the reasons which may lead to infect the devices with malware [79]. Software applications can be installed on Windows operating system machines using a .exe, .msi and .bat portable executable (PE) files, the people who developed this type of files can modify it in order to deceive the antivirus security programs [80]. The portable executable file includes several variables and fields, this complex structure of the portable executable file makes it an easy choice for malware instances hiding so that virus detection software cannot find them easily.

Tools that are based on Machine Learning (ML) have been utilized in various fields, security was one of these fields, given that tools for machine learning can enhance the malware detection efficiency and performance [81] – [84]. Unfortunately, conventional machine learning algorithms have some obstacles and limitations when employed in the field of security. For example, collecting the adequate number of the training samples especially the negative samples so that a malware detection model can be created is not practical, this is because the hackers are developing new malware in a continuous manner and the different types of malwares can exist during the training phase are almost endless. It was necessary to address the problem of data imbalance, this can be done through incorporating a machine

learning algorithm which will use data from only one class in the training process, the algorithm is the One-Class classification (OCC) [85] – [87].

Different approaches have been used to find malware in portable executable files, these strategies incorporated machine learning algorithms. Previously in chapter three two separate works have been conducted in this field [35], [36] both works used NN-MLPC and DT. Unfortunately, problems like curse of dimensionality and the class imbalance cannot be addressed by these algorithms. To cope these challenges, a novel pipeline is introduced in this work, this pipeline can improve the methods of the subspace learning-based OCC. These methods can mitigate the problem of curse of dimensionality in addition to facing the class imbalance problem. Figure 4-1 introduces the pipeline proposed in this work, this figure highlights the subspace learning-based OCC methods employment.

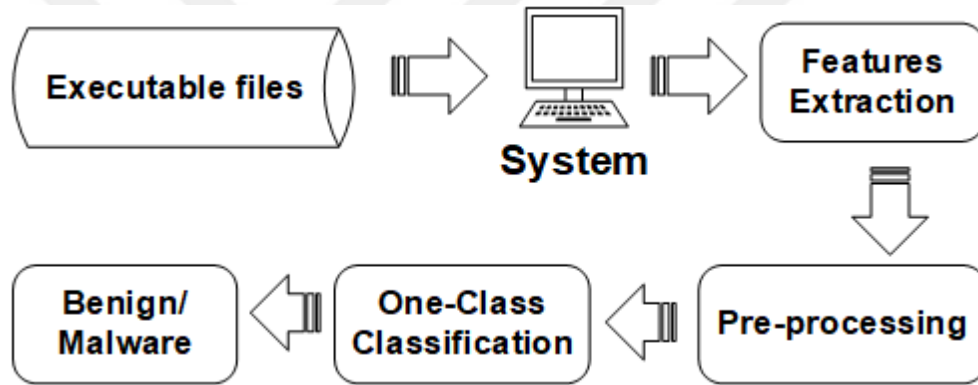


Figure 4.1: The Proposed Pipeline.

Three different datasets were used in this work [47], [44], [88]. An evaluation to the performance of the different OCC methods included in the pipeline has been done, the evaluation includes One-Class Support Vector Machine (OCSVM) [14] and Support Vector Data Description (SVDD) [15], the OCSVM and the SVDD were implemented by employing the library of LibSVM [16] this was done during stage two of the work, in addition to that, another type of evaluation has been conducted in stage three, which includes the proposed techniques that were used to face the challenge of the curse of dimensionality that OCC algorithms suffer from. Those proposed algorithms are the Subspace Support Vector Data Description (SSVDD) [17] and Graph Embedded Subspace Support Vector Data Description (GESSVDD) [18]. The linear and nonlinear data description results were reported for all the included methods in the evaluation.

4.3 MALWARE

Malware which is also called Malicious Software is a sequence of instructions that form a piece of code, The malware's primary goal is to interfere with the system's regular operation. The ways that the malware may affect the system vary, some of them try to steal the confidential information of the system through attacking them, in other types the attacker try to implant the malware in the system so that he can gain control through remote access. Based on its objectives and behavior, malware can be classified into a number of categories, for example, Botnet, Trojan, spyware, scare-ware, worms and Viruses [72], [89]. The work of [90], had defined the HBCIA (Host-Based Code Injection attack) to be the malware copying process locally and then directly executing it inside a foreign process; this process as [90] stated make the attacker reach his goals, hence many malware authors have used it. Many features the malware can have, such as endpoint exploit, industrial spying, data stealing, denial of service capabilities, aiming to target numerous architectures, detection avoidance features, use of different methods to scan the web to find new targets, enhancing efficiency such as port closing and infecting new devices using different strategies [91]. Botnet malware can include such features, but all of those features or some of them can be included in all malware in general.

Section header in the portable executable file provides information such as the length, location and characteristics. The structure of the portable executable file includes section table and file header, then section data. Also, the MS-DOS header, there is also the PE signature, then the image file header and then the optional header. The portable executable file includes also numerous fields and redundant spaces. Unfortunately, this gives the attackers the opportunity to implant their malware and the malware to propagate then hide. Malware and benign software can differ greatly in their format information, these redundant spaces and fields can make these differences where the malware code will reside in these spaces.

Many differences in the format information can be found when comparing malware and benign software [39]. Based on what was presented, features studying can contribute to the process of discriminating the files as benign or malware.

Signature-based technologies and anomaly-based technology are the main strategies which can be used to detect malware. In the Anomaly-based detection strategy two main steps or

situations must be carried out, the first is to learn or train the algorithm; in this step the detectors are used to learn the normal behavior, whereas the second step or situation is the monitoring and detection situation. Hence, the Anomaly-based technique must decide the behavior which is normal so that the malware strength investigation can be done [10]. On the other hand, in the strategy of the signature-based, a database or repository must be predefined, and includes all the malwares like Trojans, worms, viruses, etc. and such dataset or repository must be included in the system protection software, the predefined malicious software in the dataset are the expected malwares that may at any time attack the system. Accordingly, the system protection software must update its repository or dataset continuously with the expected malwares in order to not miss any new attack. It is important to know that, in the field of computer system or computer networks, the term signature is a style that is directly connected or linked with malicious attack.

4.4 SV BASED OCC FOR MALWARE DETECTION

OCC is a method used in machine learning. The OCC's approach is used to find instances that fit into one of the classes, which is the **target class**. Other names to OCC like novelty detection, concept learning and outlier detection [31]. OCC is useful when there is a shortage in the samples or it is difficult to obtain data or samples from other classes [32]. Three categories of methods are included in OCC, those methods are boundary-based, reconstruction-based and density estimation-based. This work concentrates on the boundary-based methods. This boundary-based include techniques like OCSVM and SVDD, both techniques will have a boundary that will be defined using statistical methods, The target class will be isolated from everything else by this boundary. [92]. Creating a tight boundary that will enclose the target class is the primary goal of the SVDD and OCSVM, the boundary must be as tight as possible. The boundary will be defined using the support vectors samples, these support vectors will be crucial in distinguishing the target class from other samples in the feature space. More details of the techniques employed in this work will be presented in the following subsections.

4.4.1 One-Class SVM

The OCSVM is used for the tasks of OCC, OCSVM considered as a variation of the SVM [93], [94]. The OCSVM's objective is to look for **hyperplane** within the feature space, the origin of the coordinate system must be as far as possible from this hyperplane, since this

coordinate system represent negative class hyperplane. The farthestmost half of the hyperplane from the origin defines the positive class. The constant b (referred to as the bias) and the vector w defines the hyperplane which can be written as follow:

$$w^T x + b = 0 \quad (4.1)$$

The SVM and the OCSVM are based on the same principle, they are both a popular tool that perform a binary classification task. A kernel function will be employed by the OCSVM in the non-linear cases, just like the SVM. This kernel function's purpose is to implicitly map an original space into a higher-dimensional feature space. The data can be detached using the hyperplane in that feature space. Instead of maximizing the footnote between the two classes, OCSVM aims to find a hyperplane that is sufficiently distant from the origin. [92]. Suppose the matrix $X = [x_1, x_2, x_3, \dots, x_N]$, where $x_i \in \mathbb{R}^D$ represents the training data matrix which is employed to get the boundary, where the entire sample count within the D -dimensional feature space is represented by N . OCSVM classifier objective function is represented by:

$$\begin{aligned} \min_{w, \epsilon, b} \quad & \frac{1}{2} \|W\|^2 + C \sum_1^N \xi_i - b \\ \text{s.t.} \quad & (W^T \phi(X_i)) \geq b - \xi_i, i = 1, \dots, N \\ & \xi_i \geq 0, i = 1, \dots, N. \end{aligned} \quad (4.2)$$

ξ_i represents a slack variable which allows for some training errors, and the hyperparameter $C > 0$ regulates the trade-off between the permitted number of errors and the margin's size. Non-linear relationships in the data can be handled using the function $\phi(\cdot)$ which is a non-linear function, this function can map the points of the data to the kernel space.

4.4.2 Support Vector Data Description

In SVDD a **hypersphere** serves as a boundary that is defined to surround the samples used for training in the given feature space, while in OCSVM the used boundary is a hyperplane. Minimizing the volume of the hypersphere is the main goal of the SVDD [95][15]. The following can formulate the optimal hyper-sphere finding problem:

$$\begin{aligned}
& \min_{R,a} R^2 + C \sum_{i=1}^N \xi_i \\
& \text{s.t. } \|x_i - a\|^2 \leq R^2 + \xi_i, i = 1, \dots, N \\
& \xi_i \geq 0, i = 1, \dots, N
\end{aligned} \tag{4.3}$$

The hypersphere center is indicated by the variable $\mathbf{a}$, $\mathbf{a} \in \mathbb{R}^D$, the radius of the hypersphere is denoted by R . The rest of the variables are similar to the variables defined in OCSVM.

