

A Novel Approach to Clustering Malware Behaviour to Improve Malware Detection

Rebecca Merriman

Technical Report

RHUL-ISG-2020-6

22 June 2020



Information Security Group
Royal Holloway University of London
Egham, Surrey, TW20 0EX
United Kingdom

A Novel Approach to Clustering Malware Behaviour to Improve Malware Detection

Rebecca Merriman, 100812232

Submitted as part of the requirements for the award of the

MSc in Information Security

at Royal Holloway, University of London

Supervisor: Dr Daniele Sgandurra



I declare that this assignment is all my own work and that I have acknowledged all quotations from published or unpublished work of other people. I also declare that I have read the statements on plagiarism in Section 1 of the Regulations Governing Examination and Assessment Offences, and in accordance with these regulations I submit this project report as my own work.

Signature:

A handwritten signature in black ink that reads 'Rkmerriman'.

Date: 17th August, 2019

Table of Contents

Executive Summary	3
Acknowledgements	4
1 Introduction	5
1.1 My Project	6
1.2 Statement of Objectives	7
1.3 Motivation	7
2 Literature	9
2.1 Ransomware Report	10
2.2 Backdoor Report	12
2.3 Trojan Report	13
2.4 Feature Selection and Model Construction Report	13
2.5 Machine Learning Focusing On Clustering Report	16
3 Design and Implementation for Clustering Malware	25
3.1 Description of Program and Justification of Feature Selection and Validation Methods	26
4 Results for Clustering Malware	35
4.1 Results of Clustering Ransomware	36
4.2 Results of Clustering Backdoor	45
4.3 Results of Clustering Trojan	54
5 Discussion	64
5.1 Comparison between Clustering Ransomware, Backdoor and Trojan	65
5.2 Accuracy of Clustering-Based Malware Detection	71
5.3 Self-Evaluation	75
6 Conclusion	77
6.1 Conclusion	78
Bibliography	79
7 Appendices	82
7.1 Appendix A - Program	82

7.2	Appendix B - Text Files Of Results	92
7.3	Appendix C - Euclidean Distance Results for Ransomware, Backdoor and Trojans . . .	100
7.4	Appendix D - Results from Clustering Android Application Behaviour	101

Executive Summary

My project objective is to implement clustering for operations of three different types of malware, specifically Ransomware, Backdoor and Trojan in order to evaluate the accuracy of clustering-based malware detection to then conclude using these results and results from other papers whether clustering malware behaviour improves malware detection.

To meet this aim, smaller objectives need to be met. My objectives include reviewing existing literature on Ransomware, Trojans, Backdoors and clustering and then implementing the clustering process of Ransomware, Backdoor and Trojan families from behaviour profiles (produced from static and dynamic analysis) to validation/results. The results are critically compared and the accuracy of clustering-based malware detection from my and other results is evaluated.

Ransomware, Backdoor and Trojan are all types of malware. Malware is malicious software which undermines the security of users by performing unwanted functions. The clustering of Ransomware, Backdoors and Trojans were compared because they can work alongside each other to infect a victim's system. Ransomware can be used to install Backdoors into a system and Trojans can be used to deliver Backdoors or Ransomware into a system.

In this project, the clustering process was carried out on 3 malware families (Ransomware, Trojans and Backdoor), making sure that samples in the same cluster were as similar as possible and samples in a different cluster were as dissimilar as possible. 12 experiments were run all together and the best clustering for each of the different types of malware for each of the validation metrics was evaluated. The best clustering according to all the FMS, F1, ARI and SC scores for Ransomware was the Uni-gram with the system call representation of Category and vector representation of Frequency Vector, for Backdoor it was the Di-gram with the system call representation of Full Representation and vector representation of Frequency Vector and for Trojan it was the Tri-gram with the system call representation of Category and vector representation of Frequency Vector.

The main finding was that there is a discrepancy with the accuracy of clustering-based malware and whether clustering improves malware detection. The accuracy of clustering-based malware detection is highly subjective as it depends on many factors including the type of machine learning algorithm, the features selected, the feature selection methods, the model construction methods and evaluation metrics. This is illustrated in my results where the different methods of feature selection and vector representation yielded different results (scores and best clustering methods) for the validation metrics and the accuracy was low. In contrast other papers used different methods and found that accuracy was high, providing evidence for this subjectivity and conflicting findings. Therefore, future research should be conducted to find out all the reasons that may affect the accuracy of clustering malware and discover the best methods in terms of accuracy and a good run time for clustering malware to improve malware detection. This will then help to conclude whether clustering improves malware detection or not.

Acknowledgements

Specific gratitude is given to my supervisor Daniele Sgandurra who has provided me fantastic guidance, knowledge, kindness, supervision and support throughout this MSc.

A big thank you to my sister Siobhan Merriman for proofreading my dissertation, for providing suggestions and support and for helping me with decisions along the way.

A big thank you to my parents Brendan and Shree Merriman and my siblings Siobhan and Sean Merriman for their encouragement, support and understanding throughout my education.

Finally, a big thank you to my boyfriend Athul Cyril for his motivation, encouragement and support throughout my MSc.

1

Introduction

This chapter introduces my project including background information and the objectives of my project.

In particular, it discusses my project and the clustering process involved (1.1), a statement of my objectives(1.2) and motivation for my project(1.3).

1.1 My Project

My project objective is to implement clustering for recorded behaviour types of three different types of malware, specifically Ransomware, Backdoor and Trojan in order to evaluate the accuracy of clustering-based malware detection to then conclude whether clustering malware behaviour improves malware detection.

Ransomware, Backdoor and Trojan are all types of malware. Malware is malicious software which undermines the security of users by performing unwanted functions. Malware is dangerous due to its substantial and pervasive nature. Due to the rise of the digital age, malware attacks are on the increase and there is no evidence to suggest that it is slowing down. Therefore, we need to find novel ways to detect and prevent these attacks from occurring in the future. Clustering is one of these methods that can be used. The clustering of Ransomware, Backdoors and Trojans were compared because they can work alongside each other to infect a victim's system. Ransomware can be used to install Backdoors into a system and Trojans can be used to deliver Backdoors or Ransomware into a system.

The following clustering process was used:

The dataset used in this project presents features (system call category and action) extracted from samples analysis using static and dynamic analysis. Identifying system calls from the specific families can allow the malware behaviours to be reconstructed as system calls at different levels of abstractions. This is more effective than the individual behavioural profiles themselves as the behavioural profiles will contain a lot of information that is irrelevant to the behaviour of the malware. All the irrelevant and redundant features were removed by MIST translation.

All irrelevant and redundant features were removed based on the variables measured, and the relevant features were extracted (feature selection). Different methods of feature selection were implemented with different levels of abstractions. The features (operation) were represented as a vector (bit or frequency n-grams) in order to be inputted into a clustering algorithm. Different methods of model construction were implemented with different levels of abstractions.

Hierarchical clustering was implemented on the different methods of feature selection and model construction. Machine Learning is an application of artificial intelligence which automatically detects patterns in data, to predict future data or to perform a decision making process under uncertainty[1]. There are two types of machine learning, supervised and unsupervised. In supervised learning, the model for predicting an output (label on future data[2]) is based/predicted on one or more inputs and in unsupervised learning, relationships and structure or interesting patterns can be learnt from the data[3]. Clustering is a type on unsupervised learning method which[3] tries to group similar objects together by looking at whether the "observations fall into distinct groups" based on the inputs. In relation to this project, hierarchical clustering was performed on each of the different malware types separately to try group the samples back into the correct family. It exposed clusters of Ransomware, Backdoor or Trojan malware families where all objects in the same cluster exhibit similar behaviour or have similar characteristics.

After hierarchical clustering the clusters obtained from various feature selection and model construction methods were evaluated to determine the best clustering from a given set of malware samples. The best clustering from the specific feature selection and model construction methods should yield a clustering as close to the labelled dataset as possible. The evaluation metrics Fowlkes Mallows Score (FMS), F1-Score (F1), Adjusted Rand Index Score (ARI) and Silhouette Coefficient (SC) were used to evaluate how similar the clusters were to the dataset.

The results of the FMS, F1, ARI and SC of Ransomware, Trojans and Backdoors were compared and then the accuracy of clustering-based malware was evaluated based on these results and other results from papers or projects.

1.2 Statement of Objectives

1. Review and summarise relevant existing literature on Ransomware, Trojans and Backdoors to understand their function evolution and how it works.
2. Review and summarise relevant existing literature on dynamic analysis, feature selection and model construction.
3. Review and summarise relevant existing literature on machine learning specifically clustering and validation.
4. Implement a program to use static and dynamic analysis of malware samples and perform feature selection, hierarchical clustering (cluster Ransomware into families) and validation on Ransomware, Backdoor and Trojans.
5. Describe and justify my choices for the different methods of feature selection and validation supported by results and evidence.
6. Critically compare and visualise results between clustering Ransomware into families, Backdoor into families and Trojan into families.
7. Evaluate the accuracy of clustering-based malware detection from my results of Ransomware, Backdoors, Trojans, other papers and android malware.

1.3 Motivation

Clustering malware behaviour is useful for the information security industry in detection because it can discover features which are useful in classifying the families of Ransomware, Backdoor and Trojans, find commonalities between features within the same families and give them to an antivirus program at run time to detect these families. Also if it performs with high accuracy then the features selected can be used to detect Ransomware, Backdoors and Trojans at run time in the future. For example if there is a new family of Ransomware which does not exist now then the features known to be good can be used to detect a new family of Ransomware (train the classifier to recognise the new family of Ransomware based on features of previous families) (the same applies to Backdoors and Trojans). Also there is little literature on classification of malware using machine learning; it mostly concentrates on detection and decoy.

I chose to carry out this project because I really enjoyed the clustering project which I completed for my BSc last year on android applications and I wanted to extend the skills (e.g. python or latex) and knowledge that I have learnt on my BSc project into my masters project and carry out similar work with Ransomware, Trojans and Backdoors. Additionally I learnt about these types of malware in a BSc course and I found it intellectually stimulating.

Due to huge amounts of data being available now compared to in the past (big data) and the rise in computational capacity and tools available to process and analyse data in real time, one cannot use traditional techniques or humans to analyse information. People now use machine learning techniques to do this and instead of humans manually analysing the data to look at trends and patterns, they now direct, develop and harness machine learning and interpret results[4]. Machine learning techniques are a good way of recognising trends and patterns in the data. It can be used in many domains such as the security domain for example to help identify trends and meaningful patterns to help predict, defend and respond to attacks in the future. Machine Learning and AI will not only make cyber-attacks more powerful but will also make cyber defence just as powerful. It can be used in many other domains such as the health sector or the financial sector for example to help insurers to provide digital, personal and relevant information e.g. more focused sales and marketing and reduced operational costs[5]. This is why machine learning techniques are so important. It replaces human jobs which have now become impractical, less accurate (e.g. financial services model using machine learning techniques to produce GDP estimates have said it had been very accurate for the last 16 years[6]), less time consuming to carry out and used in many domains.

This is why I performed clustering on malware behaviours. I wanted to see how accurate these techniques were at predicting malware from other malware types to help organisations predict, defend and respond to attacks in the future through the use of machine learning.

2

Literature

This chapter describes the three types of malware concentrated in this project and the process of clustering.

In particular:

Section 2.1 discusses the two types of Ransomware2.1.1 (crypto and locker Ransomware), the history of Ransomware2.1.2, stages of attack2.1.3 and examples2.1.5.

Section 2.2 and Section 2.3 discuss Backdoors and Trojans respectively, the creation of them and an example.

Section 2.4 discusses two processes (feature selection and model construction) which are used in the clustering process to capture features from samples and produce them into a form suitable for machine learning.

Section 2.5 discusses the two types of machine learning2.5.2 (supervised and unsupervised learning) and examples of both and validation metrics2.5.5). It focusses on clustering, the machine learning algorithm which is used in this project.

2.1 Ransomware Report

According to Savage et al.[7], today, Ransomware is “one of the most troublesome malware categories of our time”.

Malicious software (malware) such as viruses, Trojans, worms etc. undermine the security of users by performing unwanted functions without their permission e.g. stealing personal data. Ransomware is a type of malware specifically scareware which aims to prevent users from accessing their system or files (anything that they want/need) until a ransom is paid[7][8]. There are 2 types of Ransomware; locker Ransomware and crypto Ransomware.

2.1.1 Two Types of Ransomware

Locker Ransomware aims to lock the victims computer to prevent them from using it and then demand a fee to restore access to it. As the computer is locked, it will run with limited capabilities to only allow the user to pay the ransom, whilst the underlying system and files are left untouched (not encrypted, changed or damaged)[7]. It is middle grade Ransomware therefore it is not as destructive and is less effective in extracting payment compared to crypto Ransomware[7][8]. This means that in order to extract payment from the user, the attacker may have to carry out social-engineering techniques[7]. For example a law enforcement scam which claims that it has detected illegal activity on the computer, and so one has to pay a fine as a punishment for this activity[8].

Crypto Ransomware aims to encrypt personal data and files to make them inaccessible to the user until the victim pays the ransom to decrypt the files[7]. It is the most dangerous type of Ransomware and is very effective in extracting payment from the user[7][8]. This is because the data stored on a computer is likely to be valuable and important to victims (such as photos or work documents) therefore they will pay the ransom fee to get their data back instead of losing it. Even if the computer has been infected by Ransomware, it will still work normally as the critical system files are left untouched (it is only the encrypted files that the user cannot access)[7]. For example a pop up message which says that all the files have been encrypted and it demands a payment for decrypt them[8].

2.1.2 Ransomware History

Previously, few people owned a computer, less information was stored online, less attacks occurred and malware was used for pranks or games with no malicious intent and everything was designed for benign users not malicious users. Now attacks occur daily, everyone owns a computer, everything is now online, it's not just new malware but also variants of the same malware are occurring ([9]) and malware is used for financial gain and malicious intent. Ransomware attacks have been on the increase especially crypto Ransomware as there is no need for social engineering, compared to locker Ransomware and the victims are more likely to pay (see above). This increase is in terms of the ransom demand, the number of people affected (instead of attacking individuals, companies/ organisation such as police, hospitals, banks, education etc. are now being targeted more especially those who store personal information[9] and need this information in real-time, so the victims are more likely to pay the ransom) and frequency of attacks. There is no evidence to say that it is slowing down.

The first Ransomware occurred in 1989 with the AIDS Trojan[7]. It was introduced into systems through floppy disks[7] and hid directories and encrypted all file names on the C: drive. The users were asked to renew a licence to PC Cyborg Corporation by paying a \$189 cheque (the ransom demand)[10]. Another two notable Ransomware attacks are Wannacry (2017) and NOT-PETYA (2017). Wannacry was a worldwide attack affecting thousands of computers within hours. It used a network worm which had mechanism to spread itself automatically by exploiting vulnerabilities in Windows. Once installed on a computer it spread by using an EternalBlue exploit to infect other computers on the local network and by scanning random IP addresses to spread across the internet and infect other computers. Wannacry was stopped by a 'kill switch' feature. This is because it worked by contacting a specific domain, and if it was contactable then the mal-

ware stopped spreading otherwise it would continue to spread to other computers and encrypt files. Therefore, Wannacry was stopped by registering the used domain. Notpetya was another Ransomware which also propagated via the ExternalBlue exploit and tried to extract logins and passwords to spread the infection to other IP addresses linked to the organisation. Once in the computer, it overwrote the master boot record to hijack the normal loading and encrypted other files. After the system was rebooted, it loaded the master boot record and encrypted the entire disc and displayed a \$300 bitcoin ransom note. However it was a wiper therefore even if the victim paid the ransom, they were not able to decrypt and recover the files [9].

2.1.3 Ransomware Stages

There are many stages of a Ransomware attack[11]. They are:

1. Propagation - This is how the victim gets infected with Ransomware and can be in many forms including[7]:
 - (a) A traffic distribution service - buy redirected web traffic and point it to a website hosting an exploit kit which runs in the victims computer.
 - (b) Malvertising - malicious advertisements either clicked on by the user or hosted onto legitimate websites to redirect traffic to a website hosting an exploit kit which runs in the victims computer.
 - (c) Phishing or SPAM - via botnets usually in the form of an email containing a malicious attachment or a link to a website hosting an exploit kit which runs in the victims computer.
 - (d) Self propagation and social engineering - Ransomware which contains functionality to spread such as a worm.
2. Installation - After infecting the victims computer by copying it into various locations, the Ransomware installs itself and sets up keys in the registry to start automatically at reboot[11].
3. Key generation - The Ransomware server generates two cryptographic keys; one stored on the victims computer and one on the criminals' server[11].
4. Data encryption - The Ransomware encrypts every file it finds[11]. Often attackers want productivity files (files used the most) or valuable, important and personal user data to be attacked. This is because they affect the users the most if lost so they will pay the ransom fee to get their data back instead of losing it[7].
5. Extortion - The Ransomware displays a screen with a ransom fee and time limit to pay after which the key to decrypt the files will be destroyed[11]. The Ransomware needs to convince the victim to pay the ransom. Tactics for crypto Ransomware include scaremongering victims e.g. time limits, the endowment or fear of regret (people value their possessions) and Ellsberg paradox (people prefer known probabilities of winning in risky situations therefore they will pay the ransom to get their data back instead of considering the impact of data loss). Tactics for locker Ransomware include deception through fake authorities such as the law enforcement, using central and peripheral route to persuasion and the influence of framing (based on the prospect theory that states people become risk-averse over prospects involving gains and people become risk-loving over prospects involving losses). These tactics will pressurise them to pay the ransom. The ransom in locker Ransomware is usually paid in the form of payment vouchers as the victim cannot get access to the system and in crypto Ransomware the ransom is usually paid via cryptocurrencies such as bitcoin because of the anonymity associated with it[7].
6. Recovery - in locker Ransomware the system will be unlocked and in crypto Ransomware a decryption key will be sent to the victim[11].

2.1.4 To Pay or Not To Pay

One big talking point that needs to be considered in Ransomware is whether the victim should pay the ransom or not. Arguments for not paying the ransom include the idea that the victim will be fostering criminal activity and as the FBI said ([12]) it will provide criminals with an incentive for more people to get involved and to target more organisations and there is no guarantee that the victim will get their files decrypted (crypto Ransomware) or their system unlocked (locker Ransomware)[8]. Also there is no guarantee that the attacker is out of the system (access is revoked), they may decrypt the files and unlock the system but they do not have to revoke their access. However criminals like maintaining their reputation and a trust relationship is built between the victim and attacker so are likely to unlock the system or decrypt the files[7].