4.5 SUBSPACE LEARNING

4.5.1 SSVDD

In the Subspace Support Vector Data Description (SSVDD) approach, a subspace which is optimized is obtained for OCC along with the data description. The objective is to locate a matrix which is $Q \in \mathbb{R}^{d \times D}$ such that the lower dimensional $\mathbb{R}^d$ data will be mapped as $\mathbf{y}_i = Q\mathbf{x}_i$ can be compactly enclosed in an OCC boundary [17], [34]. The optimization problem of SSVDD can be defined as:

$$\begin{aligned}
& \min_{R,a} R^2 + C \sum_{i=1}^N \xi_i \\
& \text{s.t. } \|Q\mathbf{x}_i - a\|_2^2 \leq R^2 + \xi_i, i = 1, \dots, N \\
& \xi_i \geq 0, i = 1, \dots, N
\end{aligned} \tag{4.4}$$

Using Lagrange multipliers α (Alpha), the minimization function incorporates all of the constraints. The enhanced form of the Lagrangian $\mathcal{L}$ function can be expressed as follows:

$$\mathcal{L} = \sum_{i=1}^N \alpha_i \mathbf{x}_i^T Q^T Q \mathbf{x}_i - \sum_{i=1}^N \sum_{j=1}^N \alpha_i \mathbf{x}_i^T Q^T Q \mathbf{x}_j \alpha_j + \beta \psi, \tag{4.5}$$

In equation 4.5 the ψ represents the regularization term, in the projected space (low-dimensional) the class variance can be expressed using this regularization term. Using β , regularization term importance can be controlled. The following equation presents the definition of the regularization term ψ :

$$\psi = \text{tr}(Q X \lambda \lambda^T X^T Q^T) \tag{4.6}$$

Equation 4.6 show the trace operator $\text{tr}(\cdot)$, also, each sample contribution control during training in ψ is controlled using the trace operator $\lambda \in \mathbb{R}^N$. when using SSVDD, four cases need to be considered:

- a. Case 1, when setting the regularization term to Zero (0), in this case an obsolete form of regularization will be resulted. This situation is denoted as $\psi 0$.
- b. Case 2, describing the class variance using all data points. This can be done by using a vector of all 1s instead of λ . This situation is denoted as $\psi 1$.
- c. Case 3, in this case positive α values are used to describe the class variance. This situation is denoted as $\psi 2$.
- d. Case 4, in this case only the support vectors are taken into consideration, this is done by first, for the samples that are not belonging to the boundary, setting the values of λ_i to Zero, second, for the samples that belong to the boundary, setting the corresponding α_i values to Zero. This situation is denoted as $\psi 3$.

In order to gain the improved data description in the subspace resulted and also defining a data projection matrix Q which is optimized, an iterative approach is employed, in this approach SVDD solution is used to get the values of α , also, using a gradient step, an update is carried out to Q .

4.5.2 GESSVDD

Spectral regression-based solution and spectral solution are presented in the Graph Embedded Subspace Support Data Description (GESSVDD) [18], those solutions will be an alternative solution to the technique used in SSVDD which is the gradient-based technique. Add to that, using the graph-preserving criterion, the GESSVDD allowed the employment of different graph information, also, the GESSVDD has formulated the OCC problem as a graph embedding problem. The GESSVDD formulation is:

$$\begin{aligned}
 & \min_{R, u} R^2 + C \sum_{i=1}^N \xi_i \\
 & \text{s.t. } \|S_Q^{-1/2} Qx_i - u\|_2^2 \leq R^2 + \xi_i, i = 1, \dots, N \\
 & \xi_i \geq 0, i = 1, \dots, N
 \end{aligned} \tag{4.7}$$

Where u is a vector $u = S_Q^{-1/2} a$, a is the projected space center and the subspace geometric data relationships are encoded using the matrix S_Q , as follow:

$$S_Q = QXL_xX^TQ^T = QS_xQ^T \tag{4.8}$$

Where the graph Laplacian is represented by L_x . Depending on the used graph type it can take different forms. Using the Lagrange $\mathcal{L}$ multiplier, the corresponding dual objective function can incorporate the constraints of equation 4.7:

$$\begin{aligned} \mathcal{L} = R^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i (R^2 + \xi_i - (S^{-1/2}_Q Q_{x_i})^T S^{-1/2}_Q Q_{x_i} + \\ \mathcal{Q} u^T S^{-1/2}_Q Q_{x_i} - u^T u) - \sum_{i=1}^N \gamma_i \xi_i. \end{aligned} \quad (4.9)$$

The Lagrange multipliers are $\gamma_i \geq 0$, and $\alpha_i \geq 0$. Minimizing the equation 4.9 which is the Lagrangian equation in relation to R , u and ξ_i and on the other hand maximizing it in relation to the Lagrange multipliers γ_i , and α_i , in the event that the partial derivative is zero, the result will be as follow:

$$\frac{\partial \mathcal{L}}{\partial R} = 0 \Rightarrow \sum_{i=1}^N \alpha_i = 1, \quad (4.10)$$

$$\frac{\partial \mathcal{L}}{\partial u} = 0 \Rightarrow u = \sum_{i=1}^N \alpha_i S^{-1/2}_Q Q_{x_i}, \quad (4.11)$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \Rightarrow C - \alpha_i - \xi_i = 0 \quad (4.12)$$

Substituting the equations 4.10, 4.11 and 4.12 into equation 4.9, the result will be:

$$\begin{aligned} \mathcal{L} = \sum_{i=1}^N \alpha_i x_i^T Q^T S^{-1}_Q Q_{x_i} - \sum_{i=1}^N \sum_{j=1}^N \alpha_i x_i^T Q^T S^{-1}_Q Q_{x_j} \alpha_j \\ = \sum_{i=1}^N \alpha_i z_i^T z_i - \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j z_i^T z_j \end{aligned} \quad (4.13)$$

If equation 4.13 is maximized, this matches SVDD solving in the transformed subspace, this generates α_i values for every instance, indicating their locations within the data description. The instances that reside within the boundary will have $0 < \alpha_i < C$, if $\alpha_i = C$ then those instances reside outside the boundary, as for the instances that resides within the enclosed boundary their α_i values will have values equal to Zero ($\alpha_i = 0$).

$$\|z_i - u\|_2 < R \rightarrow \alpha_i = 0, \gamma_i = 0, \quad (4.14)$$

$$\|z_i - u\|_2 = R \rightarrow 0 < \alpha_i < C, \gamma_i = 0, \quad (4.15)$$

$$\|z_i - u\|_2 > R \rightarrow \alpha_i = C, \gamma_i > 0. \quad (4.16)$$

The trace form of the Lagrangian given in 4.13 is as follow:

$$\begin{aligned}\mathcal{L} &= \text{tr} (S_Q^{-1} QXAX^T Q^T) - \text{tr} (S_Q^{-1} QX\alpha\alpha^T X^T Q^T) \\ &= \text{tr} ((QXL_X X^T Q^T)^{-1} QX(\mathbb{A} - \alpha\alpha^T) X^T Q^T\end{aligned}\quad (4.17)$$

The diagonal of the matrix $\mathbb{A}$ include α_i values and zeros elsewhere, note that $\mathbb{A} \in \mathbb{R}^{N \times N}$, the α represents a vector of the values of α_i . If the following matrices are defined:

$$L_\alpha = \mathbb{A} - \alpha\alpha^T \quad (4.18)$$

$$S_\alpha = XL_\alpha X^T, \quad (4.19)$$

Equation 4.17 can be simplified to the following form:

$$\mathcal{L} = \text{tr} ((QS_X Q^T)^{-1} QS_\alpha Q^T) \quad (4.20)$$

The equation 4.20 is constructed in similar form to the trace ration problem [96]. The objective can either be maximized or minimized if the projection vectors which is associated with the non-zero largest/smallest eigenvalues are preserved while resolving the general eigenproblem. Three resolutions updates can be offered by GESSVDD: Spectral updates, Spectral regression-based updates and Gradient-based updates. Each solution has the ability to either minimize the objective or maximizing it, meeting a broad domain of optimization goals. Also, it allows to adjust and enhance graph information accordingly. With these preferences, the suitable graph and update that matches the goals of the optimization and the corresponding dataset characteristics can be selected using the GESSVDD. Several reasonable options were experimented, knowing that $\mathbf{L}_X$ can be served using different graphs. The start was with utilizing the approach of k-Nearest Neighbor (kNN), the geometric information which is local were leveraged. The next options were with the employment of the PCA graph. Finally, the option of setting $\mathbf{L}_X$ to $\mathbf{I}$ was considered.

4.6 WORK IMPLEMENTATION

4.6.1 Preprocessing and the Datasets

The incorporated datasets in this work are listed in Table 4-1; the work employed three different datasets; all datasets were downloaded from Kaggle. The names of the datasets are

as follow: the first one is dataset_malwares [47], second dataset is Classification of Malware (CLaMP) [97] and the third dataset is pe_section_headers [98].

Table 4.1: Datasets Used in This Work.

Dataset Name	ID	Referen ce	Features	Samples	Classes	Target_Class
dataset_malwares	A	[47]	77	19611	2	Begnine
CLaMP	B	[97]	69	5184	2	Begnine
Pe_section_headers	C	[98]	4	43293	2	Begnine

A train set and a test set were created from each dataset as part of the pre-processing step. Thirty percent of the data was set aside for testing and seventy percent was used for training. The target class training data's mean and standard deviation were used to standardize the data. On the training set, a 5-fold cross-validation was used to maximize the model's performance. Following the identification of the ideal hyperparameters, a model was trained using the complete training set, and its performance was assessed using the test set.

4.6.2 Used Hyperparameters

Hyperparameters importance was to control the learning algorithms performance. This subsection will introduce and presents a brief discussion to the hyperparameters of the employed models in this work. As mentioned previously, this work has implemented OCSVM, SVDD, SSVDD, and GESSVDD. Each of the implemented models has its own hyperparameters, some hyperparameters are similar and can be found in more than one model in many cases.

The start will be with the C hyperparameter, this hyperparameter manages the trade-off between minimizing training errors and maximizing margin. The used values range of this hyperparameter in all cases that utilized it was [0.1, 0.2, 0.3, 0.4, 0.5].

For the non-linear mapping, the Non-linear Projection Trick (NPT) was utilized within the non-linear applications [99]. The NPT employment is similar to using the popular kernel trick, which makes it possible to apply the method's linear variant. The original D -dimensional dataset contained data X in NPT is mapped in the F -dimensional space to Φ . Now, emphasizing on the non-linear implementations, the distance between two datapoints can be scaled using the hyperparameter sigma (σ), this was in the non-linear implementations. The hyperparameter σ was used during kernel matrix computation. The range of sigma used in this work during the cross-fold validation was $[-1, 0, 1, 2, 3]$.

The d hyperparameter was used to represent the dimensionality of the data or the quantity of data used within the lower-dimensional space; $[1, 2, 3, 4, 5, 10, 20]$ was the range that the value of d was selected from.

The Eta (η) hyperparameter represents the learning rate, the range that the η was selected from was $[-1, 0, 1, 2, 3]$.

In SSVDD the hyperparameter Beta (β) was used to control regularization term importance [17][33], the β value in GESSVDD has a default value of 0.1.

In GESSVDD, the optimization process can embed various graph structures; Five (5) was chosen as the number of neighbors that define the kNN graph.

4.6.3 Metrics: Evaluating Classifier Performance

The following performance-evaluating metrics have been incorporated in the process of evaluating the classifiers performance: GMean, FMeasure, Precision, Specificity (TNR), Sensitivity (TPR) and Accuracy. A brief overview of each metric will be presented in the next subsection. It is important to make a clear introduction to the terminologies that may be used to define the metrics:

- a. True Positive (TP): Positive samples that the classifier has determined to be positive.
- b. True Negative (TN): Negative samples that the classifier has determined to be negative.
- c. False Positive (FP): Samples which are negative but were mistakenly classified as positive by the classifier.

d. False Negative (FN): Samples which are positive but were mistakenly classified as negative by the classifier.

e. P: All the samples that are positive, $P = TP + FN$.

f. N: All the samples that are Negative, $N = TN + FP$.

4.6.3.1 Accuracy

Correctly classified percentage of the samples.

$$\text{Accuracy} = \frac{TP+TN}{P+N} = \frac{TP+TN}{TP+FN+TN+FP} \quad (4.21)$$

4.6.3.2 Sensitivity

Sensitivity, True Positive (Recognition) Rate (TPR) or Recall, the proportion of correctly classified positive samples.

$$\text{TPR} = \frac{TP}{P} = \frac{TP}{TP+FN} \quad (4.22)$$

4.4.3.3 Specificity

True Negative Rate (TNR) or Specificity, this metric can measure the classification model ability to correctly recognize the negative instances.

$$\text{TNR} = \frac{TN}{N} = \frac{TN}{TN+FP} \quad (4.23)$$

4.6.3.4 Precision

The percentage of the positive samples that were identified as positive. This metric can measure the ratio of the correct positive predictions, it can be identified as the exactness measure.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (4.24)$$

4.6.3.5 FMeasure

FMeasure, F1Score or FScore, this metric combines both TPR or Sensitivity and Precision in a single equation. It is the harmonic mean calculation of those two metrics, Sensitivity and Precision. High FMeasure means that both Precision and Sensitivity are high. Machine learning and data science both use this metric as an estimation metric to the classification models performance.

$$\text{FMeasure} = \frac{2 * \text{Precision} * \text{TPR}}{\text{Precision} + \text{TPR}} \quad (4.25)$$

4.6.3.6 Gmean

The Gmean or the Geometric-Mean, this metric measures the scores of TPR (Sensitivity) and TNR (Specificity) geometric mean. Where TPR stands for the positive class prediction, whereas TNR stands for the negative class prediction.

$$\text{GMean} = \sqrt{\text{TPR} * \text{TNR}} = \sqrt{\text{Sensitivity} * \text{Specificity}} \quad (4.26)$$

4.7 EXPERIMENTAL RESULTS DISCUSSION

Tables 4-2 to 4-7 provide detailed results of the experiments that were carried out, these tables include the results for each model, knowing that both the non-linear and the linear variants were taken into consideration. The evaluates models were OCSVM, SVDD, SSVDD and GESSVDD. Tables 4-2 to 4-7 presents a detailed insights into each model's performance, displaying the effectiveness of these models under various settings and conditions. The Tables 4-2 to 4-7 presents the achieved performance by each model in a comprehensive manner using the aforementioned metrics. Several graphs' structures have been utilized in GESSVDD in order to enhance its performance. To increase the model's accuracy, the kNN graph was used in the first place. Using this method, when making predictions, the neighborhood's information can be taken into consideration. Also, the GESSVDD algorithm has incorporated the PCA. Another addition was done to the GESSVDD formulation, the use the identity matrix $\mathbf{I}$ instead of $\mathbf{L}_x$. In certain scenarios, this adjustment or modification can result in better results.

Regarding Dataset A (see Table 4-1), Table 4-2 displays the outcomes for the linear model, while Table 4-3 presents the results of non-linear model, highlighting the fact that Tables 4-2 and 4-3 show the results for dataset A.

Results shown in Table 4-2 are the linear results, Maximum value of TPR has been achieved in many OCC models, this indicates the ability of identifying accurately the positive instances. Contrariwise, the highest TNR was demonstrated in OCSVM, this reflects the effectiveness of OCSVM in classifying the negative instances correctly. Add to that OCSVM has obtained the highest Precision, this means that the positive samples can be classified precisely. The OCSVM has attained the highest Gmean, GESSVDD-min-Spectral-PCA attained the best FMeasure and outperformed the other models. These results collectively draw attention to the superior performance of OCSVM for dataset A within the linear case, specifically in terms of overall sensitivity and accuracy.

Table 4-3 displays the dataset A's non-linear outcomes, these results have revealed that the highest Sensitivity was achieved in GESSVDD-min-Spectral-PCA, add to that, the Specificity attained a value of 1.00 in both GESSVDD-max-Spectral-kNN and GESSVDD-min-gradient-PCA. Also, it is crucial to note that Precision reached a value of 1.00 in GESSVDD-min-gradient-PCA. The highest FMeasure and Gmean were obtained in OCSVM, this reflects the effectiveness of OCSVM in creating a balance between Sensitivity and Precision regarding dataset A.

Moving to dataset B, the linear case, the results are presented in Table 4-4, superiority in performance went to GESSVDD, As GESSVDD outperformed the rest of the methods. Most of GESSVDD variants attained a value of 1.00 in terms of Sensitivity. Add to that SVDD and all the SSVDD variants have performed very well with regard to high TPR score. GESSVDD-max-EIG-kNN attained the highest Precision. Add to that, the highest FMeasure value obtained was 0.70, five variants of GESSVDD had obtained this highest FMeasure value. While the highest Gmean value was attained by GESSVDD-min-Spectral-kNN.

The results of the non-linear case of the dataset B are presented in Table 4-5, the subspace learning model has outperformed the other models with regard to FMeasure and Sensitivity. A 1.00 TPR score has been attained in SSVDD ψ_2 , GESSVDD-min-gradient-I and GESSVDD-max-gradient-I. A TNR score of 1.00 has been achieved by both SVDD and

GESSVDD-min-EIG-kNN. While the highest FMeasure value was achieved by GESSVDD-max-gradient-PCA and GESSVDD-min-EIG-PCA. Also, the highest Precision and Gmean have been achieved by SVDD.

Table 4-6 displays the dataset C's linear case results, the findings indicated that the subspace learning methods represented by SSVDD and GESSVDD outperformed the results of other methods across different performance metrics. Almost all the used metrics like Sensitivity, Specificity, Precision, FMeasure and GMean have attained the highest results. Sensitivity in majority of the GESSVDD methods along with SSVDD ψ_0 and SSVDD ψ_1 achieved a value of 1.00. The highest Specificity score was attained using GESSVDD-max-EIG-kNN, this indicated the effectiveness of this method in identifying the negative instances. While the highest Precision and GMean were obtained using GESSVDD-max-Spectral-kNN method. It was notable that the FMeasure value was the highest in SSVDD ψ_2 , this indicates that the combination of the Sensitivity and Precision was balanced.

The Non-linear case results for dataset C are presented in Table 4-7, in this situation the methods which employing the subspace learning have outperformed the other techniques. The Sensitivity has attained a value of 0.99 in both GESSVDD-min-Spectral-kNN and GESSVDD-max-Spectral-kNN. This indicated their effectiveness in positive instances identification. Add to that, the Specificity values was 1.00 in SVDD, GESSVDD-max-gradient-kNN and GESSVDD-max-gradient-I, this indicated their ability to classify the negative class instances correctly. Furthermore, a precision value of 1.00 was attained in GESSVDD-max-gradient-I. Finally, the highest values of FMeasure and GMean were attained with SSVDD ψ_2 , this indicates the exceptional balance between both Sensitivity and Precision also the overall performance. It should be noted that the outcomes obtained for dataset C for the non-linear case show that the subspace learning methods perform better than other methods.

Table 4.2: Linear Results for Dataset A, Dataset_Malwares.