2.1.5 Ransomware Examples

Ransomware can occur anywhere. For example it has occurred in android devices such as android defender (fake antivirus and the first android Ransomware) and lockerpin (set or change the device PIN and lock device). Android Ransomware acts similarly to a Trojan horse where it "spreads by masquerading as a legitimate application ... that will make the victim download the malware" such as trending games. Once the malware has been installed onto the device it reports back to a C&C server with device information such as the device model and IMEI and the C&C server sends commands such as lock device, steal contacts and steal/send SMS messages[13]. Ransomware has also been seen in Linux (for example Linux encoder which was the first Linux Ransomware which exploited vulnerabilities in web-based plug-ins[14]) and databases (for example mysql databases were attacked brute-forcing to get the root password and create a new table to instruct the victims to pay 0.2 Bitcoins[15]). Recently Ransomware has been provided as a service (sold to people) to those who want to carry out a Ransomware attack (Ransomware-as-a-service (RAAS)). This allows these attackers to generate a Ransomware attack without the need to create, maintain or run the Ransomware. Usually the vendor of the Ransomware gets paid commission. For example Torlocker where the buyers will be provided with the crypto Ransomware binary file and access to a control panel for US\$300. Due to technological advancements and the interconnectivity and mobility of devices there are many more attack targets in this world. The new development of IOT (internet of things) devices means that they are now a target for attacks because they are connected to the internet[7]. For example smart tv via a man-in-the-middle attack where a user downloads an app onto their device, the request is redirected to another server which sends a malicious app to the TV[16]. More serious problems can also be caused as a result of these technical advancements such as internet-enabled cars because if these were targeted by Ransomware (or any other type of malware) it can prevent the car from functioning properly or can potentially take over the controls of the car and cause a crash[7].

2.2 Backdoor Report

A Backdoor can be defined as a technique which bypasses the security of a system in order to access its data[17].

It can be created in two ways. Firstly, in a benign way by the developer or a legitimate vendor. This can be done to access an application or OS for the purpose of troubleshooting or resolving software issues or it can be introduced due to a programming error. Secondly, in a malicious way by attackers whereby it is installed by an exploit or taken advantage of by viruses and worms (other malware)[17]. The consequences of Backdoors are that security measures can be bypassed, attackers can gain privilege access (e.g. root access in Unix), data (e.g. finance and personal) can be stolen and it can be used as a platform to install additional malware such as Ransomware, spyware or carry out a Dos attack[18].

Typically, Backdoors are inserted into a system by a Trojan (type of malicious software pretending to be a legitimate program but performs malicious functions such as delivering malware, stealing data or malware installing a Backdoor such as Ransomware) or the hardware or software manufacturer in order to test applications and fix software bugs[18].

For example, Notpetya (as described above in the Ransomware report) was caused by a Backdoor Trojan disguised as a software update[18].

2.3 Trojan Report

Trojans are malicious programs that claim to perform a benign function but perform a malicious function instead by using deception and social engineering. Trojans can take the form of attachments, downloads, and fake videos/programs. It can be described as a standalone malware or a tool for other activities such as delivering future payloads e.g. Ransomware, Backdoors or spyware, communicating with the hacker or making the system easier to attack[19].

Trojans can appear in many forms such as legitimate applications, music, software or advertisements. Examples of infection methods include downloading cracked applications such as illegal free software copies (the illegal free software copies are Trojans), downloading unknown free programs from an untrusted site which is a Trojan or opening infected attachments which execute a Trojan when the attachment is clicked[19].

For example, Notpetya (as described above in the Ransomware report) was caused by a Backdoor Trojan disguised as a software update[18].

To summarise, Ransomware can be used to install Backdoors into a system and Trojans can be used to deliver Backdoors or Ransomware into a system. For this reason, I will be comparing the clustering of Ransomware, Backdoors and Trojans.

2.4 Feature Selection and Model Construction Report

Features have been defined as “behavioural characteristics of a sample”[20]. Zheng[21] defines features as numeric representations of data which are derived from the type of data available. They can either be dynamic (from executing code) or static (from analysing metadata or code) and numerical (e.g. the number of API calls) or categorical (e.g. a certificate that is signed or unsigned). If feature selection is performed correctly (i.e. the correct features are selected) then modelling becomes easier, the task is achieved and the process will have a higher chance of success[21].

Feature selection is the process of removing irrelevant or redundant features based on the variables measured so that only relevant features are used in model construction. These relevant features should have high intra-cluster similarity/ low within cluster variability (commonalities between members of the same class) and low inter-cluster similarity/ high between cluster variability (differences between members of different classes). This is needed due to the curse of dimensionality; the more features used, the lower the predictive performance. It reduces overfitting, improves accuracy and reduces training time.

There are many methods of feature selection and model construction. The following papers provide different methods of feature selection and model construction that they used to select features from behaviour traces or system calls.

2.4.1 Feature Selection and Model Construction Methods

Boutsidis, Mahoney and Drineas[22] carried out feature selection using a randomised algorithm. It is described as follows:

1. Input a matrix A , the number of clusters k and an accuracy parameter ϵ which is 0 or 1.
2. Compute the top- k right singular vectors of A .
3. Compute the normalised leverage scores π_i (square of the Euclidean norm of the i -th row of

$$V_k) P_i = \frac{\| (V_k)_i \|_2^2}{k}$$

4. Compute a sampling parameter r (the number of features to select) by $r = \theta \left(\frac{k \log(\frac{k}{\epsilon})}{\epsilon^2} \right)$.
5. Loop from 1 to r , keep the i th feature with probability p_i and multiply it by the factor $(r p_i)^{-1/2}$.
6. Return the $n \times r$ matrix containing the selected features.

After this they performed K-means clustering.

Machine learning algorithms require features to be represented as matrices and numbers mapped into Euclidean space therefore feature vectors are constructed out of the relevant features. There are different types of feature vectors. Bit vectors are where each feature is a dimension and is either present or not, histograms that splits the data into classes and counts the number of features in each class (normalised frequency), graphs, or n-grams which contain consecutive n sequences of system calls.

For example, Zheng[21] describes constructing vectors from text documents. There are many ways the vectors can be constructed they are a “bag-of-words” where the vector contains the frequency of the words (the number of appearances of that word) in each document, a “bag-of-n-grams” where an n-gram is a sequence of n words such as a word is 1-gram (uni-gram) or “Tf-Idf” (term frequency-inverse document frequency) which looks at the normalized count (word count is divided by the number of documents this word appears in) of the words in the document. If the word appears in many documents, the idf is near to 1 but 0 if it is in a few documents [21]. In contrast Canali et al.[23] used three combinations of atoms which are behavioural elements of a program e.g. library calls, system calls. They were N-grams which are “a sequence of n atoms that appear in consecutive order in the program execution trace”, tuples where a tuple signature of cardinality n (called an n-tuple) “combines the strict order relation of n-gram with the always-true matching filter of bag signatures”, and bags where a bag of cardinality n (called an n-bag) contains n atoms with no order relation.

Mutz et al.[24] assumed that all attacks occur in the system call arguments, and anything else which occurs in the normal execution is not detected. Models to characterise the features from the strings of system call arguments can include:

1. String Length - strings in system calls are usually file names (< 100 characters). In the learning phase, the system call argument string length distributions are approximated and deviants are detected. In the detection phase, the system call argument frequency is assessed and any length exceeding the mean of the lengths is classed as malicious.
2. String Character Distributions - to determine whether the string is normal. Strings have a regular structure and characters in strings are “not uniformly distributed and occur with different frequencies”. Relative frequencies of legitimate system calls decrease in value slowly compared to malicious which is fast or does not happen at all. In the learning phase, the character distributions (average of all character distributions) are calculated. In the detection phase, the probability (via statistics from the idealised character distribution) that a character distribution of an argument is a sample is calculated.
3. Structural inference - where grammar is inferred by analysing legitimate strings in the training phase. In the learning phase, structural inference is applied to the system call arguments and over-simplification (includes all training data) or over-generalisation (arbitrary strings produced) of the grammar can occur. Generalisation continues until the Markov models and Bayesian probability specify that the structural information will be lost. In the detection phase, if the string arguments are valid the model returns 1 and if the value cannot be derived from the grammar it returns 0.
4. Token finder - determines whether system call argument values are tokens or elements of enumeration. In the learning phase, the system call argument is classified based on the frequencies of the argument values as an enumeration or an identifier. In the detection phase, if it is classed as tokens of enumeration (any value is in the set of identifiers) a 1 is returned otherwise it returns 0 and if it is classed as an identifier it always returns 1.

To characterise the features of the string arguments each character was replaced by a token representing a character class e.g. digit and any repetitions of adjacent items in the same class were fused together into a token. For example the string “/etc/passwd” would become “slash-lowercase-slash-lowercase”. This was inputted into the structural inference model where any deviation was anomalous behaviour and a Bayesian network was produced. This contained a root/hypothesis node (either normal or anomalous) and child/model nodes for every model (captures the outputs connected to the root node). Dependencies could occur; these include correlation, confidence values or additional information such as anomalous behaviour. Once the normal behaviour profile is created, the variance of training data was evaluated. If it is high, a low confidence value was produced and a small feature set was used then the confidence of the correctness of the output was high. Conditional probability tables of the nodes were specified. The outputs of the model was a probability mapped onto anomaly score states (normal(0-0.5), uncommon(0.5-0.75), irregular(0.75-0.9), suspicious(0.9-0.95) or very suspicious(0.95-1)) which describes deviations of system call arguments from the expected normal values[24].

Many techniques can address feature selection to try and select the best features. They include using machine learning techniques that have methods to select the high importance features such as logistic regression, SVM (Support Vector Machines) and decision trees, L1 regularization or use a hard-coded method if the dataset is small such as “build it up” or “leave one out” approach. If there are more features than the number of data points overfitting will occur and highly correlated features results in instable decisions[2].

2.4.2 Feature Selection Considerations

There are many decisions to be made when selecting features from a dataset. The dataset could be unbalanced such as containing more “good samples” (e.g. legitimate) compared to “bad samples” (e.g. malicious). If one is using the data to cause machine learning techniques to classify rare events, then “bad samples” may not influence the classifier and the model will perform badly. To address this one can oversample the minority class, undersample the majority class or change the loss function. The dataset can also have missing features from events including delays or failures. This means that when training the data, the distribution can be wrong. To incorporate the missing features, the value of it will need to be imputed such as computing the average or medium value for the feature[2].

2.4.3 Examples

Some examples of how feature selection has been used in Ransomware can be seen in Zhang et al. They performed classification of Ransomware families with machine learning based on N-grams of opcodes[25]. Classifying Ransomware into families is useful for identifying variants of Ransomware and to reduce the workload of analysts. Their feature selection method is described as follows. They performed static analysis to analyse the Ransomware samples using the IDAPRO disassembler. They transformed opcode sequences from Ransomware samples into meaningful n-grams. They used n-grams as one opcode is not harmful but a sequence of opcodes in a specific order can be harmful and an n-gram can be used to represent many meaningful opcode sequences. Then (as in Zheng [21]) they calculated for each n-gram its term frequency-Inverse document frequency (TF-IDF) to select feature n-grams so that they can discriminate between families. TF-IDF allows the identification of important words and in a Ransomware family it evaluates the importance of an n-gram. This “increases proportionally with the frequency of occurrence of the n-gram in a family” and is “inversely proportional to the frequencies of occurrences of the same N-gram in all Ransomware families” meaning that each family can be distinguished from another family. The feature vectors were constructed using the TF values of the n-grams (values of the feature n-grams in an n-gram sequence) and were used to train the classification models by inputting them into supervised machine-learning methods (Decision Tree, Random Forest, K-Nearest Neighbour, Naive Bayes, and Gradient Boosting Decision Tree) to perform Ransomware classification[25]. Similarly Wan et al. carried out feature selection based Ransomware detection with machine learning. They used 36 features e.g. IdleTime and the selection algorithms were selected based on six feature correlations: gain ratio, information gain, correlation ranking, OneR feature, ReliefF ranking, and symmetrical. The data was labelled either locky (abnormal), cerber

(abnormal) or normal and then was trained. The data was then input into a decision tree[26].

There are many types of features one can select from Ransomware. For example, behaviours and file usage patterns[27], API calls and key management (keys are either generated on victims machine, downloaded from C&C server or embedded in the binary[28]). However Mehnaz et al.[27] said that there are several mechanisms for malware detection but a few for Ransomware. They claim that analysing Ransomware for file usage patterns or behaviours results in delayed detections and “monitoring only the process activities ... or file changes is not sufficient for effective detection”. This is because it yields high false positive and false negative results. This suggests that a combination of features should be selected to achieve the most accurate results.

2.5 Machine Learning Focusing On Clustering Report

2.5.1 Standardisation

After features are selected from behavioural profiles (feature selection) (see previous report) and represented in a suitable form (model construction) and before they are manipulated by a machine learning algorithm such as a clustering algorithm, the features be scaled (feature scaling). This is because many machine learning estimators work with features between 0 and 1 and they may behave unexpectedly if the input data is not normally distributed. Also data can become inconsistent when measuring the similarity, e.g. Euclidean distance, between features with different units and all features should contribute equally or if features have the same units but show different variances. If one feature is measured on a larger scale compared to the others, the similarity metric will be influenced more by that larger scaled feature. For example if one compared 2 features which vary differently due to their scales such as height (meters) and weight (kilograms) which varies more, then PCA may “determine that the direction of maximal variance closely corresponds more with the feature that varies more (weight) if the features are not scaled”, even if a change in height is considered more important[29]. Below (Figure 2.1) are two graphs. The one on the left shows the effect of a training dataset undergoing PCA without standardisation so the orders of magnitude are not the same. For example “feature 13 is two orders of magnitude above the other features”. The graph on the right shows the effect of a training set undergoing PCA after standardisation so the orders of magnitude are mostly the same. The prediction accuracy in the scaled dataset outperforms the unscaled version at 8.15% compared to 81.48%[30].

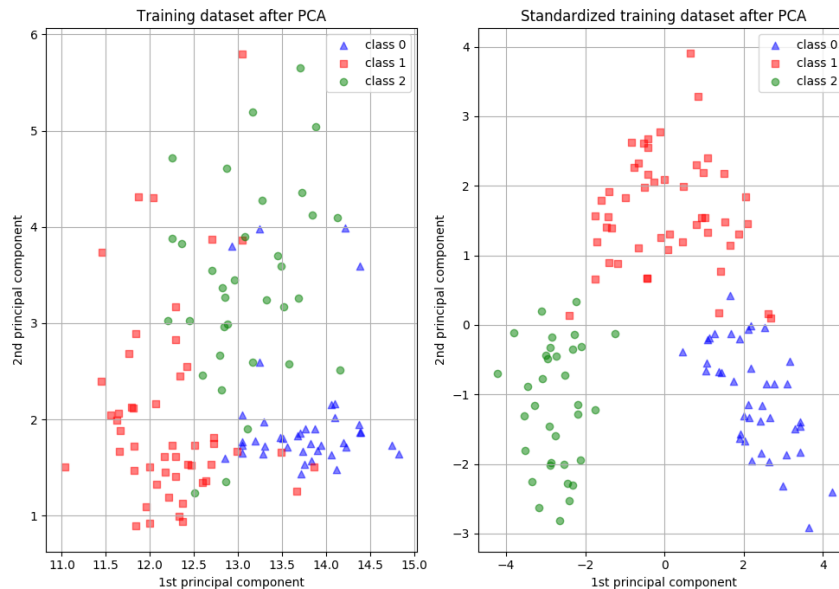


Figure 2.1: Two graphs (one without standardisation (left) and one with standardisation (right)) that illustrates the importance of scaling[30].

This shows that all features should be standardised and have “properties of a standard normal distribution with $\mu = 0$ and standard deviation from the mean $\sigma = 1$ ”. The standard scores (or z-scores) of the samples is calculated by $z = \frac{x-\mu}{\sigma}$ where x is a data point, μ is the mean calculated by $\mu = \frac{1}{N} \sum_{i=1}^N (x_i)$ and σ is the standard deviation calculated by $\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$. Alternatively, Min-Max scaling (normalisation) can be used where the features are scaled to 0 and 1. This is calculated by $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$.

Standardisation and normalisation are different. Normalisation scales features to have values between 0 and 1 but standardisation transforms features to have a mean of 0 and a standard deviation of 1. The choice of method depends on the application for example in clustering standardisation should be used to compare feature similarity based on distance measures[29].

Once features have been normalised or standardised, machine learning techniques can be used to classify/cluster the datapoints. Machine learning is a set of mathematical techniques for “information mining, pattern discovery and drawing inferences from data”. It is the process of using an algorithm to scan historical data (e.g. logs, emails or data from previous attacks) and find the best classification rule (data is analysed for patterns to indicate malicious attacks and then input into an algorithm that outputs malicious or legitimate) based on a mathematical definition of “best”[2].

Due to huge amounts of data being available now compared to in the past (big data) and the rise in computational capacity and tools available to process and analyse data in real time, one cannot use traditional techniques or humans to analyse information. People now use machine learning techniques to do this and instead of humans manually analysing the data to look at trends and patterns, they now direct, develop and harness machine learning and interpret results[4]. Machine learning techniques are a good way of recognising trends and patterns in the data. It can be used in many domains such as the security domain for example to help identify trends and meaningful patterns to help predict, defend and respond to attacks in the future. Machine Learning and AI will not only make cyber-attacks more powerful but will also make cyber defence just as powerful. It can be used in many other domains such as the health sector or the financial sector for example to help insurers to provide digital, personal and relevant information e.g. more focused sales and marketing and reduced operational costs[5]. This is why machine learning techniques are so important. It replaces human jobs which have now become impractical, less accurate (e.g. financial services model using machine learning techniques to produce GDP estimates have said it had been very accurate for the last 16 years[6]), less time consuming to carry out and used in many domains.

There are two types of machine learning, supervised and unsupervised. In supervised learning, the model for predicting an output (label on future data[2]) is based/predicted on one or more inputs (labelled dataset of historical data[2])[3] and its goal is to learn a mapping from inputs to outputs such that if x is the inputs and y is the outputs then y is an element from the number of classes ($y \in \{1, \dots, C\}$ where C is the number of classes)[3]. Techniques include cross-validation, train/validate/test and out-of-time validation[2]. Real world applications include document classification, email spam filtering, classifying flowers, image classification, handwriting recognition and face detection and recognition[1]. In unsupervised learning, relationships and structure or interesting patterns can be learnt from the data[3]. It is unknown what the patterns and the desired outputs are for each input (no labels on historic dataset[2]), and there are no obvious error metrics to use compared to supervised where you can “compare the prediction of y for a given x to the observed value”[1]. Unsupervised learning methods are useful because labelling may not exist and labelling large datasets is expensive.