Linear	Accuracy	TPR	TNR	Precision	FMeasure	GMean
OCSVM	0.68	0.36	0.79	0.37	0.37	0.53
SVDD	0.36	1.00	0.14	0.29	0.44	0.38
SSVDD ψ_0	0.35	1.00	0.13	0.28	0.44	0.37
SSVDD ψ_1	0.35	0.99	0.13	0.28	0.44	0.37
SSVDD ψ_2	0.35	1.00	0.13	0.28	0.44	0.37
SSVDD ψ_3	0.35	1.00	0.13	0.28	0.44	0.37
GESSVDD-min-gradient-kNN	0.37	0.60	0.29	0.23	0.33	0.42
GESSVDD-max-gradient-kNN	0.37	0.60	0.29	0.23	0.33	0.42
GESSVDD-min-EIG-kNN	0.46	0.55	0.42	0.25	0.34	0.48
GESSVDD-max-EIG-kNN	0.44	0.39	0.46	0.20	0.27	0.43
GESSVDD-min-Spectral-kNN	0.57	0.23	0.69	0.20	0.21	0.40
GESSVDD-max-Spectral-kNN	0.56	0.32	0.64	0.23	0.27	0.45
GESSVDD-min-gradient-PCA	0.35	1.00	0.13	0.28	0.44	0.36
GESSVDD-max-gradient-PCA	0.35	1.00	0.13	0.28	0.44	0.37
GESSVDD-min-EIG-PCA	0.35	1.00	0.12	0.28	0.44	0.35
GESSVDD-max-EIG-PCA	0.35	1.00	0.12	0.28	0.44	0.35
GESSVDD-min-Spectral-PCA	0.39	1.00	0.19	0.30	0.46	0.43
GESSVDD-max-Spectral-PCA	0.36	0.99	0.14	0.28	0.44	0.38
GESSVDD-min-gradient-I	0.35	1.00	0.13	0.28	0.44	0.36
GESSVDD-max-gradient-I	0.36	1.00	0.14	0.28	0.44	0.37
GESSVDD-min-EIG-I	0.35	1.00	0.13	0.28	0.44	0.36
GESSVDD-max-EIG-I	0.35	1.00	0.13	0.28	0.44	0.36
GESSVDD-min-Spectral-I	0.36	0.99	0.15	0.29	0.44	0.38
GESSVDD-max-Spectral-I	0.36	1.00	0.15	0.29	0.44	0.38

Table 4.3: Non-Linear Results for Dataset A, Dataset_Malwares.

Non-Linear	Accuracy	TPR	TNR	Precision	FMeasure	GMean
OCSVM	0.86	0.47	1.00	0.99	0.64	0.68
SVDD	0.77	0.19	0.97	0.70	0.30	0.43
SSVDD ψ_0	0.48	0.68	0.41	0.28	0.40	0.53
SSVDD ψ_1	0.36	0.61	0.27	0.22	0.33	0.41
SSVDD ψ_2	0.32	0.59	0.22	0.21	0.31	0.37
SSVDD ψ_3	0.48	0.68	0.41	0.28	0.40	0.53
GESSVDD-min-gradient-kNN	0.80	0.39	0.94	0.70	0.51	0.61
GESSVDD-max-gradient-kNN	0.29	0.78	0.12	0.23	0.36	0.31
GESSVDD-min-EIG-kNN	0.65	0.03	0.86	0.07	0.04	0.17
GESSVDD-max-EIG-kNN	0.65	0.03	0.86	0.07	0.04	0.17
GESSVDD-min-Spectral-kNN	0.54	0.63	0.51	0.31	0.41	0.57
GESSVDD-max-Spectral-kNN	0.75	0.01	1.00	0.67	0.02	0.10
GESSVDD-min-gradient-PCA	0.75	0.01	1.00	1.00	0.02	0.10
GESSVDD-max-gradient-PCA	0.68	0.01	0.92	0.03	0.01	0.09
GESSVDD-min-EIG-PCA	0.62	0.01	0.83	0.01	0.01	0.08
GESSVDD-max-EIG-PCA	0.61	0.01	0.82	0.01	0.01	0.08
GESSVDD-min-Spectral-PCA	0.29	0.99	0.05	0.26	0.41	0.21
GESSVDD-max-Spectral-PCA	0.66	0.00	0.89	0.00	0.00	0.03
GESSVDD-min-gradient-I	0.60	0.80	0.54	0.37	0.51	0.66
GESSVDD-max-gradient-I	0.43	0.76	0.31	0.28	0.40	0.49
GESSVDD-min-EIG-I	0.35	0.97	0.13	0.28	0.43	0.36
GESSVDD-max-EIG-I	0.71	0.01	0.96	0.06	0.01	0.09
GESSVDD-min-Spectral-I	0.36	0.09	0.46	0.05	0.07	0.20
GESSVDD-max-Spectral-I	0.71	0.00	0.95	0.00	0.00	0.00

Table 4.4: Linear Results for Dataset B, Clamp.

Linear	Accuracy	TPR	TNR	Precision	FMeasure	GMean
OCSVM	0.23	0.22	0.24	0.21	0.21	0.23
SVDD	0.58	1.00	0.19	0.53	0.69	0.44
SSVDD ψ_0	0.56	1.00	0.17	0.52	0.69	0.41
SSVDD ψ_1	0.56	1.00	0.17	0.52	0.69	0.41
SSVDD ψ_2	0.57	0.99	0.19	0.53	0.69	0.44
SSVDD ψ_3	0.56	1.00	0.17	0.52	0.69	0.41
GESSVDD-min-gradient-kNN	0.57	0.99	0.18	0.53	0.69	0.43
GESSVDD-max-gradient-kNN	0.50	0.57	0.43	0.48	0.52	0.50
GESSVDD-min-EIG-kNN	0.48	1.00	0.01	0.48	0.65	0.11
GESSVDD-max-EIG-kNN	0.61	0.32	0.88	0.71	0.44	0.53
GESSVDD-min-Spectral-kNN	0.62	0.84	0.42	0.57	0.68	0.59
GESSVDD-max-Spectral-kNN	0.56	0.22	0.88	0.62	0.32	0.44
GESSVDD-min-gradient-PCA	0.57	1.00	0.17	0.52	0.69	0.41
GESSVDD-max-gradient-PCA	0.56	1.00	0.17	0.52	0.69	0.41
GESSVDD-min-EIG-PCA	0.56	1.00	0.16	0.52	0.68	0.40
GESSVDD-max-EIG-PCA	0.58	1.00	0.20	0.53	0.69	0.44
GESSVDD-min-Spectral-PCA	0.59	1.00	0.21	0.54	0.70	0.46
GESSVDD-max-Spectral-PCA	0.59	1.00	0.21	0.54	0.70	0.46
GESSVDD-min-gradient-I	0.58	0.99	0.20	0.53	0.69	0.44
GESSVDD-max-gradient-I	0.57	1.00	0.18	0.53	0.69	0.42
GESSVDD-min-EIG-I	0.57	1.00	0.18	0.53	0.69	0.43
GESSVDD-max-EIG-I	0.58	1.00	0.20	0.53	0.70	0.45
GESSVDD-min-Spectral-I	0.59	1.00	0.21	0.54	0.70	0.46
GESSVDD-max-Spectral-I	0.59	1.00	0.21	0.54	0.70	0.46

Table 4.5: Non-Linear Results for Dataset B, CLaMP.

Non-Linear	Accuracy	TPR	TNR	Precision	FMeasure	GMean
OCSVM	0.69	0.51	0.85	0.76	0.61	0.66
SVDD	0.74	0.47	1.00	0.99	0.64	0.68
SSVDD ψ_0	0.51	0.50	0.53	0.49	0.49	0.51
SSVDD ψ_1	0.59	0.92	0.28	0.54	0.68	0.51
SSVDD ψ_2	0.56	1.00	0.16	0.52	0.68	0.40
SSVDD ψ_3	0.51	0.50	0.53	0.49	0.49	0.51
GESSVDD-min-gradient-kNN	0.56	0.53	0.59	0.54	0.54	0.56
GESSVDD-max-gradient-kNN	0.29	0.17	0.39	0.21	0.19	0.26
GESSVDD-min-EIG-kNN	0.52	0.00	1.00	0.67	0.01	0.05
GESSVDD-max-EIG-kNN	0.39	0.20	0.56	0.29	0.24	0.33
GESSVDD-min-Spectral-kNN	0.42	0.01	0.80	0.04	0.02	0.09
GESSVDD-max-Spectral-kNN	0.47	0.01	0.89	0.05	0.01	0.08
GESSVDD-min-gradient-PCA	0.54	0.97	0.14	0.51	0.67	0.37
GESSVDD-max-gradient-PCA	0.59	0.98	0.24	0.54	0.70	0.48
GESSVDD-min-EIG-PCA	0.62	0.91	0.36	0.57	0.70	0.57
GESSVDD-max-EIG-PCA	0.56	0.18	0.91	0.64	0.28	0.40
GESSVDD-min-Spectral-PCA	0.51	0.00	0.98	0.00	0.00	0.00
GESSVDD-max-Spectral-PCA	0.58	0.82	0.35	0.54	0.65	0.54
GESSVDD-min-gradient-I	0.48	1.00	0.01	0.48	0.65	0.08
GESSVDD-max-gradient-I	0.56	1.00	0.16	0.52	0.68	0.40
GESSVDD-min-EIG-I	0.43	0.15	0.69	0.30	0.20	0.32
GESSVDD-max-EIG-I	0.43	0.15	0.69	0.30	0.20	0.32
GESSVDD-min-Spectral-I	0.51	0.00	0.98	0.00	0.00	0.00
GESSVDD-max-Spectral-I	0.58	0.82	0.35	0.54	0.65	0.54

Table 4.6: Linear Results For Dataset C, Pe_Section_Headers.

Linear	Accuracy	TPR	TNR	Precision	FMeasure	GMean
OCSVM	0.52	0.48	0.55	0.49	0.49	0.51
SVDD	0.57	0.99	0.19	0.53	0.69	0.43
SSVDD ψ_0	0.54	1.00	0.12	0.51	0.67	0.34
SSVDD ψ_1	0.54	1.00	0.12	0.51	0.67	0.35
SSVDD ψ_2	0.59	0.99	0.23	0.54	0.70	0.48
SSVDD ψ_3	0.55	0.99	0.14	0.51	0.68	0.38
GESSVDD-min-gradient-kNN	0.52	0.28	0.74	0.49	0.36	0.45
GESSVDD-max-gradient-kNN	0.52	0.35	0.68	0.50	0.41	0.49
GESSVDD-min-EIG-kNN	0.41	0.23	0.58	0.33	0.27	0.37
GESSVDD-max-EIG-kNN	0.51	0.11	0.87	0.43	0.18	0.31
GESSVDD-min-Spectral-kNN	0.56	0.60	0.51	0.53	0.56	0.56
GESSVDD-max-Spectral-kNN	0.62	0.63	0.61	0.59	0.61	0.62
GESSVDD-min-gradient-PCA	0.55	1.00	0.14	0.51	0.68	0.37
GESSVDD-max-gradient-PCA	0.56	1.00	0.16	0.52	0.68	0.40
GESSVDD-min-EIG-PCA	0.51	1.00	0.06	0.49	0.66	0.23
GESSVDD-max-EIG-PCA	0.51	1.00	0.06	0.49	0.66	0.24
GESSVDD-min-Spectral-PCA	0.57	0.99	0.19	0.53	0.69	0.43
GESSVDD-max-Spectral-PCA	0.51	0.99	0.07	0.49	0.66	0.27
GESSVDD-min-gradient-I	0.56	0.99	0.18	0.52	0.68	0.42
GESSVDD-max-gradient-I	0.55	1.00	0.14	0.51	0.68	0.37
GESSVDD-min-EIG-I	0.52	1.00	0.08	0.50	0.66	0.28
GESSVDD-max-EIG-I	0.55	1.00	0.15	0.52	0.68	0.38
GESSVDD-min-Spectral-I	0.57	0.99	0.18	0.53	0.69	0.43
GESSVDD-max-Spectral-I	0.58	1.00	0.20	0.53	0.69	0.44

Table 4.7: Non-Linear Results for Dataset C, Pe_Section_Headers.