2.5.2 Types of Machine Learning

Supervised Learning

Supervised Learning can be split into classification which finds a model that separates instances into classes and regression which tries to generalize and predict real-valued numbers[1]. Classification can be binary (2 classes such as malicious or legitimate) or multi-class. Decision trees are data structures used to make decisions. It can be used for regression or classification and can

take as input numerical or categorical data without normalisation. It works by recursively splitting up nodes of the tree (a binary decision) into children based on conditions (metrics measure the quality of the split such as gini impurity, variance reduction and information gain) e.g. if the condition is “age \geq 18”, it is split into 2 children “age $<$ 18” and “age \geq 18”. It is easily explainable as every prediction (output) can be “expressed as a sequence of boolean expressions from the root node to a leaf node”, scales well and is efficient for training and predicting. However, there are problems with overfitting and generalisability (not generalise beyond training set), they are less accurate (small changes in the dataset can make large changes in the tree), biases exist towards variables with more possible values when splitting categorical values and they are inefficient with some types of relationships. Many decision trees can be combined into random forests. Here overfitting is reduced by taking the average of all decision trees, the algorithm can be parallelised, but it is storage intensive[2].

Support Vector Machines (SVM's) are a supervised learning classification algorithm where the data points (features) are separated by a margin (smallest distance between the hyperplane (boundary) and any of the samples). SVM tries to maximise the margin by taking into account the data points which are closest to the hyperplane. This means that the location of the hyperplane is not influenced by the data points that are further away i.e not support vectors[31]. They have good performance, scale well in dimensional space and are efficient but the outputs cannot be interpreted as probabilities, so extra computation and cross validation is needed[2]. Takeuchi et al.[32] used SVM's to detect Ransomware via the API calls with the view of detecting unseen Ransomware. A vector (e.g. 2-grams which were frequency or bit vectors) of the number of individual existing Ransomware API calls e.g. read or write in the execution logs of malicious programs is created (feature selection and model construction), then a standardised vector representation model is created to accommodate the diversity of programs, then a SVM is used which learns the features of malicious behaviour using the vector representation model and classifies unseen Ransomware into benign or malicious behaviour.

Linear regression predicts real numbers (future outcomes) based on feature vectors of past historical data. A best fit linear model is produced where the outputs (real numbers) are mapped onto the inputs (feature vectors). Logistical regression predicts the probability of each data point occurring from numerical feature vectors. It is efficient, scales well and is explainable as the contribution of each feature can be calculated) but it assumes each feature is linear, features should have “little or no multicollinearity” (if independent then truly independent) and requires a large sample size. Anomaly detection extends regression which tries to determine detect an unusual event when comparing an observed and predicted value[2].

Other supervised learning methods include naive bayes classifier, k-nearest neighbour and neural networks[2].

Unsupervised Learning

Unsupervised learning can be split into Principal Component Analysis (PCA) and Clustering. PCA according to James et al.[3] is “used for data visualization / data pre-processing before supervised clustering”. It is used for dimensionality reduction and looks to project high-dimensional data to a low-dimensional subspace[1][31] to explain a good fraction of the variance[3]. The dimensionality of the data needs to be reduced because even though the data is high-dimensional, there may only be a small number of degrees of variability from latent factors[1]. PCA has been used in areas such as biology, natural language processing, signal processing, computer graphics[1], data compression, pre-processing and visualisation[31].

Clustering[3] tries to group similar objects together by looking at whether the “observations fall into distinct groups” based on the inputs. A cluster is a group of data points where the distance between each point is small (objects in the same cluster are as similar to each other as possible) and the distance between points in different clusters are large (objects in different clusters are as dissimilar to each other)[31][33][34]. High intra-cluster similarity/ low within cluster variability (commonalities between members of the same class) and low inter-cluster similarity/ high between cluster variability (differences between members of different classes). The greater the similarity within a cluster and the greater the difference between clusters, the better and more distinct the clustering is[35]. Clustering can be used in many areas including psychology, economics[33],

bioinformatics, image analysis, data mining[34], medicine, summarisation, compression, business and climate[35].

Euclidean Distance is a dissimilarity metric between the features[1]. Good clustering occurs when the “within-cluster variation (amount where each feature in a cluster differs from one another) is as small as possible”[3]. A common dissimilarity measure is squared Euclidean Distance[36][1] which is the distance between 2 points x_{ij} and $x_{i'j}$, calculated by $\Delta_f(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$ [1]. It finds the square of the distance between each variable and sums the squares[34]. This may become slow as the Euclidean Distances are computed for every data point. Many proposals have been made for speeding up the algorithms such as precomputing a data structure which stores all data points that are close to each other e.g. a subtree that contains all nearby points[31].

There are two main types of clustering; these are hierarchical and partitional clustering. Partitional clustering is where all objects in a dataset are divided into non-overlapping clusters and each object is only assigned to one cluster. One type of partitional clustering is K-means clustering. In K-means clustering, the features from a dataset are partitioned into a “pre-specified number of clusters”[3] where each feature is assigned to exactly one cluster[34] (hard assignment[31]). Each feature is assigned to a cluster if its centre (centroid) is the nearest to that cluster[34]. This means that every pair of clusters are distinct[33] because if the indices of the features in each cluster denote a set of $\{C1, \dots, Ck\}$ where k is the number of clusters, then a union of the indices of the features create a set of all the indices of 1 to n ($C1 \cup C2 \cup \dots \cup CK = \{1, \dots, n\}$). Clusters are non-overlapping so no feature belongs to more than one cluster. If the indices of the features in each cluster denote a set of $\{C1, \dots, Ck\}$ where k is the number of clusters, then the intersection of the features in 2 different clusters will produce the empty set ($Ck \cap Cki = \emptyset$ for all $k \neq ki$)[3].

For example below (Figure 2.2) K-means clustering has been used to cluster “150 observations into 2 dimensions using different values of K”[3].

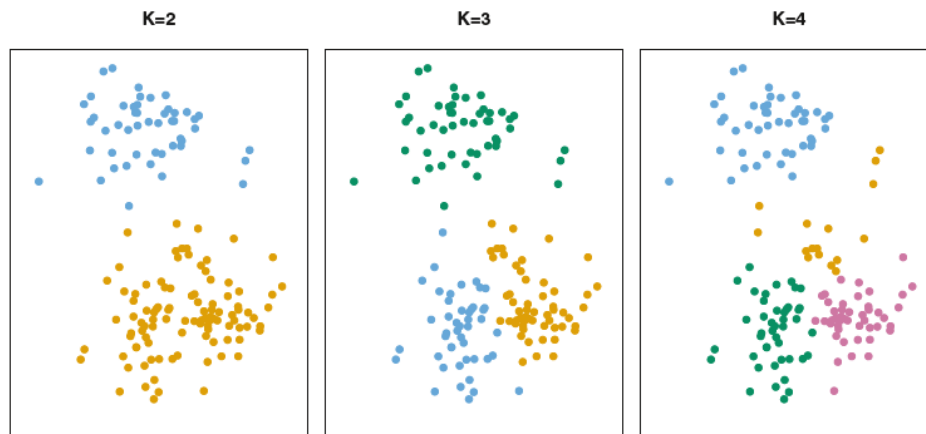


Figure 2.2: This graph illustrates k-means being performed using different k values ($k = 2, 3$ or 4)[3].

James et al.[3] provide an algorithm to partition n features into k clusters:

1. “Assign a random number, from 1 to k , to each feature.
2. Iterate until the cluster assignments stop changing.
 - (a) For each of the k clusters, compute the cluster centroid (mean of the features assigned to each cluster) for the features in the k th cluster.
 - (b) Assign each observation to the cluster whose centroid is closest (closest is defined by Euclidean Distance).”

However, there are some issues with K-means clustering. This algorithm finds a local rather than a global optimum so the results will depend on the initial cluster assignment in step 1 of

the algorithm[3][33]. This means that the algorithm should be run multiple times from different random initial configurations, until the partitioning of the features into k clusters has the smallest within-cluster variation[3]. Another problem is determining the number of clusters k . There are many methods such as $\sqrt{\binom{n}{2}}$ where n is the number of objects (Madhulatha[34]) or the elbow criterion. Choosing the value of k is hard. Different values can result in different models therefore the analysis should be run several times, randomizing the number of clusters until an optimal clustering is found. To find the optimal clustering, calculate the Sum-of-Squared-Error (SSE) for each cluster and pick the clustering with the smallest SSE[35]. It is computationally fast and is simple to understand, however it is difficult to identify and predict the value of k as it is fixed at the start of the algorithm and the final clusterings depend this value[37]. K-means does not work well on categorical or binary features. This is because when mapping into vector space the values do not make sense and with binary features one feature can dominate and determine the cluster or its information may be lost. It is less effective in high dimensions due the curse of dimensionality (the more features used, the lower the predictive performance) therefore the dimensionality should be reduced e.g. PCA or using k-means in low dimensions[2]. It has been used in the real world for image segmentation and compression[31].

The other main type of clustering is hierarchical clustering. This organises the set of clusters as a dendrogram (tree-like representation), showing the clusterings for each possible number of clusters[35]. This is used when the number of clusters is unknown as there is no pre-specified number of clusters[3] and it only requires a similarity measure. Despite this it is slow and membership to a cluster is fixed. In contrast K-means requires stronger assumptions and the number of clusters and initial centres are defined at the start[37].

There are 2 types of hierarchical clustering, bottom-up (agglomerative) clustering and top-down (divisive) clustering. In bottom-up clustering, it starts from the maximum number of clusters and works down towards a single cluster by splitting up the clusters[33][34][35]. A “dendrogram is built from the leaves and combines clusters up to the trunk”[3]. Each leaf is a feature and as you move up the dendrogram, branches start fusing (merging) together to the feature similarity. The height of the branches i.e. fusion (on the y axis) denotes how different the features are; the lower (earlier) the fusion takes place, the more similar the groups of features are to each other. The higher the fusion occurs, the more different the features are. Below (Figure 2.3) is an image which shows how to interpret a dendrogram of “9 features in two-dimensional space from the raw data”[3].

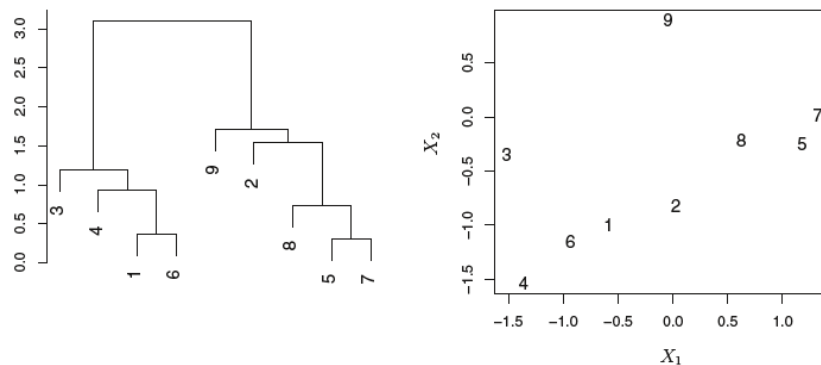


Figure 2.3: This graph illustrates the interpretation of a dendrogram (left) from raw data (right)[3].

To identify clusters, make a cut (line) horizontally through the dendrogram which splits the features into different clusters[3][1].

Below (Figure 2.4) is an image that shows this process. If the dendrogram was cut at height 2 and then the clustering $\{\{4, 5\}, \{1, 3\}\}, \{2\}$ would be obtained[1].

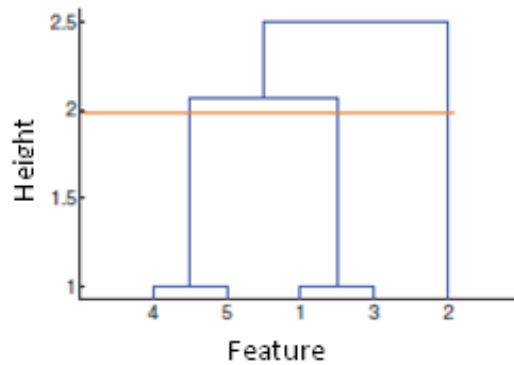


Figure 2.4: This graph illustrates a dendrogram cut at height 2[1].

James et al.[3] and Murphy[1] provide an algorithm to obtain a dendrogram where initially all individual points are clusters, and the 2 closest clusters are merged until only one cluster remains[35]:

1. Begin with n groups and start at the bottom of the dendrogram.
2. Iterate for $i = n, n - 1, \dots, 2$.
 - (a) Examine all pairwise inter-cluster dissimilarities among the i clusters via the Euclidean Distance of all the $n(n - 1)/2$ pairwise dissimilarities. Identify the pair of clusters that are most similar and fuse (merge) the 2 clusters together. The dissimilarity between these two clusters indicates the height in the dendrogram where the fusion should be placed.
 - (b) Compute the new pairwise inter-cluster dissimilarities among the $i - 1$ remaining clusters until all of the observations belong to one single cluster i.e. there is one single group.

The linkage defines the dissimilarity between two groups of features. There are different types of linkages:

1. Complete - "Maximal intercluster dissimilarity"[3] which is the maximum distance between the objects in the cluster[33]. This is known as Furthest Neighbour clustering[1] where all pairwise dissimilarities are computed between the observations in cluster A and the observations in cluster B, and the largest is recorded[3]. Two groups are close only "if all of the observations in their union are relatively similar". Its complexity is $O(n^3)$ [1].
2. Single = "Minimal intercluster dissimilarity"[3] which is the minimum distance between the objects in the cluster[33]. This is known as Nearest Neighbour clustering[1] where all pairwise dissimilarities are computed between the observations in cluster A and the observations in cluster B and the smallest is recorded[3]. The tree generated is a minimum spanning tree and clusters are merged by connecting the two closest features of the clusters together. Its complexity is $O(n^2)$ [1].
3. Average = "Mean intercluster dissimilarity"[3] which is the average distance between the objects in the cluster[33]. This is the preferred method[1] where all pairwise dissimilarities are computed between the observations in cluster A and the observations in cluster B, and the average is recorded[3]. As averages are used, any change to the measurement scale can change the result. Its complexity is $O(n^3)$ [1].
4. Centroid = "Dissimilarity between the centroid for cluster A and the centroid for cluster B"[3]. The centroid of a cluster is the average of the objects in the cluster. "Each feature of the centroid vector is the average feature value of the vectors of all objects in the cluster"[33].

Average and complete produce more balanced dendrograms. Centroid linkage can cause inversion where two clusters are fused at a height below one of the individual clusters[3].

This clustering ranks the objects so that it is easier to display objects. The algorithm is flexible as the number of clusters is not fixed and so can be chosen dynamically. Small clusters are produced so they are easier to understand and analyse. However, if an incorrect clustering is obtained at an early stage then objects cannot be relocated later and results can vary depending on the dissimilarity measure used[37].

The other type of hierarchical clustering is top-down (divisive) clustering. This starts from a single cluster and works up towards the maximum number of clusters by merging the closet pairs of clusters[33][34][35]. Below is an algorithm which “starts with all the data in a single cluster, and then recursively divides each cluster into two daughter clusters, in a top-down fashion”[1]:

1. Starts with all the data in a single cluster $D = \{x_1, \dots, x_N\}$ where “ D_i is the set of the data points at the leaves of a subtree T_i ”.
2. Compute the probability of each data point in the cluster given the tree T_i where $i = 1$ to N ($p(D_i|T_i)$).
3. Iterate until all clusters are merged. For each pair of clusters i and j , compute the probability of the merge of D_i and D_j given the merged tree T_{ij} ($p(D_{ij}|T_{ij})$).
4. Find the pair D_i and D_j that have the highest merge probability and merge them ($D_k = D_i \cup D_j$) and delete D_i and D_j .

This clustering focuses on the top levels of the dendrogram and at every stage the algorithm will have access to all the data so the best solution should be achieved. However, difficulties in computation when splitting up the clusters can arise and like bottom-up clustering, the results vary based on the dissimilarity measure used[37]. It can work with any distance metric or similarity function but can be more complex to analyse, has high time complexity (see above) so is inefficient with large datasets and due to the tree, it consumes more storage compared to k-means[2].

Other unsupervised techniques include locality-sensitive hashing, K-D trees and Density-Based Spatial Clustering of Applications with Noise (DBSCAN)[2].

An issue with clustering is that it could be affected by the order of features in the data. To overcome this, the analysis should be carried out several times, randomizing the order of the features and averaging the cluster centres. Another issue is that clusters maybe inappropriate as the clustering algorithms try and assign everything into clusters. These clusters may become distorted due to having outliers that do not belong to a cluster. Also “some algorithms are not robust to perturbations of the data”[3].

An example where clustering has been used in classifying malware is in Hamid et al.[38]. They used k-means clustering to separate the android malware into ransomware, scareware or goodware depending on 2 features, presence or absence of locks and encryption. If the trace had a lock and encryption then it was classified as ransomware, if the trace had a lock but not encryption then it was classified as scareware and if the trace did not have both a lock and encryption then it was classified as goodware. It was performed on two datasets with one having accuracy at 98.1% and the other at 74.7%. This means that clustering malware can be accurate but it can be made more accurate depending on the number features used. Here only 2 features were used (presence or absence of locks and encryption), so in theory the more features used the more accurate the classification will be.

2.5.3 Resiliency and Trustworthiness of Machine Learning Algorithms

Chen et al.[39] looked at the resiliency and trustworthiness of machine learning algorithms for Ransomware detection via the generative adversarial network (GAN). They collected Ransomware samples from VirusTotal and used many machine learning techniques including classifiers including Text-CNN, XGBoost, Linear Discriminant Analysis (LDA), Random Forest, Naive Bayes and

Support Vector Machines (with Linear and Radical Kernels). Text-CNN was the most accurate and had the highest true positive score, F-score and AUC score. This means that it is a highly effective Ransomware classifier so will be selected by a security defender. They tested resilience of high performing Ransomware detectors (combination of the above classifiers) and found that Text-CNN, LDA & Text-CNN, Naive Bayes & Text-CNN, SVM-linear & Text-CNN all failed to detect any malicious samples. They also found that the most robust classifier was SVM-radial & Text-CNN with 100% detection rate and a “weighted score between the XGBoost & Text-CNN classifier and the SVM-radial & Text-CNN classifier gives the defender a lot of coverage in the space of execution logs for Ransomware”.

2.5.4 Machine Learning Algorithm Problems

Two main problems with machine learning algorithms are overfitting and underfitting. Overfitting is when the model does not generalise well to unseen data because it thoroughly matches the training data. Here all points are correctly classified, but it is unlikely to separate new points effectively. Underfitting is when the model is too simple meaning that it does not generalise to unseen data. Here, most of the time the points are correctly classified but lots of errors can occur and it suffers from poor performance on unseen data. To minimise over and under fitting regularisation should be used in the training procedure which adds a term to the loss function such as the norm of the coefficient vector in logistic regression. Other factors to consider with machine learning is the model to use based on computational and mathematical complexity and explainability[2].