Non-Linear	Accuracy	TPR	TNR	Precision	FMeasure	GMean
OCSVM	0.70	0.54	0.84	0.76	0.63	0.67
SVDD	0.72	0.41	1.00	0.99	0.58	0.64
SSVDD ψ_0	0.52	0.28	0.73	0.49	0.36	0.45
SSVDD ψ_1	0.52	0.28	0.73	0.49	0.36	0.45
SSVDD ψ_2	0.68	0.76	0.60	0.63	0.69	0.68
SSVDD ψ_3	0.67	0.36	0.96	0.88	0.51	0.59
GESSVDD-min-gradient-kNN	0.43	0.79	0.11	0.45	0.57	0.29
GESSVDD-max-gradient-kNN	0.58	0.14	1.00	0.96	0.24	0.37
GESSVDD-min-EIG-kNN	0.33	0.28	0.38	0.29	0.28	0.32
GESSVDD-max-EIG-kNN	0.33	0.28	0.38	0.29	0.28	0.32
GESSVDD-min-Spectral-kNN	0.53	0.99	0.12	0.51	0.67	0.34
GESSVDD-max-Spectral-kNN	0.53	0.99	0.12	0.51	0.67	0.34
GESSVDD-min-gradient-PCA	0.52	0.98	0.10	0.50	0.66	0.32
GESSVDD-max-gradient-PCA	0.39	0.06	0.70	0.15	0.08	0.20
GESSVDD-min-EIG-PCA	0.52	0.99	0.08	0.50	0.66	0.28
GESSVDD-max-EIG-PCA	0.53	0.19	0.84	0.52	0.28	0.40
GESSVDD-min-Spectral-PCA	0.65	0.29	0.97	0.91	0.44	0.53
GESSVDD-max-Spectral-PCA	0.65	0.29	0.97	0.91	0.44	0.53
GESSVDD-min-gradient-I	0.37	0.25	0.47	0.31	0.28	0.35
GESSVDD-max-gradient-I	0.52	0.00	1.00	1.00	0.01	0.06
GESSVDD-min-EIG-I	0.55	0.15	0.91	0.60	0.24	0.37
GESSVDD-max-EIG-I	0.55	0.09	0.97	0.72	0.16	0.30
GESSVDD-min-Spectral-I	0.64	0.56	0.71	0.64	0.60	0.63
GESSVDD-max-Spectral-I	0.64	0.56	0.71	0.64	0.60	0.63

In general, in both linear and non-linear scenarios, the techniques which employs the subspace learning have outperformed other techniques which employed the traditional OCC. When comparing the outcomes of the linear and non-linear cases, it is evident that the TPR in the linear cases overwhelmingly approaches the 1.00 value, this indicates that the positive instances can be identified in high accuracy. Moving to the non-linear data description cases, it is noted that Specificity in this case attained higher values and a TNR score of 1.00 in some cases, this indicates that the false positive rate is low. The non-linear situations attained a higher value of Precision in comparison to linear scenarios. The Precision values in the non-linear cases were nearer to the maximum. Whereas in general, the attained Precision values in the linear cases were lower. For the FMeasure results, this metric demonstrated variability, sometimes the outcomes of the linear and the non-linear cases were similar but at other times the results were approximate. Finally, for the GMean metric, it was observed that the non-linear scenarios attained the highest GMean results consistently.

4.8 SUMMARY AND CONCLUSIONS OF THE WORK

Recently, many researchers have focused on incorporating AI techniques in security applications, one of these applications was the malware detection in portable executable files, the Windows operating system usually use these types of files. PE files have the potential to contain malicious code, this malicious program can play the role of harming the computer. Several previous implementations have employed different AI-based techniques to detect, identify and prevent such type of malicious code to affect or cause harm to the system. Despite the progress that these previous implementations have achieved, and although these techniques have proven to be effective in detecting malicious behavior in portable executable files and some of them have become a widely used approach in detecting malware, there is always a plenty room for making some refinements, improvements and adding new algorithms in this research area.

This work has evaluated the subspace learning-based OCC methods effectiveness in detecting malicious codes using the header features of the portable executable files. It is known that the traditional one-class classification techniques like OCSVM and SVDD suffer from curse of dimensionality problem, anyway, for comparison reasons OCSVM and SVDD were incorporated also in this work. This work has examined in particular the SSVDD and GESSVDD performance and implementation as subspace learning-based OCC methods. The non-linear and linear versions were both tested of both SSVDD and GESSVDD, a comprehensive documentation to the results were done.

The results attained by using the subspace learning techniques represented by the SSVDD and GESSVDD which was intended to reveal malware, have showed their potential as a worthy tool in cybersecurity field. Where these techniques performance have outperformed the performance of other OCC traditional techniques, these techniques have proven their effectiveness in fighting malware through detecting them. Several SSVDD implementations have been carried out, compared to other variations, SSVDD ψ_2 attained the superior performance. Moving to GESSVDD, for malware detection, kNN graph is recommended. As for the future work and in order to enhance malware detection system robustness and accuracy we plane to investigate the ensemble approaches implementation.

5. SUMMARY AND CONCLUSIONS

5.1 SUMMARY

Cybersecurity a terminology that can be given to any technology, practice or measure that aims to prevent any kind of cyberattacks, or at least mitigate the impact of the cyberattack. The main aim of cybersecurity is to protect the organizations and individuals' assets from the malicious codes, viruses and any other type of attack such as costly ransomware attacks.

Several implementations worked on different fields that prevent various types of cyberattacks, malware was one of these widely spread attacks that affect the system in different ways, for example, the malware can be the code that the attacker embed in the system to spy on that system and send some reports about the system behaviour, or it can malfunction the system in many ways, or be the gate that the attacker can access the system and do harmful things. The beginning of the malware was before the spread of networking and the Internet, at that time the malware uses the floppy disk or any removable disk to spread, However, as networking and the Internet proliferated, malware developed the capacity to spread more widely throughout the network or even outside of it. In all cases the malicious people who write and evolve the malicious code always try to find a backdoor or any vulnerability through which they can have an access to the system to implant the malware or in other cases they can send the malware through malicious attachment or picture that the victim will open and this malware will have an access to the system.

In Windows operating system, portable executable file's structure represents the ideal environment that the malware can exploit and hide inside, if the system uses a signature-based security system, the process of detecting the malware will be a hard task, this is due to updating the security program, or if the malware is new and unknown to security applications like the Zero-day attack. While in case of anomaly-based security system, the algorithm may misclassify the file as a malware one due to some problems.

This work has investigated different strategies and algorithms in a trial to reach an optimized algorithm. The investigation of this work was divided into two basic phases, the traditional machine learning algorithms phase and the Subspace learning phase. It was important to investigate both phases for the following reasons:

- a. If the aim is to find an optimized solution for the malware detection problem, the correct start must be as close to scratch as possible, this will help the researcher to explore all the important milestones and to not miss any point that may affect the whole findings.
- b. Results comparisons, this is a very important issue that must be taken into consideration, comparing the results will prove the effectiveness of the developed algorithm.
- c. Covering all the wholes and mistakes that previous algorithms fallen into them in any phase. It is so important to take all the drawbacks and problems of the previously implemented problems into consideration, and such problems must be bypassed and a solution must be created to each single problem.
- d. Going through various implementation will enlighten the researcher's awareness and expand his horizons. This enables the researcher to catch glitches that may appear in the innovative new algorithm early during the testing and experimenting phase.

The start was with stage one, traditional machine learning algorithms, the aim was to investigate their power in recognizing malicious portable executable files and focus on the problems that these algorithms may have in order to put the suitable solutions in the future. The algorithm selection was not random, on the contrary, an accurate testing and experimenting have been carried out to various algorithms ranged between classification and clustering algorithms, but the classification algorithms got the higher percentage of testing especially after recommendations from more than one reference that classification can work better in the process of malware detection. The selected classification algorithms were NN_MLPC and DT. After algorithm testing and carrying out the selection process to the algorithms with best performance and good level of recommendations from previous implementations, two main parts were done to finalize this stage of research:

- a. The first part was done by collecting a set of Twenty-Nine features, those features represent part of the features that the Portable executable header file includes. The selection of those particular features was based on the recommendation of previous implementations done by various researchers. The primary goal of the work was to investigate the impact of adding features in obtaining a better accuracy in the process

of discrimination between malware and benign portable executable files. One of the aims was to elect a subset of features taken from that set so that it can be said that those features incorporation will always lead to accurately recognize malicious files and attains the highest accuracy, but for this particular aim there was a need to widen the study to reach a reasonable result and try different strategy in selecting the features from the original set of features, this was the first reason that leads to extend the work to another wider part. The used strategy in this part (part one) was to start with certain collection of features and do all the test runs and record the results, then single feature was added progressively and the test runs were done again and results were recorded. This was important to see the impact of each feature when added to the set, also this showed that not all the features have a positive impact on accuracy, between the outcomes that this part included was the number of features used, it was notable that incorporating a large number of features will not necessarily lead to high accuracy or improved the accuracy. Including the graphs in showing the results obtained was so useful in showing when the accuracy attains an improvement or level decrease. More than 100 runs had been carried out. NN_MLPC and DT algorithms each with two options of test_size and Random_state were selected to implement all the tests, selecting two different algorithms was to study their effectiveness and for comparisons purposes. A specified control dataset was selected to carry out all the tests and assessing the performance. The accuracy metric was selected to be the gauge with which the performance will be evaluated this was done with different and large number of groups of studies. The highest accuracy obtained in NN_MLPC was 0.979 and for DT was 0.987. Several observations were also resulted from implementing this part, these observations were the main guidelines that have been used in part two to figure out new outcomes and attain better results, fortunately this is what have happened.