2.5.5 Validation

After clustering has been performed, the clusters obtained need to be validated and evaluated to verify that if an independent set of features was obtained, the same set of clusters would be displayed. This measures the correctness of clusterings without using external information.

The measures of cluster validity are split into cluster cohesion (how related the features in the clusters are) and cluster separation (how distinct or separate one cluster is from another)[35]. As many methods can be used the most interoperable and useful solution should be picked. One method is calculating the inter-cluster variance (gap between different clusters which should be large as dissimilar objects should be distant) and intra-cluster variance (objects inside the same cluster which should be small as similar objects should be close).

If graph-based clustering algorithms are being used e.g. hierarchical clustering, cohesion is the “sum of the weights of the links in the graph that connects points within a cluster” and the separation between 2 clusters is the sum of the weights of the links from points in one cluster to points in another cluster. $Cohesion(C_i) = \sum_{x \in C_i, y \in C_i} proximity(x, y)$ and $Separation(C_i, C_j) = \sum_{x \in C_i, y \in C_j} proximity(x, y)$. For prototype-based clustering algorithms e.g. K-means clustering, the cohesion is the sum of the proximities with respect to the centroids of the cluster and separation between 2 clusters is the proximity of the 2 clusters. There are two equations to account for the separation of clusters from an overall clustering directly relating to each other. $Cohesion(C_i) = \sum_{x \in C_i} proximity(x, C_i)$ and $Separation(C_i, C_j) = proximity(C_i, C_j)$ or $Separation(c_i, c_j) = proximity(c_i, c_j)$. For Euclidean Distance dissimilarity measures, the cohesion is equivalent for both graph-based and prototype-based clustering algorithms. A cluster with a high cohesion is better than one with low cohesion.

To calculate the validity of clustering the sum of the validity of individual clusters is used[35]. $Overall\ validity = \sum_{i=1}^K (w_i) validity(C_i)$. For partitional clusterings cluster validity can be measured using a proximity matrix. This looks at the “correlation between a similarity matrix for a dataset and an ideal version of a matrix based on the cluster labels from a cluster analysis of the dataset”. All points within a cluster have a similarity of 1 and all points outside the cluster have a similarity of 0. “A high correlation between the ideal and actual similarity matrices show that points that belong to the same cluster are close to each other and vice versa for low correlation”. For hierarchical clusterings, cophenetic correlation can be used where the cophenetic distance between two objects is the initial proximity of objects in the same cluster. The cophenetic corre-

lation coefficient is the correlation between the entries of the cophenetic distance matrix and the dissimilarity matrix[35].

To evaluate the clusters obtained many metrics can be used. Firstly, purity where an empirical distribution is defined over a class of labels to calculate a result ranging from 0 (bad) to 1 (good) of individual clusters and the total clustering. Secondly rand index of two different clustering's U and V where one calculates the number of true positives ("number of pairs that are in the same clusters in both U and V"), true negatives ("number of pairs that are in the different clusters both U and V"), false negatives ("number of pairs that are in the different clusters in U but the same cluster in V") and false positives ("number of pairs that are in the same cluster in U but different clusters in V"). The result between 0 and 1 is the fraction of clustering decisions that were correct. Thirdly by mutual information between two different clustering's U and V by using "the probability that a random object belongs to cluster u_i in U and v_j in V, the probability that a random object belongs to cluster u_i in U and the probability that a random object belongs to cluster v_i in V"[1][3].

To measure the quality of a clustering which uses the Euclidean Distance dissimilarity measure, the Sum-of-Squared-Error (SSE) should be used by calculating the Euclidean Distance to the closest centroid and then compute the total SSE. If two different clusters are produced by different executions of the K-means algorithm on the same data set, then pick the one with the smallest SSE as the "centroids are a better representation of the points in their cluster"[35].

Other metrics include contingency tables that describe the association between two partitions, silhouette value which represents each cluster as a silhouette and displays the "objects which lie within a cluster and which objects are marginal to the cluster". It combines both cohesion and separation[35]. Class-based Precision and Recall can be used where the precision and recall scores are produced by "calculating the least number of object links required to align the equivalence classes"[33].

Li et al.[40] looked at the challenges of evaluating malware clustering and found that the make-up of the ground-truth data (method of selection i.e. distribution sizes of the malware families) from prior evaluations "biases their results toward high accuracy" specifically increasing the likelihood of good precision and recall. One problem with clustering is how to determine the ground-truth labels. A common method is to use an existing anti-virus tool as they use hard-coded rules however many anti-virus engines disagree on the labels therefore one should only use the labelling if it is agreed by all anti-virus tools. Also the ground truth labels and the malware to cluster should have similar cluster-size distributions e.g. both highly biased. They used the sequences of system calls and API calls of malware and performed single-linkage hierarchical clustering, precision, recall and F-measure. They concluded and proved that as the plagiarism detectors and BCHKK-algo (used by Bayer et al.[20]) performed well on the clustering (greater precision, worse recall and similar f-measure) with ground truth labels inferred by antivirus tools and was better with the BCHKK-data dataset (used by Bayer et al.[20]) compared to the VXH-data dataset (malware instances see[40]). This may be due to way the systems gathered the malware traces because they can vary in the length and composition of API sequences or due to the differences in the presence and frequencies of certain activities in each dataset. The main difference noted in the datasets were the distribution of cluster sizes as the BCHKK-data dataset is highly biased with 2 large clusters covering more than half of the malware instances. The effect of this could be seen when removing the 2 large clusters as the F-measure was lower compared to when they were in. This was also seen when introducing perturbations into the BCHKK-algo distance matrices, performing clustering and evaluating the drop of the precision and recall rate. A high precision and recall in the VXH-data case was more significant as minor errors in the BCHKK-algo distance matrices were more amplified (was more sensitive and less immune) when the clusters were distributed as in the VXH-data.

3

Design and Implementation for Clustering Malware

This chapter describes the design and implementation of a novel way to cluster malware. Different feature selection and model construction methods (Section 2.4), clustering methods (Section 2.5), dissimilarity metrics (Section 2.5), validation metrics (Section 2.5.5) and different aspects of the clustering process such as the importance of scaling (Section 2.5.1) are explored in the chapter above.

Section 3.1 describes my program including the description, justification and implementation of the clustering process. In particular:

Subsection 3.1.1 describes the generation of malware behavioural profiles [41].

Subsection 3.1.2 describes the different feature selection and model construction methods that were carried out. This is the process of extracting features from the behavioural profiles to produce feature vectors.

Subsection 3.1.3 describes the process of performing hierarchical clustering on the feature vectors, including the dissimilarity metric and scaling. This process combines samples that exhibit similar behaviour into the same cluster.

Subsection 3.1.4 describes the different processes of validating the clusters obtained for the different methods of feature selection and model construction to evaluate how similar the clusters were to the dataset (i.e. the accuracy of the different methods).

3.1 Description of Program and Justification of Feature Selection and Validation Methods

A summary of the process of clustering is described below:

1. Dynamic Analysis - monitors the execution of malware samples in a controlled environment. This is performed by a sandbox such as cuckoo sandbox where a file is submitted, and cuckoo sandbox produces a report (json) of the behaviour of the file when executed. The dataset[41] used in this project represents features extracted from samples analysis using static and dynamic analysis. There are 15 Ransomware families, 5 Backdoor families and 9 Trojan families with several sanitized evidences in each family. Please see the all the samples in the "Proj2/Program/" directory in my repository (<https://github.com/Rebecca22/Proj2/tree/master/Program>). This contains all the malware families and json files for each family.
2. Feature Selection - Parsing the json files and carrying out feature selection. The features from the behavioural profiles are extracted and feature vectors are produced (feature selection and model construction). The features are the operation and each json file is represented as a bit or frequency feature vector (n-grams) with each dimension representing a system call. Three types of N-grams were used; 1-gram where one system call is a dimension, 2-grams where pairs of system calls are dimensions and 3-grams where triples of system calls are dimensions. A bit vector is when each dimension is 1 if the system call is observed or 0 if the system call is not observed and a frequency vector is when each dimension has the number of system calls observed in a trace. I have used 2 feature selection methods; these are full representation or by category. There are 12 experiments run all together (2 experiments per system call representation for bit and frequency vectors per N-gram = $2 * 4 * 3 = 24$). This is because for each N-gram type, of which there are 3, there are 2 system call representations to be represented and each vector is a bit or frequency vector.
3. Hierarchical Clustering - Clustering the vectors according to the behavioural profile where samples that exhibit similar behaviour are combined in the same cluster. The process includes, deciding a dissimilarity matrix, scaling the feature vectors and hierarchical clustering. The dissimilarity metric I used was squared Euclidean Distance (finding the square of the distance between each point and summing the squares) and scaling is carried out on the frequency feature vectors by standardisation (mean of 0 and standard deviation of 1) and centred around the norm. Hierarchical clustering produces a dendrogram constructed by identifying the two feature vectors that were the most similar (Euclidean Distance is the smallest) and merging them together.
4. Validation - Cut each of the dendrograms at each height to obtain clusters which are validated using metrics to find the best clustering. The clusters are compared to the labelled dataset used for the samples analysis. The evaluation metrics Fowlkes Mallows Score (FMS), F1-Score (F1), Adjusted Rand Index Score (ARI) and Silhouette Coefficient (SC) were used to evaluate how similar the clusters were to the dataset. The cut that was most similar to the labelled data set was the best clustering (i.e. the one that yielded a score the closest to 1.0). For Ransomware the best clustering method was the Uni-gram with the system call representation of Category and vector representation of Frequency Vector. For Backdoor, the best clustering method was the Di-gram with the system call representation of Full Representation and vector representation of Frequency Vector. For Trojan, the best clustering method was the Tri-gram with the system call representation of Category and vector representation of Frequency Vector.

Please see Appendix A for the installation requirements and user manual for my program. I used a git repository (which was used to run on google colabatory) to store the code which only produces and downloads all the text files for running the individual experiments for one malware family at a time (please see <https://github.com/Rebecca22/Proj2> for the code that produces all the text files). **Please note:** The text files were too large to store on git hub which meant that I could not run my best cluster algorithm to find the best clustering using the text files according to the FMS, F1, ARI and SC scores. This meant only the code for the production of the text files is on

github, all the rest of the program code is stored on my local machine as it is too large to store on github.

3.1.1 Stage 1 - Dynamic Analysis:

The dataset[41] used in this project represents features extracted from samples analysis using static and dynamic analysis meaning that many features were collected. Static analysis involved features such as file sections and syscall/API and dynamic analysis involved features such as contacted IP addresses and AV signatures. Both static and dynamic analysis involved sandboxes e.g. cuckoo to form ontologically homogeneous blocks called Analyses Results and then MIST translation (Malware Instruction Set for Behaviour Analysis) removed all the irrelevant and redundant features. It is needed due to the curse of dimensionality; the more features used, the lower the predictive performance. It reduces overfitting, improves accuracy and reduces training time[21]. This is more effective than the individual behavioural profiles themselves as the behavioural profiles will contain a lot of information that is irrelevant to the behaviour. MIST[42] encodes instructions as the category of the system call, operation of the system call and arguments for the system call. The dataset used in this project was based on the MIST dataset but without encoding the operation. Sanitised techniques were used to hash the evidences such that the algorithm speed was increased but the meaning was unaffected[41].

In each json file for each family, "properties" lists the action performed within the category of system calls and then the sanitised evidences that perform this category action. For example "file_access: 40770804 f5ddaf0c 3d801aa5" means that the category is file, action is access and there are 3 sanitised evidences of 40770804 f5ddaf0c 3d801aa5 where the sample accesses a file[41].

I used this dataset because it was produced recently (2016), therefore it is relevant to today's world and can be generalised to today's problems (as the features in the dataset will be similar to the features in current malware). It looked realistic as everything was described in detail (see[41]) so it had undergone rigorous processing, it was sanitised and as both static and dynamic analysis were used, many features were collected and many evidences were produced. All these properties made this dataset reliable, representative of today's malware and accurate/valid as my program could be based on many features and evidences that exist in current malware, and that is why I used this dataset.

Please see the all the samples in the "Proj2/Program/" directory in my repository (<https://github.com/Rebecca22/Proj2/tree/master/Program>). This contains all the malware families and json files for each family.

3.1.2 Stage 2 - Feature Selection and Model Construction:

I carried out feature selection before clustering to produce a better clustering where the operations from the json files are used to represent the features in the feature vectors. The other information in the json file is irrelevant and should not be used to construct the feature vectors as they do not contribute to the predictive accuracy. This increases performance and decreases storage requirements. Redundant features do not provide any extra information to the system calls and can produce anomalies and inconsistencies within the vectors. The removal of irrelevant and redundant features is needed due to the curse of dimensionality; the more features used, the lower the predictive performance. It reduces overfitting, improves accuracy and reduces training time[21].

I used two feature selection methods to extract the system calls from the behavioural profiles in the json files. Firstly, category where each system call was represented by only the name of the category of the operation e.g. file, pe, reg. Secondly, full representation where everything is represented (i.e. the category and action of the operation) e.g. file_access, file_delete, pe_imports, reg_read. This enriched the feature space. This was used because the first method groups together all operations into a category such as all the operations of file_access, file_delete, file_read, file_write and file_drop were grouped under the category file. However, this does not fully distinguish each evidence and may not be able to distinguish between different Ransomware, Backdoor or Trojan families therefore a greater system call representation was

needed (second experiment) to fully represent the actual behaviours. This meant more information needed to be expressed to fully represent the behaviour of each evidence within each Ransomware, Backdoor or Trojan family. The first method of feature selection has the largest feature space and the least semantics whereas the last method has the smallest feature space but the most semantics.

After these features were extracted from the behavioural profiles, they were used to construct feature vectors for each .json file. Each json file is represented as a feature vector where each dimension in a vector is a system call. The dimensions of the vectors in the feature space are worked out by looking through the whole dataset of samples and counting how many unique system calls are in the samples (feature selection). The features (operations) from the json files were constructed as feature vectors (model construction).

N-grams were to produce feature vectors. These are consecutive n-sequences of system calls that are observed in the trace. Three methods of model construction were used. Firstly 1-gram (Uni-gram) where one system call is a dimension in the feature vector e.g. if the system calls in the json files were pe, str and sig then the dimension of the 1-gram would be {pe, str, sig}. Secondly 2-grams (Di-grams) where pairs of system calls are dimensions in the feature vector e.g. if the system calls in the json files were pe, str and sig then the dimensions of the 2-gram would be {pe pe, str str, pe str, str pe, sig sig, pe sig, sig pe, str sig, sig str}. Thirdly, 3-grams (Tri-grams) where Triples of system calls are dimensions in the feature vector e.g. if the system calls in the json files were pe, str and sig then the dimensions of the 3-gram would be {pe pe pe, str str str, pe pe str, pe str pe, str pe pe, sig sig sig, pe pe sig, pe sig pe, sig pe pe, pe str str, str pe str, str str pe, pe str sig, pe sig str, str pe sig, str sig pe, sig pe str, sig str pe, pe sig sig, sig pe sig, sig sig pe, str str sig, str sig str, sig str str, str sig sig, sig str sig, sig sig str}.

Finally, these N-grams would either be represented as a bit vector or a frequency vector. In a bit vector the feature vector will have a 1 if the feature (system call) is present in the behavioural profile or 0 if the feature is not present in the behavioural profile. For example in the 1-gram above, {1, 0, 1} means that in the behavioural profile pe and sig operations were observed in the trace but the str operation was not observed in the trace. In frequency vectors, the feature vector will contain the number of system calls observed in the traces per json file per family for example in relation to the 1-gram above, {6, 0, 1} means that in the behavioural profile 6 pe, 0 str and 1 sig operations were observed in the trace. From Uni-grams to Tri-grams the feature vector dimensions (feature space) decrease but the semantics of the vectors increase.

Twelve experiments were used for all the methods of feature selection (system call representation) and model construction (feature vector representation) to find the best clustering. As this is unknown, lots of experiments need to be run and then evaluated against the labelled dataset provided by Ramilli[41] to obtain the best clustering.

3.1.3 Stage 3 - Hierarchical Clustering:

Firstly, the dissimilarity matrix of Squared Euclidean Distance was used. This is the distance between two points calculated by finding the square of the distance between each point and summing the squares. The smaller the distance between the two features, the similar the feature vectors are to each other. This means the within-cluster variation (amount where each feature in a cluster differs from one another) is small and the between-cluster variation (amount where each feature in different clusters differs from one another) is large. This metric was used because it is one of the most common metrics and the Hierarchical clustering package in the “scipy” library supports Euclidean Distances (scipy.cluster.hierarchy).

The Euclidean Distance between two feature vectors $x = [x_1, x_2, \dots, x_n]$ and $y = [y_1, y_2, \dots, y_n]$ is calculated by: $\sqrt{[y_1 - x_1]^2 + [y_2 - x_2]^2 + \dots + [y_n - x_n]^2}$

For example if the feature vector had dimensions: file_access, file_delete, sig_persistence_autorun, sig_copies_self, file_write, file_read} and two files had feature frequency vectors {6, 1, 6, 3, 3, 1} and {2, 1, 0, 0, 0, 2} respectively then after standardisation (above) the feature vectors are {-1. 0. 1. 1. 1. -1.} and {-1. 0. -1.

-1. -1. 1. }. The Euclidean Distance is 4.472135955, calculated by $\sqrt{[-1 - 1]^2 + [0 - 0]^2 + [1 - -1]^2 + [1 - -1]^2 + [1 - -1]^2 + [-1 - 1]^2} = \sqrt{[-2]^2 + [0]^2 + [2]^2 + [2]^2 + [2]^2 + [-2]^2} = \sqrt{(4 + 4 + 4 + 4 + 4)} = \sqrt{20} = 4.472135955$

Secondly, frequency feature vectors were scaled. The bit vectors did not need to be standardised, as all the dimensions in the feature vector were either 1 (system call present) or 0 (system call not present). This means the number of json files will not affect the vector dimensions and so even if one sample has more json files than another sample, the clustering algorithm will not be affected or influenced more by that sample. If one feature vector is measured on a larger scale than the other feature vectors e.g. weight vs height, it will be influenced by that feature e.g. weight. If frequency feature vectors were not scaled, they may become inconsistent when calculating the Euclidean Distance similarity measure and many machine learning algorithms work with features between 0 and 1. All the values in the feature vector were the frequency of the system calls therefore the number of json files will directly affect the dimensions of the feature vectors. If one sample has more json files than another sample, the dimensions for the first sample will have greater values for system calls compared to the second sample and so the Euclidean Distances between the features and the clustering algorithm will be affected and influenced greatly by that sample. The larger the number of json files in a sample, the larger the number of values of system calls observed in the trace, and the more the clustering algorithms and Euclidean Distances will be affected.