- b. The observations and outcomes that the researcher obtained from the implementation of the first part made it necessary for the researcher to carry out complementary work in the second part, but this time the testing and the experiments was on a larger scale and using strategies which was based on the outcomes and observations of the first part so that the largest number of features combinations can participate, as including all the possibilities of combining the twenty-nine feature in all possible combination

will cost a very long time and take an endless possibilities. In the second part of phase one the same Twenty-Nine features set and same selected algorithms in part one has been employed, about 147 different runs in total were implemented, selecting the same features set and algorithms came from the fact that previous observations in part one required more and wider exploration, contemplation and additional experiments in order to translate these observations to fact.

The strategy of this part (part two of stage one) was based on the method of subgrouping features, note that in part one the features were added progressively one by one and accuracy was recorded with each run. There were multiple questions in part two that required answers. Some of the questions included: which feature will improve accuracy when added? Is it possible to select a trustworthy feature set that can distinguish between a malicious and a benign portable executable file? Will the features' combination have an impact on malware detection's accuracy? This work implemented **Seventy-Seven** different runs using NN_MLPC and **Seventy** runs on DT. The number of runs vary between the NN_MLPC and DT this is due to the variation in observations that each algorithm had produced in the first part, those observations as mentioned previously were the guide to select the number of features and which features will participate in each run. The attained results in this part were improved and outperformed the obtained results of part one, numerically, in terms of accuracy, the obtained results were 0.979 and 0.98 for NN_MLPC and 0.983 and 0.986 for DT. Practically, the results was the obtained answers on the raised questions, where the total amount of features used and the features that are involved in the run are thought to be significant factors influencing the accuracy level, add to that feature combination also has a crucial role. The experiments had shown that incorporating high number of features will not necessarily result in high accuracy level, on the contrary, the curse of dimensionality problem will arise if the number of features increases. So, used set of features must include the correct parameters starting from how many features there are, features with a positive impact and the combination of the applied features. Another outcome is that each algorithm may behave differently with the same features set that produced the highest accuracy levels, this was clear when both NN_MLPC and DT were implemented same experiments on same set of features each of them attained different level of accuracy.

The next two stages were to move gradually to a promising school of classification algorithms, the school of Subspace learning, actually this algorithm can bypass many of the problems that datasets and other algorithms may suffer from. The start was with stage two, some traditional One-Class Classification (OCC) algorithms, OCSVM and SVDD, that can work efficiently when there is a shortage in one of the classes or in other words under the conditions of imbalance problem, in this implementation the malware samples suffer from being significantly smaller than the benign samples, another issue that needed to be taken into consideration regarding malware is that the malware nature is dynamic and evolving, so, this makes the malware incorporate different characteristics, and this will be considered as another problem that makes the detection algorithm fail in detecting them as malware and consider them as benign which may lead to security vulnerability. Hence, one-class classification came as a promising algorithm which can detect malware through using only the well-known benign class and classify any sample which differs from the target class as malware.

The traditional one-class classification solved more than one problem that the traditional AI algorithms cannot bypass, but unfortunately, there is another problem that needed to be bypassed in order to reach the optimized promising algorithm in malware detection, the Curse of Dimensionality problem; traditional OCC suffer from this problem which may affect the detection accuracy. Solving this problem can be done using the novel subspace learning techniques which were proposed by [7]. Incorporating this technique in stage three can be the key to find its purpose in achieving the optimized technique. Accordingly, in stage three, the work had adopted the use of subspace learning-based OCC methods, to get around the curse of dimensionality issue in addition to handle the other problems like the problem of class imbalance. The work has translated this adoption into a pipeline whose goal is to detect malware by employing methods that will simultaneously optimize both the subspace and the data description for the OCC. Three different datasets were employed in this study to assess the efficiency of different OCC. Traditional OCC techniques used were OCSVM and SVDD while the subspace-learning-based data description techniques used were SSVDD and GESSVDD, results obtained from this work showed that the subspace-learning-based data description techniques has outperformed the traditional OCC in malware detection.

5.2 CONCLUSIONS

This dissertation has adopted employing AI techniques in malware detection, the case study used in this dissertation was the portable executable files, the researcher has implemented different traditional AI techniques in studying this type of file and tried to obtain the best accuracy in malware detection, then the researcher moved to other technique which is the One-Class Classification, and tried two different variants OCSVM and SVDD using the LibSVM library, then to bypass all the problems that all the implemented types have, the subspace-learning technique has been used and the particular techniques used were SSVDD and GESSVDD.

Each time the researcher implements a technique, results were satisfying and fortunately the next adoption attains an enhanced result. In the traditional AI when NN_MLPC and DT were incorporated after testing the results attained were leading results in comparison with previous implementations which used the same algorithms, mainly DT, but still the doubts were there, Accordingly, the researcher tried an algorithm that bypass this problem, since the main target of the dissertation was to develop a promising algorithm which has as low number of problems as possible, the algorithm was the OCC variants OCSVM and SVDD, in many cases the results have outperformed other obtained results, but again curse of dimensionality was the barrier between the researcher and the satisfaction with these results, so, after searching and studying, the subspace learning techniques were the promising savior that gave the researcher what he was looking for.

Results obtained when using the subspace learning technique represented by SSVDD and GESSVDD was as follow, when using SSVDD the SSVDD ψ_2 attained the best performance, and for GESSVDD the recommended version is the kNN graph. Actually, the obtained results reflected the potential of SSVDD and GESSVDD as a vital tool in the cybersecurity domain where performance of these techniques has outperformed traditional OCC and traditional AI techniques. It is highly recommended by the researcher that the proposed pipeline can be a worthy tool that can improve the computer and network security, where these tools can be used to accurately detect the malware the protecting the system from potential attacks.

REFERENCES

- [1] N Ahmed, U Kulsum, IB Azad, ASZ Momtaz, ME Haque, MS Rahman, “Cybersecurity Awareness Survey: An Analysis from Bangladesh Perspective,” 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), 2017.
- [2] Tirumala, SS and Valluri, Maheswara Rao and Babu, GA, “A survey on cybersecurity awareness concerns, practices and conceptual measures,” 2019 International Conference on Computer Communication and Informatics (ICCCI), Pages 1-6, IEEE, 2019.
- [3] Faruk, Md Jobair Hossain and Tahora, Sharaban and Tasnim, Masrura and Shahriar, Hossain and Sakib, Nazmus, “A review of quantum cybersecurity: threats, risks and opportunities,” 2022 1st International Conference on AI in Cybersecurity (ICAIC), pages 1-8, IEEE, 2022.
- [4] Elshweikh, Ahmed A and Mattar, Ahmed M and Hussein, Mohamed and Elbayoumy, Ashraf D, “Literature Survey for Cybersecurity for Internet of Things (IoT),” 2022 International Telecommunications Conference (ITC-Egypt), pages 1-5, IEEE, 2022.
- [5] H.Y. Ali and W. El-Medany, Wael, “IoT security: A review of cybersecurity architecture and layers,” 2nd Smart Cities Symposium (SCS 2019), page (7 pp.), IET Digital Library, 2019.
- [6] Ali Maki Al-Aali, Abdulla Al-Ateeq, Wael Elmedany, “Cybersecurity Threats and Solutions of IoT Network Layer,” 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), pages 250-257, IEEE, 2022.
- [7] Venkatasubramanian, Madumitha and Lashkari, Arash Habibi and Hakak, Saqib, “IoT Malware Analysis using Federated Learning: A Comprehensive Survey,” IEEE Access, Volume: 11, Pages: 5004 – 5018, DOI: 10.1109/ACCESS.2023.3235389. Date of Publication: 09 January 2023.

- [8] Chen, Chin-Wei and Su, Ching-Hung and Lee, Kun-Wei and Bair, Ping-Hao, “Malware family classification using active learning by learning,” 2020 22nd International Conference on Advanced Communication Technology (ICACT), pages 590-595, DOI: 10.23919/ICACT48636.2020.9061419, IEEE, 2020.
- [9] Manojkumar T Kamble and Sridevi, “Feature Extraction and Analysis of Portable Executable Malicious File,” 2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA), pages 1-6, IEEE, Gunupur-India, 2022.
- [10] Deylami, Hanif Mohaddes and Muniyandi, Ravie Chandren and Ardekani, Iman Tabatabaei and Sarrafzadeh, Abdolhossein, “Taxonomy of malware detection techniques: A systematic literature review,” 2016 14th Annual Conference on Privacy, Security and Trust (PST), pages 629--636, IEEE, 2016.
- [11] Fahad Mira, “A Systematic Literature Review on Malware Analysis,” 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), pages 1-5, IEEE, 2021.
- [12] Manoj Sai, Aakansha Tyagi, Kajal Panda, Sanjeev Kumar, “Machine learning-based malware detection using stacking of opcodes and bytecode sequences,” 2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC), pages 204-209, IEEE, 2022.
- [13] Alessandro Parisi. “Hands-On Artificial Intelligence for Cybersecurity. Implement smart AI systems for preventing cyber-attacks and detecting threats and network anomalies,” Published by: Packt Publishing. August 2019.
- [14] Bernhard Scholkopf, Robert C. Williamson, Alex Smolax and John Shawe-Taylor, “SV estimation of a distribution’s support,” Adv. Neural Inf. Process. Syst, volume: 41, pages: 582—588, publisher: Citeseer, 2000.
- [15] David M. J. Tax and Robert P.W. Duin, “Support vector data description,” Machine Learning Journal, Volume 54, pages 45–66, 2004 Kluwer Academic Publishers. Manufactured in The Netherlands, 2004.