Standardisation should be used in clustering to compare similarities between features[29]. For illustration purposes, below is a graph (Figure 3.1) generated from two vectors {165, 134, 40, 37, 46, 0} and {146751, 2193, 750, 814, 5832, 501} respectively. If they were both standardised and plotted on a graph, all the points were around the same value (green) and so the Euclidean Distances were not be affected by larger numbers in the vector dimensions and if the vectors were not standardised then the values were all over the place (red) and were affected by larger numbers in the vector dimensions.

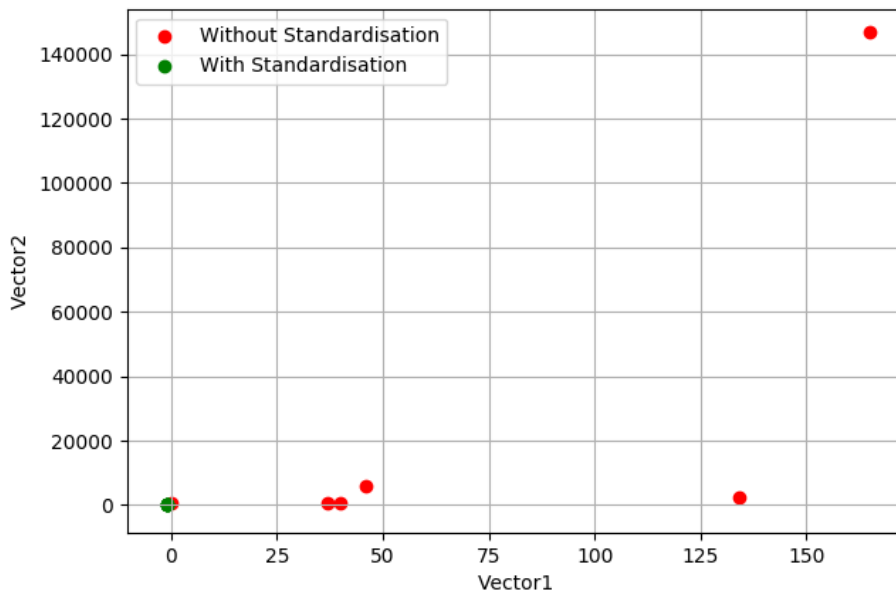


Figure 3.1: This graph illustrates the difference between standardised (red) and non-standardised (green) vectors. Standardised points are all located around the same value whereas non-standardised are not.

Standardisation makes the mean = 0 and the standard deviation = 1. The mean is the average of the system call frequencies amongst the dataset and the standard deviation is how much the system call frequencies of the different feature vectors (each malware file for each family) differ from the mean value of the group. First the means and standard deviations are calculated and

then these are used for centering and scaling. Its distribution will have a mean of 0 and standard deviation of 1.

The mean is calculated by: $\frac{1}{N} \sum_{i=1}^N (x_i)$ [29]. In a maxtrix the mean is computed as: $1'x(1'1)^{-1} = 1'x(1/n)$ where $1 = nx1$ column vector of ones and x is a $nx1$ column vector of scores x_1, x_2, \dots, x_n and $n =$ number of dimensions. For example if the feature vector had dimensions: `file_access`, `file_delete`, `sig_persistence_autorun`, `sig_copies_self`, `file_write` and `file_read`. If two files had feature vector $\{6, 1, 6, 3, 3, 1\}$ and $\{2, 1, 0, 0, 0, 2\}$ respectively then the mean would be 4 for the first dimension. This was calculated by $[11] \binom{6}{2} ([11] \binom{1}{1})^{-1} = 6 + 2 * (1 + 1)^{-1} = 8 * 2^{-1} = 8/2 = 4$. The standard deviation calculated by: $\sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$ [29]. In a maxtrix the standard deviation is computed as: $1'x(1'1)^{-1} - xi = 1'x(1/n) - xi$ (i.e. mean – value of column vector for the files) where $1 = nx1$ column vector of ones and x is a $n \times 1$ column vector of scores x_1, x_2, \dots, x_n and $n =$ number of dimensions. For example with the same feature vector described for the mean (above), the standard deviation would be -2 for the first dimension for the first file. This was calculated by mean – first value of column vector for the file = $[11] \binom{6}{2} ([11] \binom{1}{1})^{-1} - \binom{6}{2} = 4 - 6 = -2$.

Once the Euclidean Distances have been calculated for all the samples and the frequency vectors have been standardised, hierarchical clustering is carried out by producing a dendrogram of all the samples against Euclidean Distances. The dendrogram is produced by repeatedly identifying the two samples that are the most similar (ie the Euclidean Distance is the smallest) and merging the 2 clusters together until all feature vectors belong to one single cluster.

For example if the feature vector had dimensions: `{file_access, sig_antiaav_servicestop, file_delete, sig_persistence_autorun, reg_read, sig_copies_self, file_write, file_read, reg_access}`. If there were four files which had feature frequency vectors $0 = \{2, 0, 0, 0, 0, 0, 0, 0, 0\}$, $1 = \{6, 0, 1, 6, 0, 3, 3, 1, 0\}$, $2 = \{0, 1, 0, 0, 0, 0, 0, 0, 0\}$, $3 = \{2, 0, 1, 0, 0, 0, 0, 2, 0\}$, $4 = \{1, 0, 0, 0, 1, 0, 0, 1, 1\}$ respectively. The feature vectors are transformed by its distribution having mean of 0 and standard deviation of 1 (see above on how the mean and standard deviations are calculated). The transformed feature vectors are: $0 = \{-0.098 \ -0.5 \ -0.816 \ -0.5 \ -0.5 \ -0.5 \ -0.5 \ -1.069 \ -0.5\}$, $1 = \{1.863 \ -0.5 \ 1.225 \ -0.5 \ 2. \ 2. \ 0.267 \ -0.5\}$, $2 = \{-1.079 \ 2. \ -0.816 \ -0.5 \ -0.5 \ -0.5 \ -0.5 \ -1.069 \ -0.5\}$, $3 = \{-0.098 \ -0.5 \ 1.224 \ -0.5 \ -0.5 \ -0.5 \ -0.5 \ 1.604 \ -0.5\}$, $4 = \{-0.588 \ -0.5 \ -0.816 \ -0.5 \ 2. \ 0.5 \ -0.5 \ 0.267 \ 2.\}$

Next the Euclidean Distances are calculated for all the samples (see above on how to calculate it) and for every iteration the vectors that have the smallest distance are merged together. The matrix below (where each row is in the format `[vector1, vector1, dist, sample_count]`) shows us which feature vectors were merged in each iteration. For example, the top row shows that the cluster 0 and 2 were merged as they were the most similar with a Euclidean Distance of 2.68543078.

```
[[0 2 2.68543078 2]
 [3 5 3.36296355 3]
 [4 6 3.81131197 4]
 [1 7 4.93779993 5]]
```

Finally, a dendrogram is constructed from this matrix. For illustration purposes, this matrix looks like this on a graph (Figure 3.2):

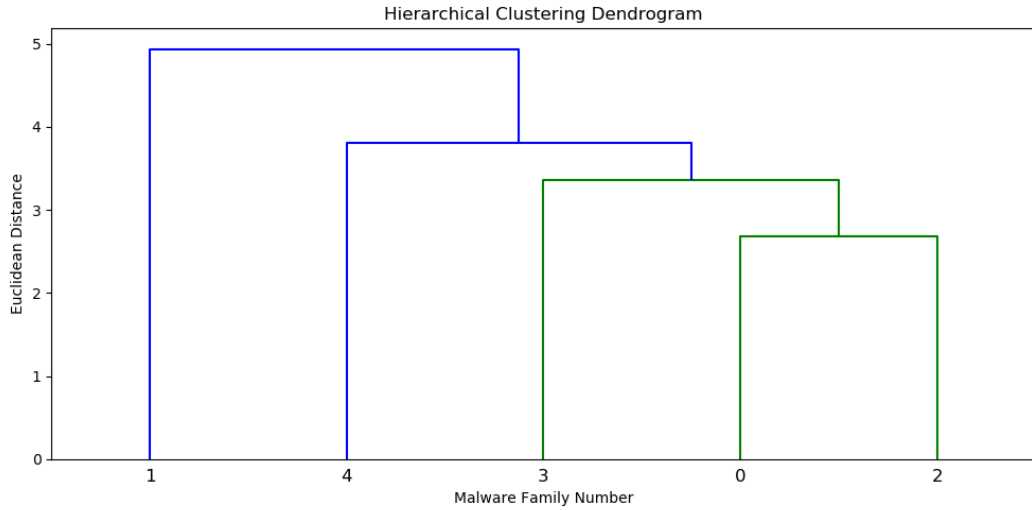


Figure 3.2: This graph is the illustration of the matrix on a dendrogram. Vectors 0 and 2 are the most similar (smallest distance) and vector 1 is the most dissimilar.

This dendrogram shows that the feature vectors 0 and 2 ($\{2, 0, 0, 0, 0, 0, 0, 0, 0\}$ and $\{0, 1, 0, 0, 0, 0, 0, 0, 0\}$), were the most similar followed by vectors 3 and 4. Vector 1 is the most dissimilar.

The process of hierarchical clustering was carried out on all the 12 experiments for all the methods of feature selection (system call representation) and model construction (feature vector representation).

3.1.4 Stage 4 - Validation:

Each dendrogram above was cut at each height to obtain clusters and these clusters were validated using metrics to find the best clustering. The cuts split up the different feature vectors into different clusters. The features in the same cluster are as similar as possible to each other and have a small within-cluster variation/ intra-cluster variance (amount where each feature in a cluster differs from one another). The inter-cluster variance (amount where each feature in a cluster differs from a feature in another cluster) should be large and the samples should be as dissimilar as possible. To illustrate the process of cutting a dendrogram a horizontal line (cut) is made through the dendrogram which splits the features into different clusters. For example with the dendrogram below (Figure 3.3), if the dendrogram was cut at height 2 then the clustering would be $\{\{4, 5\}, \{1, 3\}\}, \{2\}$. This shows that the features 4, 5, 1 and 3 would be in one cluster and 2 would be in another cluster.

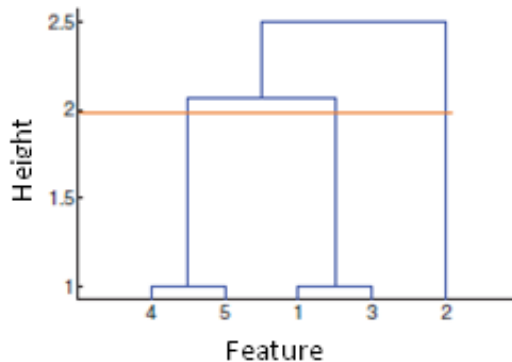


Figure 3.3: This graph illustrates a dendrogram cut at height 2[1].

In the program each dendrogram was cut at every height and this obtained different clusters at each height. For example in this diagram above, the program would cut the dendrogram at height 1 to obtain the clustering $\{\{4, 5\}, \{1, 3\}, \{2\}\}$, height 2 to obtain the clustering $\{\{4, 5\}, \{1, 3\}\}, \{2\}$ and height 2.5 to obtain the clustering $\{\{4, 5\}, \{1, 3\}, \{2\}\}$.

After the dendrogram was cut at each height the clusterings obtained for all the json files by the cut are compared to the labelled dataset from Ramilli[41] and the evaluation metrics Fowlkes Mallows Score (FMS), F1-Score (F1) and Adjusted Rand Index Score (ARI) were used to evaluate how similar the clusters were to the dataset. The cut that was most similar to the labelled data set was the best clustering (i.e. the clusters obtained were most like the labelled dataset and each of the families had the correct/ near correct json files in them). The evaluation metric Silhouette Coefficient (SC) was used to compare the clusterings obtained for all the json files against Euclidean Distance matrices and the cut that produces the highest SC score was the best clustering.

FMS works as follows. If U = true assignments of the dataset and V = the assignments of clusterings by the cut of the dendrogram, then FMS calculates the number of true positives (TP = number of pairs that are in the same clusters in both U and V), true negatives (TN = number of pairs that are in different clusters in both U and V), false positives (FP = number of pairs that are in the same cluster in U but different clusters in V) and false negatives (FN = number of pairs that are in different clusters in U but the same cluster in V) of the true class assignments (the dataset where the json files are split up into families) and compares this against the clustering algorithm assignments of the same samples (the cut of the dendrogram produced for the experiment). FMS is calculated by $\frac{tp}{\sqrt{((tp+fp)*(tp+fn))}}$.

F1 is calculated similarly to FMS. Using U , V and calculating TP, TN, FP and FN (as described above in FMS), F1 is calculated by $2 * \frac{(\text{precision} * \text{recall})}{(\text{precision} + \text{recall})}$ where $\text{precision} = \frac{tp}{(tp+fp)}$ and $\text{recall} = \frac{tp}{(tp+fn)}$.

Rand Index (RI) is calculated in a similar way to FMS and F1. Using U , V and calculating TP, TN, FP and FN (as described above in FMS), RI is calculated by $\frac{tp+tn}{(tp+tn+fp+fn)}$.

These metrics yield a score between 0 and 1. If the true class assignments are equal to the predictive assignments (clustering algorithm assignments) the score should produce a 1.0. The higher the score (nearer to 1) the more similar the true and predictive assignments are and the lower the score (near to 0), the more dissimilar the true and predictive assignments are and the assignments are independent.

Silhouette coefficient (SC) is a metric where if a = mean distance between a sample and all other samples in the same cluster and b = mean distance between a sample and all other samples in the next nearest sample, then the SC of a sample = $\frac{b-a}{\max(a,b)}$. The SC for a set of samples is the mean of the SC for each sample. SC yields a score between -1 and 1. The higher the score (nearer to 1), the better the clustering obtained as the model has well defined clusters. Low scores (nearer to 0) indicate overlapping clusters and negative scores indicate that samples have been assigned to the wrong cluster and another cluster more similar to the sample[30].

For example, a dataset consisted of two malware families with four files in each and each file had feature bit vectors of {0: [1, 0, 0, 0, 0, 0, 0, 0, 0], 1: [1, 0, 1, 1, 0, 1, 1, 1, 0], 2: [0, 1, 0, 0, 0, 0, 0, 0, 0], 3: [1, 0, 1, 0, 0, 0, 0, 1, 0], 4: [0, 0, 0, 0, 0, 0, 0, 1], 5: [0, 1, 0, 0, 1, 0, 0, 0, 1], 6: [0, 0, 0, 0, 0, 0, 0, 1, 0], 7: [0, 1, 0, 1, 1, 1, 1, 0, 0]} respectively. Next the Euclidean Distances were calculated for all the samples (see above on how to calculate it) and a dendrogram (Figure 3.4) was produced.

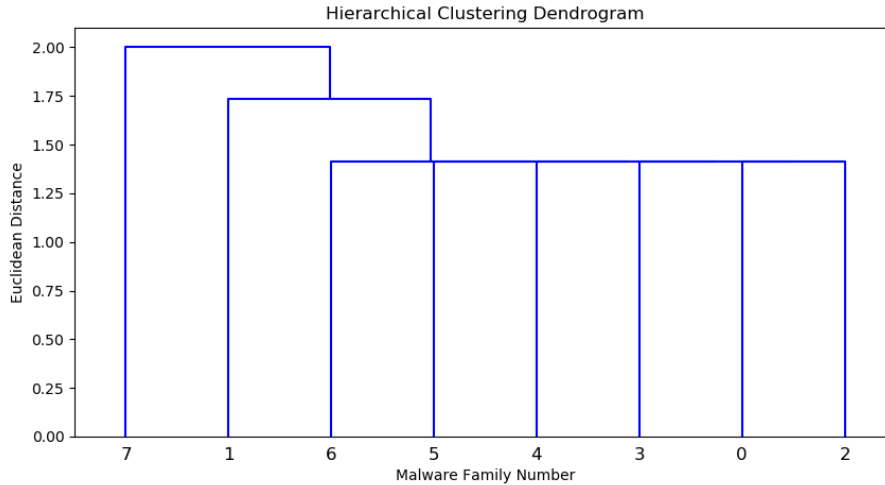


Figure 3.4: This graph illustrates a dendrogram of a dataset consisting of 2 malware families each with four files.

Next the dendrogram obtained was cut at different heights. For each cut, the true assignments (TA) and predictive assignments (PA) were calculated and then the FMS, F1 and RI are calculated from the number of TP, TN, FP, and FN. SC is calculated using PA and the Euclidean Distance matrix. The true class assignments (TA = actual true assignments of the dataset (i.e. json files split up into families)) is [0, 0, 0, 0, 1, 1, 1, 1] as four files is from one family and four files were from another family (the number in the list denotes the cluster number that the file is assigned to). In this example, the dendrogram should be cut 7 times. One such cut splits up the dendrogram into one cluster at height 2.0. Here the predictive assignment (PA = clustering algorithm assignments from the cut) is [0, 0, 0, 0, 0, 0, 0, 0]. This is because when the dendrogram is split into one cluster, every file is in the same cluster (the number in the list denotes the cluster number that the file is assigned to). Calculating the number of TP (number of pairs that are in the same clusters in both TA and PA), TN (number of pairs that are in the different clusters both TA and PA), FP (number of pairs that are in the same cluster in TA but different clusters in PA) and FN (number of pairs that are in the different clusters in TA but the same cluster in PA) at this cut generates 12 true positives, 16 false negatives and 0 false positives and true negatives. The values of these can be calculated by looping through the true assignments and checking the values of pairs in the true assignments against values of pairs in the predictive assignments.

Input : The true assignments (TA) and predictive assignments (PA)

Output: The number of True Positives (tp), True Negatives (tn), False Positives (fp) and False Negatives (fn)

```

for i ← 0 to len(TA) do
  for j ← i + 1 to len(TA): do
    if TA[i] == TA[j] and PA[i] == PA[j]: then
      | tp+ = 1
    end
    if TA[i] != TA[j] and PA[i] != PA[j]: then
      | tn+ = 1
    end
    if TA[i] == TA[j] and PA[i] != PA[j]: then
      | fp+ = 1
    end
    if TA[i] != TA[j] and PA[i] == PA[j]: then
      | fn+ = 1
    end
  end
end

```

end

Algorithm 1: This algorithm calculates the number of true positives, false positives, false negatives and false negatives between the true and predictive assignments.

Finally the scores are calculated. $FMS = \frac{tp}{\sqrt{((tp+fp)*(tp+fn))}} = \frac{12}{\sqrt{((12+0)*(12+16))}} = \frac{12}{\sqrt{((12*28))}} = \frac{12}{\sqrt{336}} = 0.654653670708$, $F1 = 2 * \frac{(precision*recall)}{(precision+recall)}$ where $precision = \frac{tp}{(tp+fp)}$ and $recall = \frac{tp}{(tp+fn)}$, therefore $precision = \frac{12}{(12+0)} = 1$ and $recall = \frac{12}{(12+16)} = \frac{12}{28} = 0.42857142857$ and $F1 = 2 * \frac{(precision*recall)}{(precision+recall)} = 2 * \frac{(1*0.42857142857)}{(1+0.42857142857)} = 2 * \frac{0.42857142857}{1.42857142857} = 2 * 0.3 = 0.6$, $RI = \frac{tp+tn}{(tp+tn+fp+fn)} = \frac{12+0}{(12+0+0+16)} = \frac{12}{(12+16)} = \frac{12}{28} = 0.4285714285714$ and $SC = -1$ as the SC only works with a cut greater than 1 (ie the samples should be split into more than one cluster).

This process continues for all cuts for each dendrogram and the best clustering from the 12 experiments at the different cuts is the cut that has the FMS, F1, RI and SC nearest to 1. RI needs to be adjusted as to be corrected for chance. Python has a function (Adjusted Rand Index) that already adjusts the manual RI calculation above and so this project used ARI instead of RI.

FMS, F1 and RI were used because there was access to a labelled dataset from Ramilli[41] and these metrics used when the true assignments are known. All ensure that any random predictive assignment has a score close to 0 which means that the assignments are different and independent from each other and no assumptions are made about the clustering method used or the cluster structure produced. In reality, testing labels will not be available so the true class assignments will not be known. Therefore, other clustering validation techniques must be used such as SC. SC is used when the true assignments are not known. The score is higher when clusters are dense and well separated[30].

If testing labels (ground truth) are not known, once the best clustering is obtained a programmer would randomly select a number of objects for each cluster and count whether it is correct or not. This result should be enriched by a confidence interval (measure of uncertainty given the whole population) because only a certain number of objects would have been selected. In this project I had access to labelled dataset so using FMS, F1 and ARI I could tell how close and correct the results are to the dataset.

4

Results for Clustering Malware

This chapter visualises, explains and analyses the results from my approach for clustering malware.

In particular:

Section 4.1 visualises the results of clustering Ransomware samples into families including looking at the results of the system call dimensions, euclidean distances and validation metrics.

Section 4.2 visualises the results of clustering Backdoor samples into families including looking at the results of the system call dimensions, euclidean distances and validation metrics.

Section 4.3 visualises the results of clustering Trojan samples into families including looking at the results of the system call dimensions, euclidean distances and validation metrics.

4.1 Results of Clustering Ransomware

In total, 12 experiments were run for each system call representation and feature vector representation. The results for each experiment are detailed below:

There are 15 different malware families from the Ramilli[41] dataset with 10727 evidences (includes duplicates) in total. The table below (Table 4.1) displays the Ransomware families and the number of evidences for each family:

Name of Family	Number of Evidences
Ransomware.Cryptowall-1201142	465
Ransomware.Jigsaw-1201143	898
Ransomware.Locky-1201144	863
Ransomware.Matsnu-1201145	525
Ransomware.Petya-1201146	1019
Ransomware.Petya-1201147	1909
Ransomware.Radamant-1201148	786
Ransomware.Satana-1201149	274
Ransomware.Satana-1201150	466
Ransomware.TeslaCrypt-1201151	748
Ransomware.TeslaCrypt-1201152	321
Ransomware.TeslaCrypt-1201153	368
Ransomware.Vipasana-1201154	719
Ransomware.Vipasana-1201155	722
Ransomware.Vipasana-1201156	644

Table 4.1: This table displays all the Ransomware families and the number of evidences for each family.

There were 10727 different feature vectors for each experiment. The number of system call dimensions in the vectors changed depending on the system call representation and type of feature vector used. Please see Appendix B for text files which show the results each experiment across the three different types of malware.

4.1.1 System Call Dimensions

Trends can be seen from the results. Within a specific n-gram, the more semantics being represented (e.g. full representation compared to category), the higher the number of dimensions in a vector and the larger the feature space. For example, the table below (Table 4.2) shows this for the Uni-grams.

Vector	System Call Dimensions
Full Representation	45
Category	8

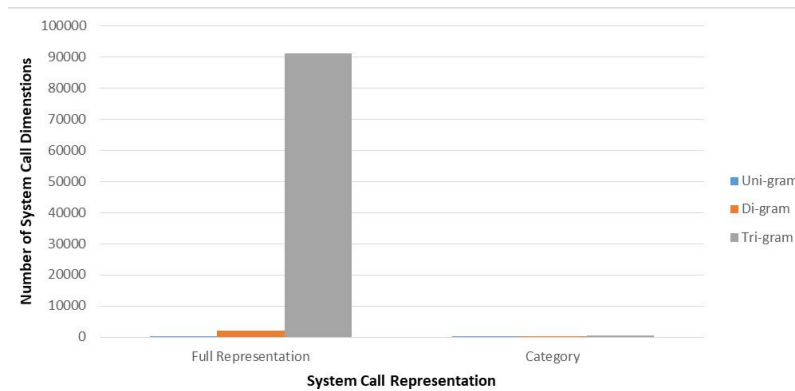
Table 4.2: This table displays that within a specific n-gram, the more semantics being represented the higher the number of dimensions in a vector and the larger the feature space.

Secondly, for each n-gram the higher the number of system calls per dimension (e.g. 1-gram = 1 system call per dimension), the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector. For example, the table below (Table 4.3) shows this for the Uni-gram Category Bit Vectors. This is because the feature space increases and the number of dimensions of the feature vectors are larger.

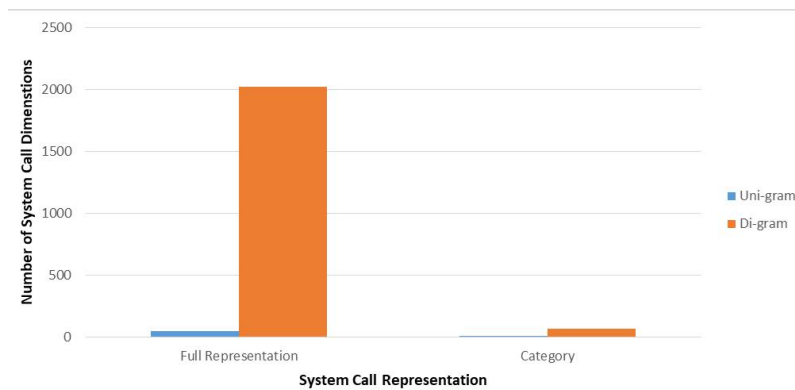
Vector	System Call Dimensions
Uni-gram	8
Di-gram	64
Tri-gram	512

Table 4.3: This table displays the higher the number of system calls per dimension, the higher the number of dimensions in a vector.

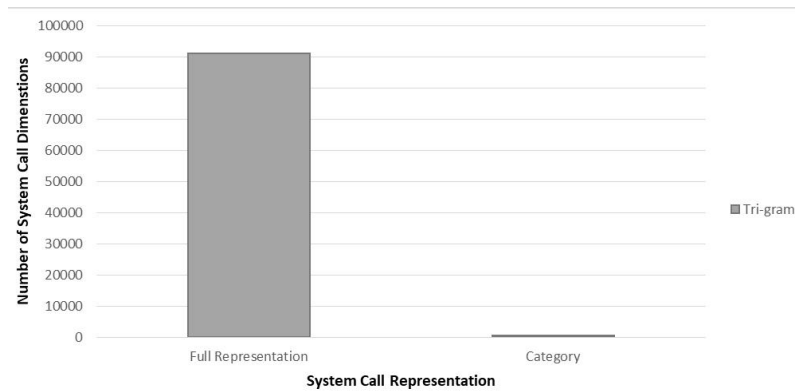
The graphs below (Figure 4.1) illustrates these trends. You will see three graphs. The first shows the Uni-grams, Di-grams and Tri-grams all on one graph, the second shows the Uni-grams and Di-grams and the third one shows the Tri-grams on their own. This was done because the number of system call dimensions were much larger for the Tri-grams compared to the Uni-grams and Di-grams so one cannot see what the graph is trying to illustrate. These graphs illustrate that the more semantics that were being represented within an n-gram, the higher the number of system calls per dimension and the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector.



(a) The Uni-grams, Di-grams and Tri-grams.



(b) The Uni-grams and Di-grams.



(c) The Tri-grams.

Figure 4.1: These graphs illustrate that the more semantics that were being represented (higher number of system calls per dimension) within an n-gram, the higher the number of dimensions in a vector.

4.1.2 Euclidean Distance

The tables below (Tables 4.4, 4.5, 4.6 and 4.7) display the Euclidean Distance matrices for each system call representation and type of vector. The columns are the distances being represented, the rows are the type of n-gram and the cells represent the number of evidences that had the distance. A small distance between two evidences means that the two evidences are similar to one another (dissimilar for large distances). A distance of 0 between 2 evidences means that the two evidence feature vectors are the same:

1. Full Representation

(a) Bit Vector

Vector	0	1	1-2	2-3	3-4	Largest Distance
Uni-gram	10646	58	21	1	0	2
Di-gram	10611	66	36	11	1	3.162
Tri-gram	10588	77	38	19	4	3.606

Table 4.4: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Full Representation and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0-5	5-10	10-20	20-50	50-100	100-150	150-300	Largest Distance
Uni-gram	10524	117	38	24	16	6	1	0	103.577
Di-gram	10531	68	29	21	39	26	10	2	174.894
Tri-gram	10522	51	20	25	33	44	21	10	295.971

Table 4.5: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Full Representation and Vector Representaion of a Frequency Vector.

2. Category

(a) Bit Vector

Vector	0	1	1-2	Largest Distance
Uni-gram	9846	17	2	1.414
Di-gram	9839	23	3	1.732
Tri-gram	9846	17	2	1.732

Table 4.6: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Category and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0- 5	5-10	10-60	60-150	Largest Distance
Uni-gram	9802	40	19	4	0	52.593
Di-gram	9809	25	9	17	4	107.300
Tri-gram	9822	19	4	12	7	143.378

Table 4.7: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Category and Vector Representaion of a Frequency Vector.

Three trends can be seen. Within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances (especially at 0). In comparison, frequency vectors that have the same n-gram and system call representation have larger range of distances and the largest distances were bigger. For example the table below (Table 4.8) shows this with for Uni-gram Full Representation Vectors (this also applies to Di-grams and Tri-grams). This is because the distance between the feature vectors are larger, so there are more distances and less vectors that are similar.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Bit Vector	10646	0, 1, between 1 and 2 and between 2 and 3	2
Frequency Vector	10524	0, between 0 and 5, between 5 and 10, between 10 and 20, between 20 and 50, between 50 and 100, between 100 and 150 and between 150 and 300	104

Table 4.8: This table displays that within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances compared to frequency vectors.

The same trend was seen within an n-gram (Uni-gram, Di-gram, Tri-gram). Within an n-gram with the same vector representation (bit or frequency), the more semantics that is represented in the feature vector (full representation), the higher the number of evidences at lower distances (especially at 0). In comparison to the same n-gram with the same feature vector representation but less semantics are represented. In this case, less distances are represented and the largest distances were lower. For example, the table below (Table 4.9) shows this for Uni-gram Bit Vectors (this also applies to Di-grams and Tri-grams). This is because the feature space decreases and the number of dimensions of the feature vectors are smaller and the distance between the feature vectors are smaller and so there are less distances and more vectors that are similar.

System Call Representation	Number of evidences at 0	Distances Represented	Largest Distance
Full Representation	10646	0, 1, between 1 and 2 and between 2 and 3	2
Category	9846	0, 1 and between 1 and 2	1.4

Table 4.9: This table displays that less distances were represented and the largest distances were lower.

Another trend could be seen with respect to the type of n-gram being represented. With the same system call representation and type of vector representation (bit or frequency), the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower the number of evidences at lower distances (especially at 0). When there were higher numbers of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram) with the same system call representation and type of vector, more distances were represented and the largest distances were bigger. For example, the table below (Table 4.10) shows this for Full Representation Bit Vectors (this also applies to the category representation). This is because the feature space increases and the number of dimensions of the feature vectors are larger and the distance between the feature vectors are larger and so there are many more distances and less vectors that are similar.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Uni-gram	10646	0, 1, between 1 and 2 and between 2 and 3	2
Di-gram	10611	0, 1, between 1 and 2, between 2 and 3 and between 3 and 4	3.2
Tri-gram	10588	0, 1, between 1 and 2, between 2 and 3 and between 3 and 4	3.6

Table 4.10: This table displays that with the same system call representation and type of vector representation, the lower the number of system calls per dimension, the lower the number of evidences at lower distances.

This graph 4.2 below shows that within an n-gram, eventhough the system call representations were the same (Full Representation), the number of evidences that had a distance 0 was higher with the bit vectors than the frequency vectors.

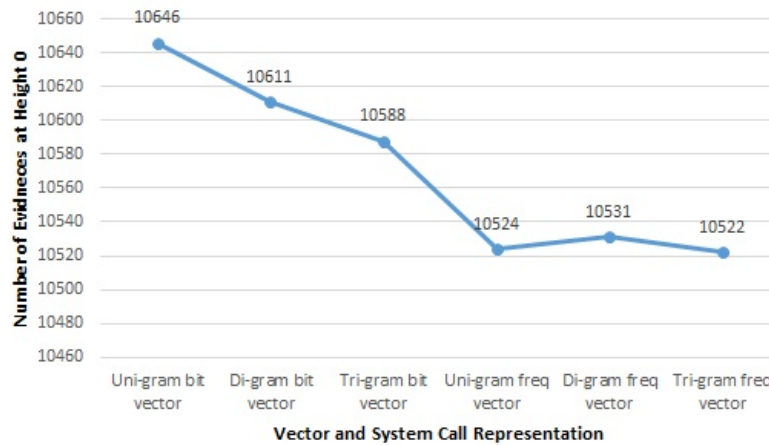
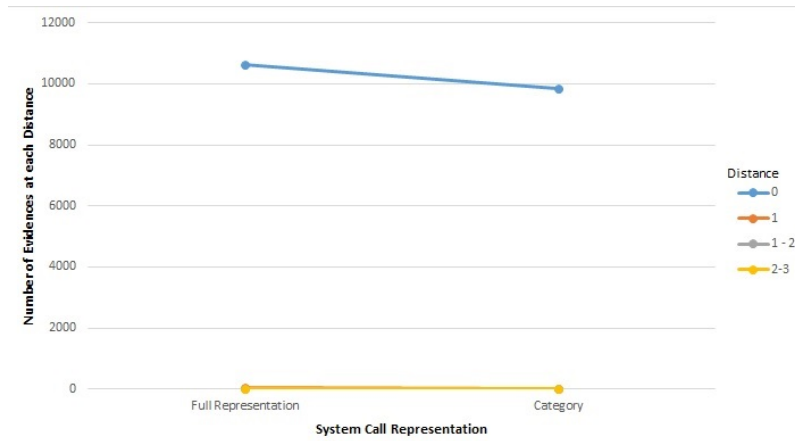
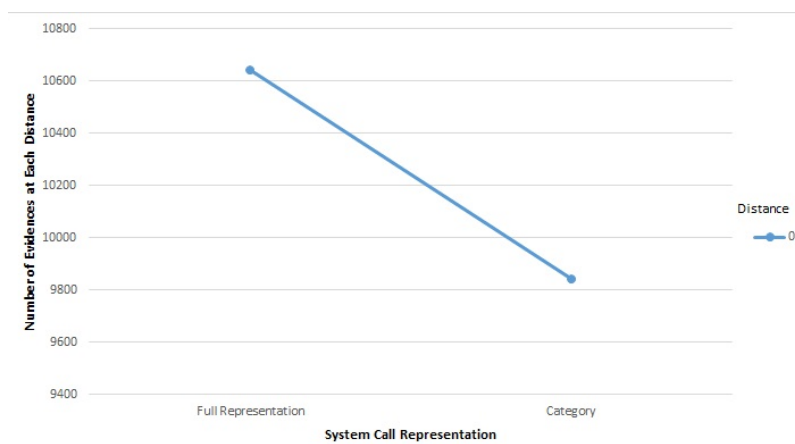


Figure 4.2: This graph illustrates that the number of evidences that had a distance 0 was higher with the bit vectors than the frequency vectors.

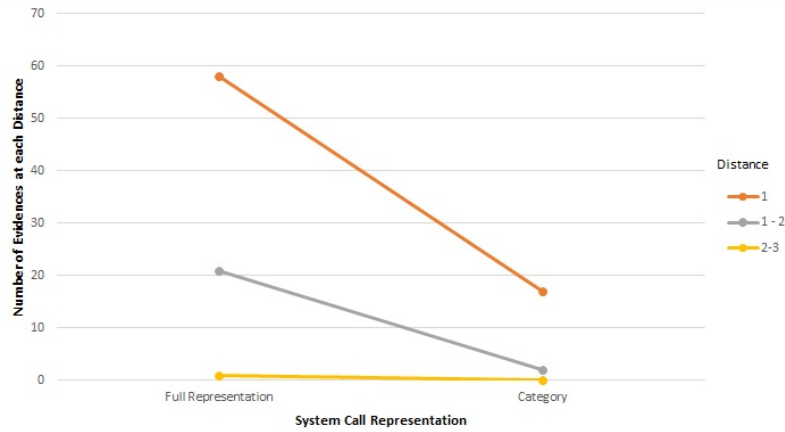
The graphs below (Figure 4.3) show that the fewer the semantics represented in the feature vector by the system call representations (fewer semantics is the category representation), the lower the number of evidences at lower distances compared to the more semantics represented. You will see three graphs. The first shows all the distances all on one graph, the second shows just the distance 0 and the third graph shows all the distances except 0. This was done because the number of evidences was much larger for the distance 0 compared to the other distances so one cannot see what the graph is trying to illustrate.



(a) Distances 0, 1, 1-2 and 2-3.