- [16] Chih-Chung Chang and Chih-Jen Lin, "LIBSVM: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, volume 2, number 3, pages 1-27, Publisher: Acn New York, NY, USA, 2011.
- [17] F. Sohrab, J. Raitoharju, M. Gabbouj, and A. Iosifidis, "Subspace support vector data description," in *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 722–727, 2018.
- [18] F. Sohrab, A. Iosifidis, M. Gabbouj, and J. Raitoharju, "Graph-embedded subspace support vector data description," *Pattern Recognition*, vol. 133, p. 108999, 2023.
- [19] Akram M. Radwan, "Machine learning techniques to detect maliciousness of portable executable files," in *Int. Conf. on Promising Electronic Technologies (ICPET)*, Gaza, Palestine, pp. 86–90, IEEE, 2019.
- [20] Yesol Kim, Jeongoh Moon, Seong Je Cho, Minkyu Park, Sangchul Han, "Efficient Identification of Windows Executable Programs to Prevent Software Piracy," *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages: 236-240, IEEE, 2014.
- [21] Nagababu Pachhala, S. Jothilakshmi, Bhanu Prakash Battula, "A comprehensive survey on identification of malware types and malware classification using machine learning techniques," *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, pages: 1207-1214, IEEE, 2021.
- [22] Tina Rezaei, Ali Hamze, "An Efficient Approach For Malware Detection Using PE Header Specifications," *2020 6th International Conference on Web Research (ICWR)*, pages: 234-239, IEEE, 2020.
- [23] Güney Uğurlu, Koray Açıcı, "An Analysis Tool that Detects the Code Caves in Specified Sizes for Portable Executable Files," *2022 International Conference on Theoretical and Applied Computer Science and Engineering (ICTASCE)*, Ankara, TURKEY, pages: 38-41, IEEE, 2022.
- [24] Koray Açıcı, Güney Uğurlu, "A Reverse Engineering Tool that Directly Injects Shellcodes to the Code Caves in Portable Executable Files," *2022 International*

Conference on Theoretical and Applied Computer Science and Engineering (ICTASCE), Istanbul, TURKEY, pages: 42-45, IEEE, 2022.

- [25] Tukkappa K Gundoor, Sridevi, "Identification Of Dominant Features in Non-Portable Executable Malicious File," 2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA), pages: 1-6, IEEE, 2022.
- [26] Yamiala Supriya; Gautam Kumar; Dammu Sowjanya; Deepali Yadav; Devarakonda Lakshmi Kameshwari, "Malware Detection Techniques: A Survey," 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), pages: 25-30, 2020.
- [27] Prateek Josh, "Artificial Intelligence with Python. A Comprehensive Guide to Building Intelligent Apps for Python Beginners and Developers," Published by Packt Publishing Ltd. January 2017.
- [28] https://scikit-learn.org/stable/modules/neural_networks_supervised.html, Date: Sunday, August 20, 2023. 1:32PM (At Work in Electricity). 1.17. Neural network models (supervised) — scikit-learn 1.3.0 documentation.
- [29] What Is a Decision Tree and How Is It Used? (careerfoundry.com). Tuesday, August 22, 2023. 8:49PM.
- [30] Shehroz S. Khan, Michael G. Madden, "One-class classification: taxonomy of study and review of techniques," The Knowledge Engineering Review, Vol. 29, Pages: 345 - 374, 2004, Cambridge University Press.
- [31] Moya, M. M.; Koch, M. W.; Hostetler, L. D., "One-class classifier networks for target recognition applications," NASA STI/Recon Technical Report, Volume 93, Page: 24043, 1993.

- [32] Bilal Hadjadji, Youcef Chibani, "Optimized selection of training samples for One-Class Neural Network classifier," 2014 International Joint Conference on Neural Networks (IJCNN), pp. 345–349, IEEE, 2014.
- [33] Fahad Sohrab, Jenni Raitoharju, Alexandros Iosifidis, Moncef Gabbouj, "Ellipsoidal Subspace Support Vector Data Description," IEEE Access, Volume 8, pages 122013-122025, IEEE, 2020.
- [34] Fahad Sohrab, Jenni Raitoharju, Alexandros Iosifidis, Moncef Gabbouj, "Multimodal subspace support vector data description," Pattern Recognition Journal, Volume 110 Pages 107648, Publisher: Elsevier, 2021.
- [35] Hasan H. Al-Khshali, Muhammad Ilyas, Osman N. Ucan, "Effect of PE File Header Features on Accuracy," 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pages: 1115-1120, IEEE, IEEE Explore 05 January 2021.
- [36] Hasan H. Al-Khshali and Muhammad Ilyas, "Impact of Portable Executable Header Features on Malware Detection Accuracy," Computers, Materials and Continua, Volume 75, number 1, Tech Science Press, 2023.
- [37] Nahid Malek, Mehdi Bateni, Hamid Rastegar, "An Improved Method for Packed Malware Detection using PE Header and Section Table Information," International Journal of Computer Network and Information Security journal, Volume 11, number 9, pages 9, publisher: Modern Education and Computer Science Press, Published Online September 2019 in MECS.
- [38] Mohaddeseh Zakeri, Fatemeh Faraji Daneshgar, Maghsoud Abbaspour, "A static heuristic approach to detecting malware targets," Security and Communication networks Journal, publisher: Wiley Online Library, Volume 8, Issue number 17, Pages 3015-3027, 08 April 2015.

- [39] Jinrong Bai, Junfeng Wang and Guozhong Zou, "A Malware Detection Scheme Based on Mining Format Information," The Scientific World Journal, publisher: Hindawi, Volume 2014, Article ID 260905, 2014.
- [40] A. A. Zaidan, B. B. Zaidan, Fazidah Othman, "New Technique of Hidden Data in PE-File with in Unused Area Two," International Journal of Computer Science and Network Security (IJCSNS), Volume 13, Issue (number) 5, pages 108, 2013.
- [41] Yibin Liao, "PE-Header-Based Malware Study and Detection," Retrieved from: The University of Georgia, Athens, GA 30605, https://cobweb.cs.uga.edu/~liao/PE_Final_Report.pdf, 2012.
- [42] Dhruwajita Devi and Sukumar Nandi, "PE File Features in Detection of Packed Executables," International Journal of Computer Theory and Engineering, Vol. 4, Issue (No.) 3, pages 476, publisher IACSIT Press, June 2012.
- [43] Ionut Indre and Camelia Lemnaru, "Detection and prevention system against cyber attacks and botnet malware for information systems and Internet of Things," 2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP), pages 175 – 182, IEEE, 2016.
- [44] Ajit Kumar, K.S. Kuppusamy, G. Aghila, "A learning model to detect maliciousness of portable executable using integrated feature set," Journal of King Saud University - Computer and Information Sciences, Volume 31, issue (number) 2, pages 252 – 265, publisher Elsevier, 2019.
- [45] Edward Raff, Jared Sylvester and Charles Nicholas, "Learning the PE Header, Malware Detection with Minimal Domain Knowledge," Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Pages 121–132, November 2017.

- [46] Nguyen Thanh Vu. and Dung Hoang Le., “A virus detection model using portable executable feature extraction,” 2019 by the author(s). Distributed under a Creative Commons CC BY license-Posted: 26 August 2019.
- [47] Data Set obtained from the following site: <https://www.kaggle.com/amauricio/pe-files-malwares/data>. And the License to use this Data Set can be found at <https://creativecommons.org/publicdomain/zero/1.0/>.
- [48] Filip Zatloukal and Jiri Znoj, “Malware Detection Based on Multiple PE Headers Identification and Optimization for Specific Types of Files,” Journal of Advanced Engineering and Computation (JAEC), Volume 1 Issue 2, pages 153-161, 2017.
- [49] Janu Gupta and Jasbir Singh, “Detecting Anomaly Based Network Intrusion Using Feature Extraction and Classification Techniques,” International Journal of Advanced Research in Computer Science, Vol. 8, Issue 5, pages 1453-1456, 2017.
- [50] Hoda El Merabet and Abderrahmane Hajraoui, “A Survey of Malware Detection Techniques based on Machine Learning,” International Journal of Advanced Computer Science and Applications (IJACSA), vol. 10, no. 1, pp. 366–373, 2019, Publisher: Science and Information (SAI) Organization Limited.
- [51] Ajit Kumar, G. Aghila, “Portable executable scoring: What is your malicious score?” 2014 International Conference on Science Engineering and Management Research (ICSEMR), pages 1–5, IEEE, 2014.
- [52] Liu Binxiang, Zhao Gang and Sun Ruoying, “A Deep Reinforcement Learning Malware Detection Method Based on PE Feature Distribution,” 2019 6th International Conference on Information Science and Control Engineering (ICISCE), pages 23–27, IEEE, 2019.

- [53] Tzu-Yen Wang, Chin-Hsiung Wu and Chu-Cheng Hsieh, "Detecting Unknown Malicious Executables Using Portable Executable Headers," 2009 Fifth International Joint Conference on INC, IMS and IDC, pages 278-284, IEEE, 2009.
- [54] Tzu-Yen Wang and Chin-Hsiung Wu, "Detection of packed executables using support vector machines," 2011 International Conference on Machine Learning and Cybernetics, Volume 2, Pages 717-722, IEEE, 2011.
- [55] Vasaka Visoottiviseth, Pranpariya Sakarin, Jetnipat Thongwilai and Thanakrit Choobanjong, "Signature-based and Behavior-based Attack Detection with Machine Learning for Home IoT Devices," 2020 IEEE Region 10 Conference (TENCON), pages 829-834, IEEE, 2020.
- [56] Rafiqul Islam, A.W. Naji, A. A. Zaidan and B. B. Zaidan, "New System for Secure Cover File of Hidden Data in the Image Page within Executable File Using Statistical Steganography Techniques," International Journal of Computer Science and Information Security, IJCSIS, Vol. 7, No. 1, pp. 273-279, 2010, USA.
- [57] Joseph Clark and Shankar Banik, "Building Contemporary and Efficient Static Models for Malware Detection," 2020 IEEE SoutheastCon, Pages 1-6, IEEE, 2020.
- [58] Zhang Fuyong and Zhao Tiezhu, "Malware Detection and Classification Based on N-Grams Attribute Similarity," 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), volume 1, pages 793-796, IEEE, 2017.
- [59] Anitta Patience Namanya, Qublai Khan Ali Mirza, Hamad Al-Mohannadi, Irfan U. Awan and Jules Ferdinand Pagna Disso, "Detection of Malicious Portable Executables Using Evidence Combinational Theory with Fuzzy Hashing," 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), pages 91-98, IEEE, 2016.