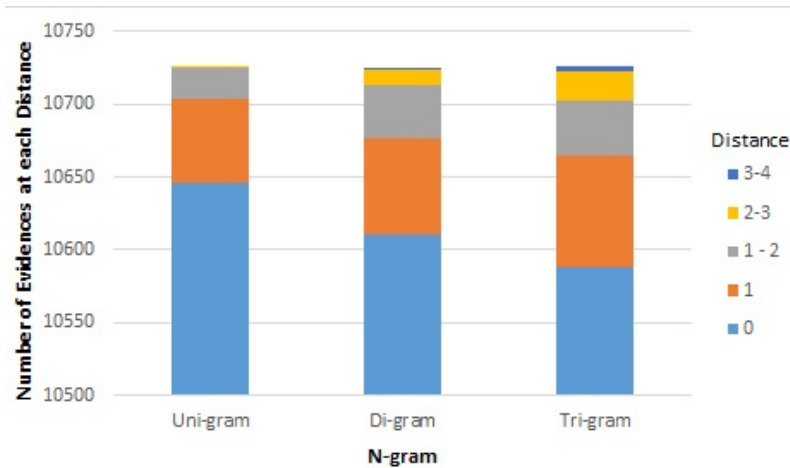
(b) Distance 0.



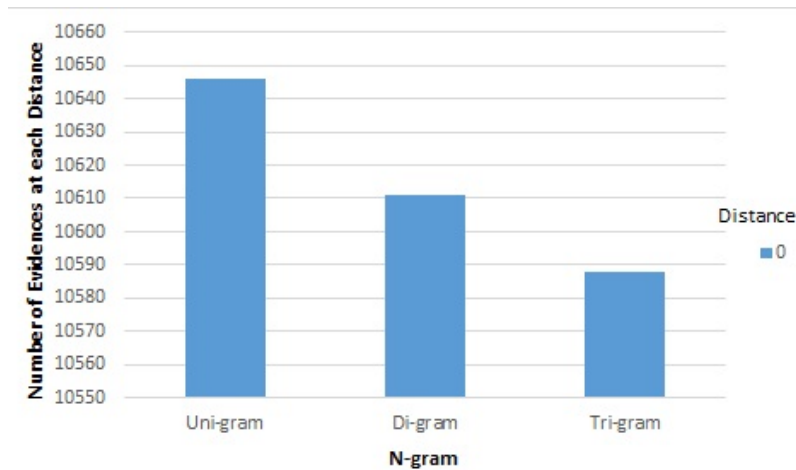
(c) Distances 1, 1-2 and 2-3.

Figure 4.3: These graphs illustrate that the fewer the semantics represented in the feature vector by the system call representations (fewer semantics is the category representation), the lower the number of evidences at lower distances compared to the more semantics represented.

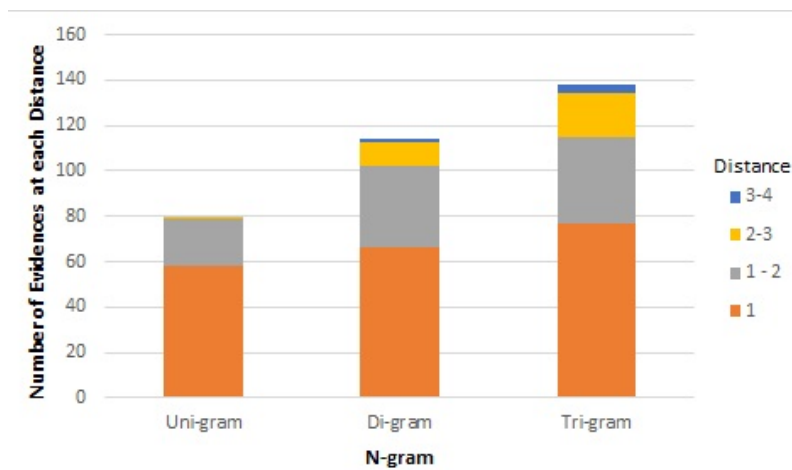
The graphs below (Figure 4.4) show that the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the higher number of evidences at lower distances (especially at 0). You will see three graphs. The first shows all the distances all on one graph, the second shows just the distance 0 and the third graph shows all the distances except 0. This was done because the number of evidences was much larger for the distance 0 compared to the other distances so one cannot see what the graph is trying to illustrate.



(a) Distances 0, 1, 1-2, 2-3 and 3-4.



(b) Distance 0.



(c) Distances 1, 1-2, 2-3 and 3-4.

Figure 4.4: These graphs illustrate that the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the higher the number of evidences at lower distances (especially at 0).

These trends can be seen because when more features are being represented, the feature space increases and the number of dimensions of the feature vectors are larger and the distance between the feature vectors are larger and so there are many more distances and less vectors that are similar.

4.1.3 Validation Metrics

The table below (Table 4.11) displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at. (Red indicates the highest FMS, F1, ARI or SC score for each n-gram):

Vector and Syscall Representation	Best FMS	Best F1	Best ARI	Best SC
Uni-gram, Full Representation, Bit Vector	0.293 at height 4.173	0.074 at height 0.0	0.002 at height 1.0	0.997 at height 1.0
Uni-gram, Full Representation, Frequency Vector	0.293 at height 1.818	0.129 at height 1.0	0.003 at height 1.0	0.992 at height 0.0

Uni-gram, Category, Bit Vector	0.275 at height 0.0	0.076 at height 0.0	0.002 at height 1.0	0.999 at height 1.0
Uni-gram, Category, Frequency Vector	0.302 at height 0.0	0.076 at height 22.390	0.002 at height 1.0	0.998 at height 1.0
Di-gram, Full Representation, Bit Vector	0.293 at height 0.0	0.0675 at height 18.667	0.001 at height 1.0	0.995 at height 1.0
Di-gram, Full Representation, Frequency Vector	0.293 at height 5.398	0.148 at height 10.329	0.001 at height 1.0	0.991 at height 1.0
Di-gram, Category, Bit Vector	0.301 at height 4.173	0.068 at height 1.0	0.001 at height 1.0	0.999 at height 1.0
Di-gram, Category, Frequency Vector	0.301 at height 14.286	0.070 at height 1.0	0.001 at height 0.0	0.998 at height 1.0
Tri-gram, Full Representation, Bit Vector	0.293 at height 3.606	0.049 at height 2.449	0.0003 at height 1.0	0.993 at height 1.0
Tri-gram, Full Representation, Frequency Vector	0.293 at height 295.971	0.049 at height 146.479	0.001 at height 13.374	0.988 at height 0.528
Tri-gram, Category, Bit Vector	0.301 at height 4.173	0.102 at height 0.0	0.0 at height 0.0	0.999 at height 1.0
Tri-gram, Category, Frequency Vector	0.301 at height 0.0	0.103 at height 0.0	0.000 at height 22.100	0.998 at height 1.0

Table 4.11: This table displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at.

Looking at this table (Table 4.11), the table below (Table 4.12) shows the experiments with the best clustering according to FMS, F1, ARI and SC.

Metric	N-gram, System Call Representation and Vector Representation	Score and Height
FMS	Uni-gram Category Frequency Vector	0.302 at height 0.0
F1	Di-gram Full Representation Frequency Vector	0.148 at height 10.329
ARI	Uni-gram Full Representation Frequency Vector	0.003 at height 1.0
SC	Uni-gram Category Bit Vector	0.999 at height 1.0

Table 4.12: This table displays that the best clustering according to FMS, F1, ARI and SC were all different.

In all evaluation metrics, the best clustering method that was chosen was different (see above). The method of feature selection (system call representation) was the same in FMS and SC (category) but different in F1 and ARI (full representation). The method of model construction (feature vector representation) was the same in FMS and F1 and ARI (frequency vector) but different in SC and the N-gram was the same in FMS, ARI and SC but different in F1. These experiments are completely different to each other in terms of the number of dimensions in a vector (45 for Uni-gram, 2025 for Di-gram and 91125 for Tri-gram for full representation and 8 for Uni-gram, 64 for Di-gram and 512 for Tri-gram for category) and the features abstracted from the behavioural profiles. The type of n-gram was different and the scores produced were different. No n-gram had the highest score at the same experiment. The highest scores for each n-gram were frequency vectors (except for SC) and a mixture of system call representation with a few exceptions in the vectors in SC.

In the FMS, F1 and ARI metrics, the scores produced were not high as they were closer to 0. Even though each metric calculated the best method of feature selection and model construction for each dendrogram cut which produced the best clustering, these experiments were not similar to the dataset[41] (true assignments) so these results should be taken with caution. As the scores produced were not near to 1 and the same experiment did not produce the best clustering in all FMS, F1 and ARI, if a clustering system used either one of the best clustering methods and was presented with another dataset, there is no guarantee that the system will group samples from

the same Ransomware families in the same clusters so the user will not be able to distinguish between the different Ransomware families. However, in SC the best clustering was very near 1.0 (the highest score possible) and most scores produced were close to 1. This means that this model has well defined clusters.

The table below (Table 4.13) shows the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up. For example, the Uni-gram Category Frequency Vector is the best FMS score so will yield a score of 1 whereas the Uni-gram Category Bit Vector is the worst FMS score so will yield a score of 12. This table is ordered from lowest (best) to highest (worst) total score.

Vector	FMS	F1	ARI	SC	Total
Uni-gram Category Frequency Vector	1	6	2	5	14
Uni-gram Full Representation Frequency Vector	6	2	1	10	19
Di-gram Category Bit Vector	2	9	7	3	21
Tri-gram Category Frequency Vector	4	3	11	4	22
Uni-gram Category Bit Vector	12	5	4	1	22
Tri-gram Category Bit Vector	5	4	12	2	23
Di-gram Full Representation Frequency Vector	7	1	5	11	24
Di-gram Category Frequency Vector	3	8	8	6	25
Uni-gram Full Representation Bit Vector	8	7	3	7	25
Di-gram Full Representation Bit Vector	9	10	6	8	33
Tri-gram Full Representation Frequency Vector	10	11	9	12	42
Tri-gram Full Representation Bit Vector	11	12	10	9	42

Table 4.13: This table displays the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up.

This table (Table 4.13) shows that overall the best clustering method is Uni-gram with the system call representation of Category and vector representation of Frequency Vector.

4.2 Results of Clustering Backdoor

There are 5 different malware families from the Ramilli[41] dataset with 5003 evidences (includes duplicates) in total. The table below (Table 4.14) displays the Backdoor families and the number of evidences for each family:

Name of Family	Number of Evidences
Backdoor.MSIL.Tyupkin-1201109	1103
Backdoor.MSIL.Tyupkin-1201110	1128
Backdoor.MSIL.Tyupkin-1201111	1106
Backdoor.MSIL.Tyupkin-1201112	1103
Backdoor.MSIL.Tyupkin-1201113	563

Table 4.14: This table displays all the Backdoor families and the number of evidences for each family.

There were 5003 different feature vectors for each experiment. The number of system call dimensions in the vectors changed depending on the system call representation and type of feature vector used. Please see Appendix B for text files which show the results each experiment across the three different types of malware.

4.2.1 System Call Dimensions

Trends can be seen from the results. Within a specific n-gram, the more semantics being represented (e.g. full representation compared to category), the higher the number of dimensions in a vector and the larger the feature space. For example, the table below (Table 4.15) shows this for the Uni-grams.

Vector	System Call Dimensions
Full Representation	7
Category	3

Table 4.15: This table displays that within a specific n-gram, the more semantics being represented the higher the number of dimensions in a vector and the larger the feature space.

Secondly, for each n-gram the higher the number of system calls per dimension (e.g. 1-gram = 1 system call per dimension), the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector. For example, the table below (Table 4.16) shows this for the Uni-gram Category Bit Vectors. This is because the feature space increases and the number of dimensions of the feature vectors are larger.

Vector	System Call Dimensions
Uni-gram	3
Di-gram	9
Tri-gram	27

Table 4.16: This table displays the higher the number of system calls per dimension, the higher the number of dimensions in a vector.

The graph below (Figure 4.5) illustrate these trends. It shows that the more semantics that were being represented within an n-gram, the higher the number of system calls per dimension and the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector.

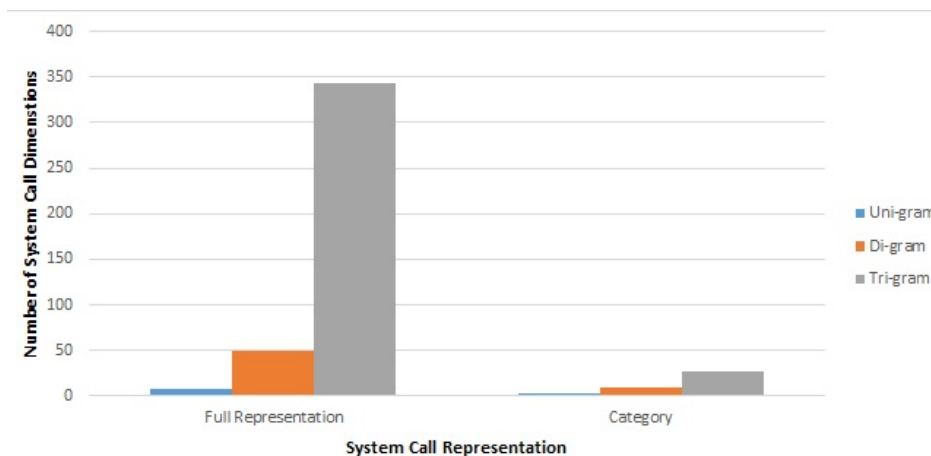


Figure 4.5: This graph illustrates that the more semantics that were being represented (higher number of system calls per dimension) within an n-gram, the higher the number of dimensions in a vector.

4.2.2 Euclidean Distance

The tables below (Tables 4.17, 4.18, 4.19 and 4.20) display the Euclidean Distance matrices for each system call representation and type of vector. The columns are the distances being represented, the rows are the type of n-gram and the cells represent the number of evidences that had the distance. A small distance between two evidences means that the two evidences are similar to one another (dissimilar for large distances). A distance of 0 between 2 evidences means that the two evidence feature vectors are the same:

1. Full Representation

(a) Bit Vector

Vector	0	1	1-2	Largest Distance
Uni-gram	4995	4	3	1.414
Di-gram	4996	6	0	1.0
Tri-gram	4998	4	0	1.0

Table 4.17: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Full Representation and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0-5	5-10	10-20	20-50	Largest Distance
Uni-gram	4986	10	2	4	0	14.807
Di-gram	4991	6	0	2	3	31.648
Tri-gram	4996	2	0	0	3	34.320

Table 4.18: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Full Representation and Vector Representaion of a Frequency Vector.

2. Category

(a) Bit Vector

Vector	0	1	1-2	Largest Distance
Uni-gram	4870	2	1	1.414
Di-gram	4870	3	0	1.0
Tri-gram	4871	2	0	1.0

Table 4.19: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Category and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0 - 5	5 - 10	10 - 60	60 - 150	Largest Distance
Uni-gram	4865	4	2	2	0	16.514
Di-gram	4867	3	0	3	0	37.335
Tri-gram	4870	0	0	2	1	69.821

Table 4.20: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Category and Vector Representation of a Frequency Vector.

Five trends can be seen. Within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances (especially at 0). In comparison, frequency vectors that have the same n-gram and system call representation have larger range of distances and the largest distances were bigger. For example the table below (Table 4.21) shows this with for Uni-gram Full Representation Vectors (this also applies to Di-grams and Tri-grams). This is because the distance between the feature vectors are larger, so there are more distances and less vectors that are similar.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Bit Vector	4995	0, 1 and between 1 and 2	1.4
Frequency Vector	4986	0, between 0 and 5, between 5 and 10 and between 10 and 20	14.8

Table 4.21: This table displays that within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances compared to frequency vectors.

The same trend was seen was seen within an n-gram (Uni-gram, Di-gram, Tri-gram). Within an n-gram with the same vector representation (frequency), the more semantics that is represented in the feature vector (full representation), the higher the number of evidences at lower distances (especially at 0). In comparison to the same n-gram with the same feature vector representation but less semantics are represented. In this case, more distances are represented and the largest distances were higher. For example, the table below (Table 4.22) shows this for Uni-gram Frequency Vectors (this also applies to Di-grams and Tri-grams). This is because the feature space decreases and the number of dimensions of the feature vectors are smaller and the distance between the feature vectors are smaller and so there are less distances and more vectors that are similar.

System Call Representation	Number of evidences at 0	Distances Represented	Largest Distance
Full Representation	4986	0, between 0 and 5, between 5 and 10 and between 10 and 20	14.8
Category	4865	0, between 0 and 5, between 5 and 10 and between 10 and 60	16.5

Table 4.22: This table displays that less distances were represented and the largest distances were lower.

Another aspect to note from these results is that with different system call representations but the same vector representation of a bit vector and n-gram, the same distances are represented and the largest distances were the same. For example, the table below (Table 4.23) shows this for the Uni-gram bit vectors (this also applies to Di-grams and Tri-grams).

System Call Representation	Number of evidences at 0	Distances Represented	Largest Distance
Full Representation	4995	0, 1 and between 1 and 2	1.4
Category	4870	0, 1 and between 1 and 2	1.4

Table 4.23: This table displays that when there were different system call representations but the same vector representation of a bit vector and n-gram, the same distances were represented and the largest distances were the same.

Another trend could be seen with respect to the type of n-gram being represented. With the same system call representation and type of vector representation (frequency), the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower the number of evidences at lower distances (especially at 0). When there were higher numbers of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram) with the same system call representation and type of vector, more distances were represented and the largest distances were bigger. For example, the table below (Table 4.24) shows this for Full Representation Frequency Vectors (this also applies to the category representation). This is because the distance between the feature vectors are larger, so there are more distances and less vectors that are similar.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Uni-gram	4986	0, between 0 and 5, between 5 and 10 and between 10 and 20	14.8
Di-gram	4991	0, between 0 and 5, between 5 and 10, between 10 and 20 and between 20 and 50	31.6
Tri-gram	4996	0, between 0 and 5, between 5 and 10, between 10 and 20 and between 20 and 50	34.3

Table 4.24: This table displays that with the same system call representation and type of vector representation, the lower the number of system calls per dimension, the lower the number of evidences at lower distances.

Another aspect to note from these results is that with the same vector representation of a bit vector and system call representation, the higher the number of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram), less distances were represented and the largest distances were lower. For example, the table below (Table 4.25) shows this for the Full Representation bit vectors (this also applied to the category representation).

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Uni-gram	4995	0, 1 and between 1 and 2	1.4
Di-gram	4996	0 and 1	1.0

Table 4.25: This table displays that the higher the number of system calls per dimension in an n-gram with the same system call representation and type of vector, less distances were represented and the largest distances were lower.

This graph below (Figure 4.6) shows that within an n-gram, even though the system call representations were the same (Full Representation), the number of evidences that had a distance 0 was higher with the bit vectors than the frequency vectors.