- [60] M Shahid Yousaf, Muhammad Hanif Durad and Maleeha Ismail, "Implementation of Portable Executable File Analysis Framework (PEFAF)," 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), pages 671-675, IEEE, 2019.
- [61] Venkata Atluri, "Malware Classification of Portable Executables using Tree-Based Ensemble Machine Learning," 2019 SoutheastCon, pages 1-6, IEEE, 2019.
- [62] Babak Bashari Rad, Maslin Masrom and Suahimi Ibrahim, "Opcodes histogram for classifying metamorphic portable executables malware," 2012 International Conference on E-Learning and E-Technologies in Education (ICEEE), Pages 209-213, IEEE, 2012.
- [63] Muhammad Umair Saeed, Dale Lindskog, Pavol Zavarsky and Ron Ruhl, "Two techniques for detecting packed portable executable files," International Conference on Information Society (i-Society 2013), pages 22-26, IEEE, 2013.
- [64] Fauzan Hikmah Ramadhan, Vera Suryani and Satria Mandala, "Analysis Study of Malware Classification Portable Executable Using Hybrid Machine Learning" 2021 International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA), pages 86– 91, IEEE, 2021.
- [65] Hongbi Kim and Taejin Lee, "Research on Autoencoder Technology for Malware Feature Purification," 2021 21st ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter), pages 236-239, IEEE, 2021.
- [66] Julianne Alyson Diaz and Argel Bandala, "Portable Executable Malware Classifier Using Long Short Term Memory and Sophos-ReversingLabs 20 Million Dataset," TENCON 2021 - 2021 IEEE Region 10 Conference (TENCON), pages 881-884, IEEE, 2021.

- [67] Farnoush Manavi and Ali Hamzeh, "Static Detection of Ransomware Using LSTM Network and PE Header," 2021 26th International Computer Conference, Computer Society of Iran (CSICC), pages 1-5, IEEE, 2021.
- [68] Wanjia Zheng and Kazumasa Omote, "Robust Detection Model for Portable Execution Malware," ICC 2021 - IEEE International Conference on Communications, pages 1-6, IEEE, 2021.
- [69] Deniz Demirci, Nazenin şahin, Melih şirlancis and Cengiz Acarturk, "Static Malware Detection Using Stacked BiLSTM and GPT-2," IEEE Access, Volume 10, pages 58488-58502, 2022.
- [70] Yun Gao, Hirokazu Hasegawa, Yukiko Yamaguchi and Hajime Shimada, "Malware Detection Using LightGBM With a Custom Logistic Loss Function," IEEE Access, Volume 10, pages 47792-47804, IEEE, 2022.
- [71] S. Mihajlović, A. Kupusinac, D. Ivetić, I. Berković, "The Use of Python in the field of Artificial Intelligence," International Conference on Information Technology and Development of Education – ITRO 2020, Zrenjanin, Republic of Serbia, 2020.
- [72] Saima Naz and Dushyant Kumar Singh, "Review of Machine Learning Methods for Windows Malware Detection," 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pages 1-6, IEEE, 2019.
- [73] Dang Kien Hoang, Dai Tho Nguyen and Duy Loi Vu, "IoT Malware Classification Based on System Calls," 2020 RIVF International Conference on Computing and Communication Technologies (RIVF), pages 1-6, IEEE, 2020.
- [74] Cristiano Antonio de Souza, Carlos Becker Westphall, Renato Bobsin Machado, Leandro Loffi, Carla Merkle Westphall and Guilherme Arthur Geronimo, "Intrusion detection and prevention in fog based IoT environments: A systematic literature review," Computer Networks, Volume 214, pages 109154, 4 September 2022, publisher Elsevier.

- [75] Hamed HaddadPajouh, Ali Dehghantanha, Raouf Khayami, Kim-Kwang Raymond Choo, "A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting," *Future Generation Computer Systems*, Volume 85, Pages 88-96, August 2018, publisher Elsevier.
- [76] Shiming Xia, Zhisong Pan, Zhe Chen, Wei Bai and Haimin Yang, "Malware Classification with Markov Transition Field Encoded Images," 2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC), pages 1-5, IEEE, 2018.
- [77] S. H. Lee, S. C. Lan, H. C. Huang, C.W. Hsu, Y.-S. Chen, and S. Shieh, "Ec-model: An evolvable malware classification model," in 2021 IEEE Conference on Dependable and Secure Computing (DSC), pp. 1–8, IEEE, 2021.
- [78] M. A. H. Saeed, "Malware in computer systems: Problems and solutions," *International Journal on Informatics for Development (IJID)*, vol. 9, no. 1, pages 1–8, 2020.
- [79] J. Acharya, A. Chuadhary, A. Chhabria, and S. Jangale, "Detecting malware, malicious URLs and virus using machine learning and signature matching," in 2021 2nd International Conference for Emerging Technology (INCET), pages 1–5, IEEE, 2021.
- [80] H. Shukla, S. Patil, D. Solanki, L. Singh, M. Swarnkar, and H. K. Thakkar, "On the design of supervised binary classifiers for malware detection using portable executable files," in 2019 IEEE 9th International Conference on Advanced Computing (IACC), pages 141–146, IEEE, 2019.
- [81] Alberto Huertas Celdrán, Pedro Miguel Sánchez Sánchez, Miguel Azorín Castillo, Gérôme Bovet, Gregorio Martínez Pérez and Burkhard Stiller, "Intelligent and behavioral-based detection of malware in IoT spectrum sensors," *International Journal of Information Security*, Volume 22, issue (number) 3, pages 541-561, 2023, publisher Springer.

- [82] Mo'ath Zyout, Raed Shatnawi and Hassan Najadat, "Malware classification approaches utilizing binary and text encoding of permissions," *International Journal of Information Security*, volume 22, pages1687–1712, 2023. Publisher Springer.
- [83] Kateryna Chumachenko, "Machine learning methods for malware detection and classification," 2017.
- [84] Qian Gong, Phil DeMar and Mine Altunay, "ThunderSecure: deploying real-time intrusion detection for 100G research networks by leveraging stream-based features and one-class classification network," *International Journal of Information Security*, Volume 21, number 4, pages 799-812, 2022. Publisher Springer.
- [85] Qiguang Miao, Jiachen Liu, Ying Cao and Jianfeng Song, "Malware detection using bilayer behavior abstraction and improved one-class support vector machines," *International Journal of Information Security* volume 15, pages361–379, 2016. Springer.
- [86] Sertac Kilickaya, Mete Ahishali, Fahad Sohrab, Turker Ince and Moncef Gabbouj, "Hyperspectral Image Analysis with Subspace Learning-based One-Class Classification," *arXiv preprint arXiv:2304.09730*, 2023.
- [87] A. Degerli, F. Sohrab, S. Kiranyaz, and M. Gabbouj, "Early myocardial infarction detection with one-class classification over multi-view echocardiography," in *2022 Computing in Cardiology (CinC)*, vol. 498. IEEE, 2022, pp. 1–4.
- [88] Angelo Oliveira and R SASSI, "Malware analysis datasets: Pe section headers," *IEEE Dataport*, Volume 10, 2019.
- [89] K. Li, W. Ma, H. Duan, H. Xie, J. Zhu, R. Liu, "Unbalanced network attack traffic detection based on feature extraction and GFDA-WGAN," *Computer Networks*, Volume 216, pages 109283, 2022.

- [90] T. Barabosch and E. Gerhards-Padilla, "Host-based code injection attacks: A popular technique used by malware," in 2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE). IEEE, 2014, pp. 8–17.
- [91] B. Vignau, R. Khoury, and S. Hallé, "10 years of IoT malware: A feature based taxonomy," in 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2019, pp. 458–465.
- [92] Pramuditha Perera, Poojan Oza, Vishal M. Patel, "One-Class Classification: A Survey," arXiv preprint arXiv:2101.03064, 2021.
- [93] S. Yu, X. Li, X. Zhang, and H. Wang, "The ocs-svm: An objective-cost-sensitive svm with sample-based misclassification cost invariance," IEEE Access, vol. 7, pp. 118 931–118 942, 2019.
- [94] F. Sohrab and J. Raitoharju, "Boosting rare benthic macroinvertebrates taxa identification with one-class classification," in 2020 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2020, pp. 928–933.
- [95] R. Vlasveld, "Introduction to one-class support vector machines," <http://precog.iiitd.edu.in/people/anupama>, Jul 2013.
- [96] H. Wang, S. Yan, D. Xu, X. Tang, and T. Huang, "Trace ratio vs. ratio trace for dimensionality reduction," in IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.
- [97] A. Kumar, K. Kuppusamy, G. Aghila. A learning model to detect maliciousness of portable executable using integrated feature set (2019). URL <https://github.com/urwithajit9/clamp>.
- [98] A. Oliveira. Malware analysis datasets: Pe section headers (2019). DOI 10.21227/2czhes14. URL <https://dx.doi.org/10.21227/2czhes14>.

- [99]N. Kwak, “Nonlinear projection trick in kernel methods: An alternative to the kernel trick,” IEEE transactions on neural networks and learning systems, vol. 24, no. 12, pp. 2113–2119, 2013.