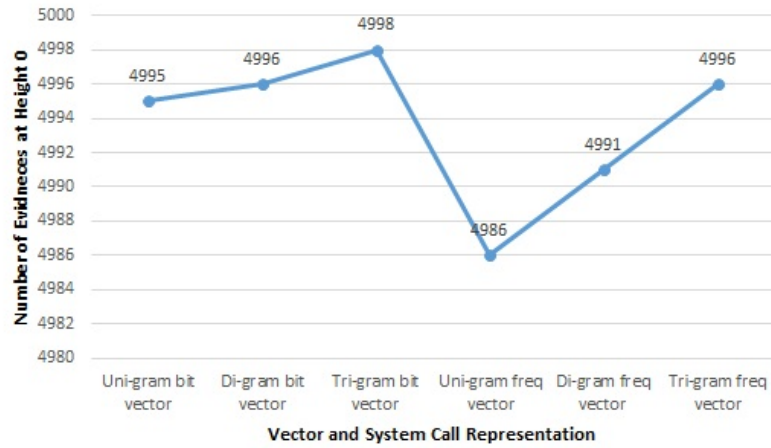
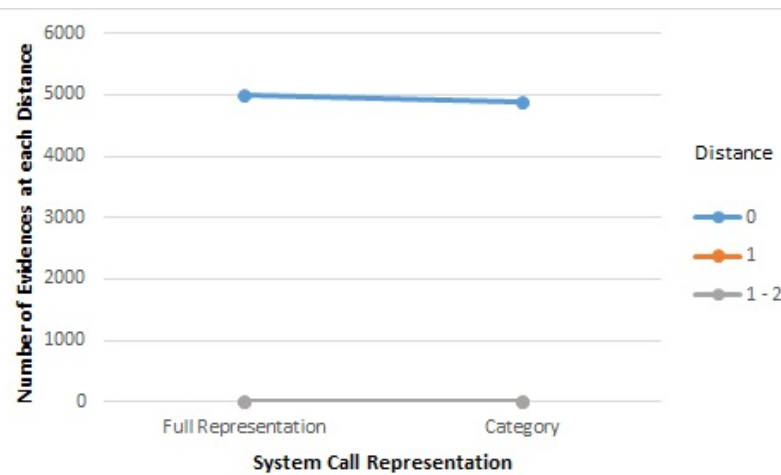
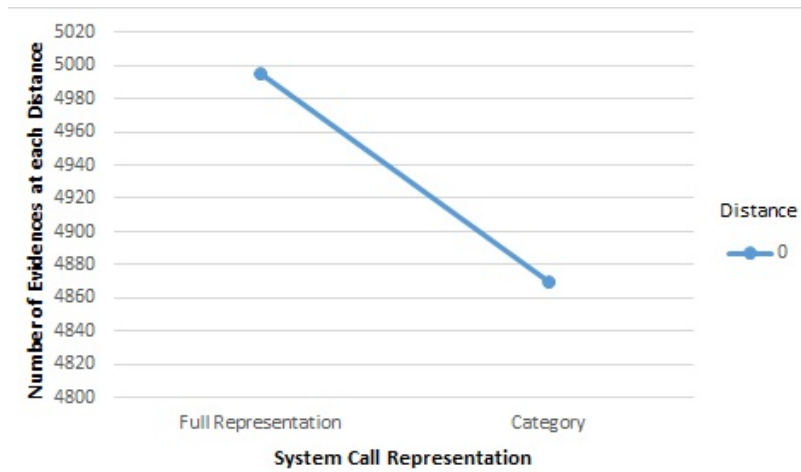


Figure 4.6: This graph illustrates that the number of evidences that had a distance 0 was higher with the bit vectors than the frequency vectors.

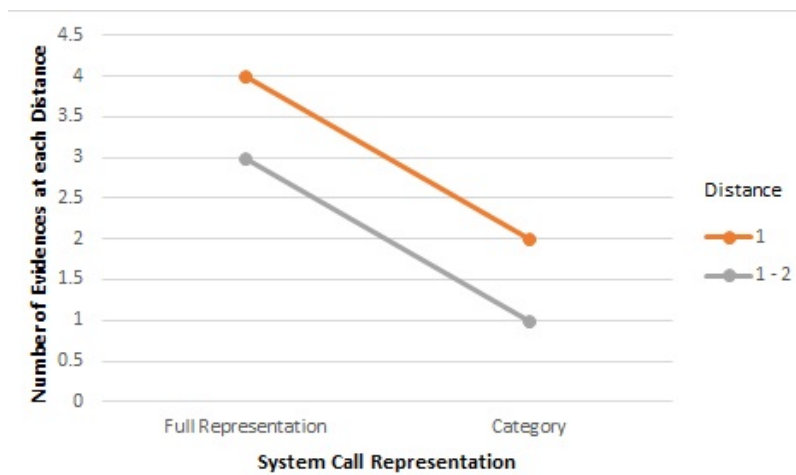
The graphs below (Figure 4.7) show that the fewer the semantics represented in the feature vector by the system call representations (fewer semantics is the category representation), the lower the number of evidences at lower distances compared to the more semantics represented. You will see three graphs. The first shows all the distances all on one graph, the second shows just the distance 0 and the third graph shows all the distances except 0. This was done because the number of evidences was much larger for the distance 0 compared to the other distances so one cannot see what the graph is trying to illustrate.



(a) Distances 0, 1 and 1-2.



(b) Distance 0.



(c) Distances 1 and 1-2.

Figure 4.7: These graphs illustrate that the fewer the semantics represented in the feature vector by the system call representations (fewer semantics is the category representation), the lower the number of evidences at lower distances compared to the more semantics represented.

The graph below (Figure 4.8) shows that the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower number of evidences at lower distances (especially at 0).

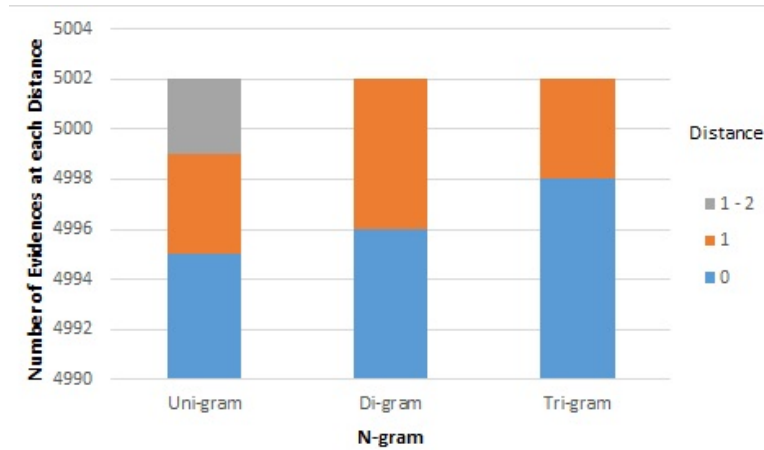


Figure 4.8: This graph illustrates that the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower number of evidences at lower distances (especially at 0).

These trends can be seen because when more features are being represented, the feature space increases and the number of dimensions of the feature vectors are larger and the distance between the feature vectors are larger and so there are many more distances and less vectors that are similar.

4.2.3 Validation Metrics

The table below (Table 4.26) displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at. (Red indicates the highest FMS, F1, ARI or SC score for each n-gram):

Vector and Syscall Representation	Best FMS	Best F1	Best ARI	Best SC
Uni-gram, Full Representation, Bit Vector	0.443 at height 0.0	0.221 at height 0.0	0.003 at height 1.0	1.0 at height 1.0
Uni-gram, Full Representation, Frequency Vector	0.455 at height 10.329	0.223 at height 1.0	0.003 at height 1.0	0.999 at height 1.0
Uni-gram, Category, Bit Vector	0.453 at height 0.0	0.216 at height 0.0	0.0002 at height 1.0	1.0 at height 1.0
Uni-gram, Category, Frequency Vector	0.457 at height 37.335	0.217 at height 4.173	0.002 at height 4.173	0.999 at height 1.0
Di-gram, Full Representation, Bit Vector	0.402 at height 1.0	0.213 at height 1.0	0.003 at height 1.0	1.0 at height 1.0
Di-gram, Full Representation, Frequency Vector	0.457 at height 37.335	0.221 at height 4.173	0.003 at height 1.0	0.999 at height 1.0
Di-gram, Category, Bit Vector	0.433 at height 1.0	0.218 at height 1.0	0.001 at height 1.0	1.0 at height 1.0
Di-gram, Category, Frequency Vector	0.457 at height 18.667	0.217 at height 1.0	0.001 at height 1.0	0.999 at height 1.0
Tri-gram, Full Representation, Bit Vector	0.439 at height 1.0	0.218 at height 1.0	0.002 at height 1.0	1.0 at height 1.0
Tri-gram, Full Representation, Frequency Vector	0.457 at height 18.667	0.220 at height 18.667	0.002 at height 1.0	0.999 at height 1.0
Tri-gram, Category, Bit Vector	0.456 at height 1.0	0.216 at height 1.0	8.1e-05 at height 1.0	0.999 at height 1.0
Tri-gram, Category, Frequency Vector	0.457 at height 4.173	0.216 at height 1.0	8.1-05 at height 1.0	0.999 at height 1.0

Table 4.26: This table displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at.

Looking at this table (Table 4.26), the table below (Table 4.27) shows the experiments with the best clustering according to FMS, F1, ARI and SC.

Metric	N-gram, System Call Representation and Vector Representation	Score and Height
FMS	Tri-gram Full Representation Frequency Vector	0.457 at height 18.667
F1	Uni-gram Full Representation Frequency Vector	0.223 at height 1.0
ARI	Di-gram Full Representation Bit Vector	0.003 at height 1.0
SC	Uni-gram Full Representation Bit Vector, Uni-gram Category Bit Vector, Di-gram Full Representation Bit Vector, Di-gram Category Bit Vector and Tri-gram Full Representation Bit Vector	1.0 at height 1.0

Table 4.27: This table displays that the best clustering according to FMS, F1, ARI and SC were all different.

In all evaluation metrics, the best clustering method that was chosen was different (see above). The method of feature selection (system call representation) was the same in FMS, F1, ARI and some of SC (full representation). The method of model construction (feature vector representation) was the same in FMS and F1 and some of SC (frequency vector), it was the type of n-gram that was different and the scores produced were different. These experiments are completely different to each other in terms of the number of dimensions in a vector (7 for Uni-gram, 49 for Di-gram and 343 for Tri-gram) and the features abstracted from the behavioural profiles. The type of n-gram was different and the scores produced were different. No n-gram had the highest score at the same experiment. The highest scores for each n-gram were frequency vectors (except for ARI) and full representation with a few exceptions in the vectors in SC.

In the FMS, F1 and ARI metrics, the scores produced were not high as they were closer to 0. Even though each metric calculated the best method of feature selection and model construction for each dendrogram cut which produced the best clustering, these experiments were not similar to the dataset[41] (true assignments) so these results should be taken with caution. As the scores produced were not near to 1 and the same experiment did not produce the best clustering in all FMS, F1 and ARI, if a clustering system used either one of the best clustering methods and was presented with another dataset, there is no guarantee that the system will group samples from the same Backdoor families in the same clusters so the user will not be able to distinguish between the different Backdoor families. However, in SC the best clustering was 1.0 (the highest score possible) and most scores produced were close to 1. This means that this model has well defined clusters. However, 5 experiments had the same value of 1.0. This is not beneficial as this means that a programmer can use any of these experiments, despite them being different to each other in terms of the number of dimensions in a vector and the features abstracted from the behavioural profiles.

The table below (Table 4.28) shows the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up. For example, for the Tri-gram Frequency Vector with Full Representation, it is the best FMS score so will yield a score of 1 whereas Di-gram Bit Vector with Full Representation is the worst FMS score so will yield a score of 12. This table is ordered from lowest (best) to highest (worst) total score.

Vector	FMS	F1	ARI	SC	Total
Di-gram Full Representation Frequency Vector	2	3	2	4	11
Tri-gram Full Representation Frequency Vector	1	4	6	3	14
Uni-gram Full Representation Bit Vector	9	2	3	1	15
Uni-gram Full Representation Frequency Vector	7	1	4	5	17
Tri-gram Full Representation Bit Vector	10	6	5	1	22

Di-gram Category Bit Vector	11	5	8	1	25
Di-gram Full Representation Bit Vector	12	12	1	1	26
Di-gram Category Frequency Vector	4	7	9	7	27
Uni-gram Category Frequency Vector	5	8	7	8	28
Uni-gram Category Bit Vector	8	10	10	1	29
Tri-gram Category Frequency Vector	3	9	12	6	30
Tri-gram Category Bit Vector	6	11	1	2	30

Table 4.28: This table displays the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up.

This table (Table 4.28) shows that overall the best clustering method is Di-gram with the system call representation of Full Representation and vector representation of Frequency Vector.

4.3 Results of Clustering Trojan

In total, 12 experiments were run for each system call representation and feature vector representation. The results for each experiment are detailed below:

There are 5 different malware families from the Ramilli[41] dataset with 5298 evidences (includes duplicates) in total. The table below (Table 4.29) displays the Trojan families and the number of evidences for each family:

Name of Family	Number of Evidences
Trojan.Bladabindi-1201171	372
Trojan.Destover-Sony-1201172	476
Trojan.Dropper.Gen-1201173	620
Trojan.Loadmoney-1201174	236
Trojan.NSIS.Win32-1201175	363
Trojan.Regina-1201176	200
Trojan.Regina-1201177	162
Trojan.Regina-1201178	168
Trojan.Regina-1201179	199
Trojan.Regina-1201180	186
Trojan.Regina-1201181	88
Trojan.Regina-1201182	83
Trojan.Regina-1201183	252
Trojan.Regina-1201184	49
Trojan.Regina-1201185	42
Trojan.Regina-1201186	215
Trojan.Regina-1201187	83
Trojan.Regina-1201188	49
Trojan.Shylock.Skype-1201189	819
Trojan.StabUniq-1201190	303
Trojan.Win32.Bechiro.BCD-1201191	333

Table 4.29: This table displays all the Trojan families and the number of evidences for each family.

There were 5298 different feature vectors for each experiment. The number of system call dimensions in the vectors changed depending on the system call representation and type of feature vector used. Please see Appendix B for text files which show the results each experiment across the three different types of malware.

4.3.1 System Call Dimensions

Trends can be seen from the results. Within a specific n-gram, the more semantics being represented (e.g. full representation compared to category), the higher the number of dimensions in a vector and the larger the feature space. For example, the table below (Table 4.30) shows this for the Uni-grams.

Vector	System Call Dimensions
Full Representation	27
Category	8

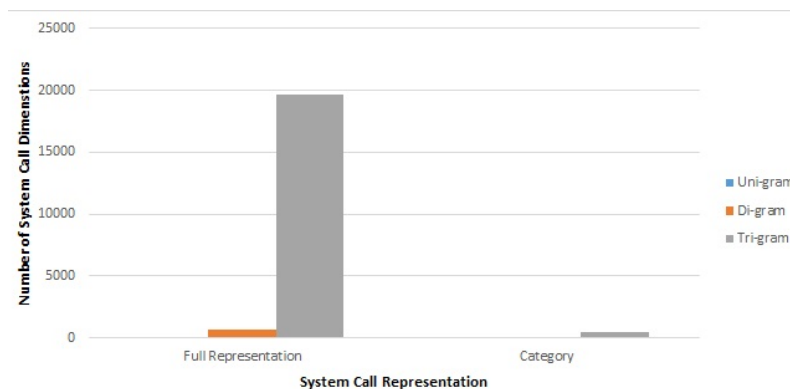
Table 4.30: This table displays that within a specific n-gram, the more semantics being represented the higher the number of dimensions in a vector and the larger the feature space.

Secondly, for each n-gram the higher the number of system calls per dimension (e.g. 1-gram = 1 system call per dimension), the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector. For example, the table (Table below 4.31) shows this for the Uni-gram Category Bit Vectors. This is because the feature space increases and the number of dimensions of the feature vectors are larger.

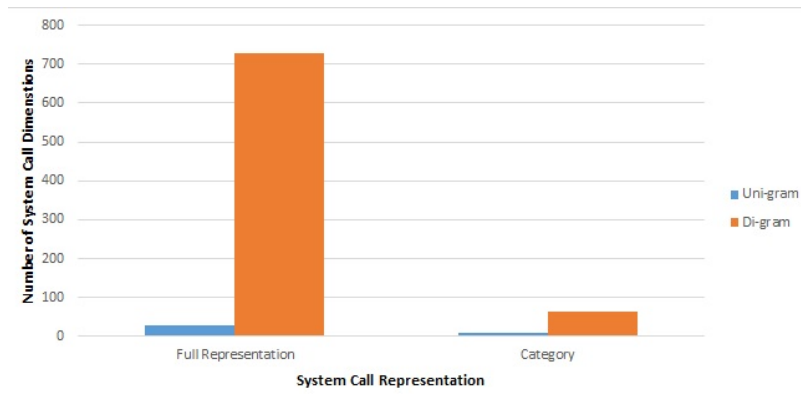
Vector	System Call Dimensions
Uni-gram	8
Di-gram	64
Tri-gram	512

Table 4.31: This table displays the higher the number of system calls per dimension, the higher the number of dimensions in a vector.

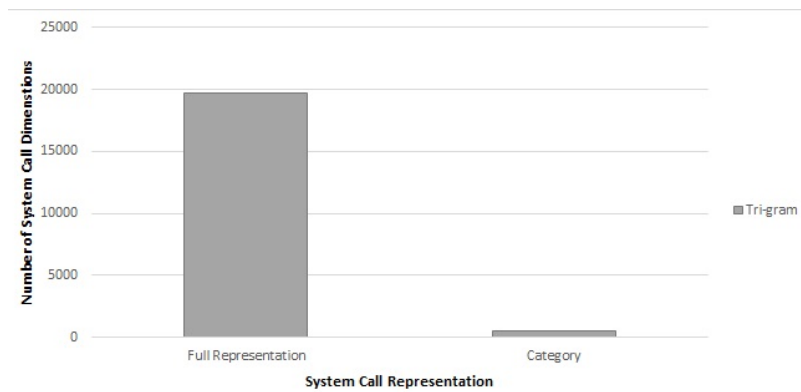
The graph below (Figure 4.9) illustrates these trends. You will see three graphs. The first shows the Uni-grams, Di-grams and Tri-grams all on one graph, the second shows the Uni-grams and Di-grams and the third one shows the Tri-grams on their own. This was done because the number of system call dimensions were much larger for the Tri-grams compared to the Uni-grams and Di-grams so one cannot see what the graph is trying to illustrate. These graphs illustrate that the more semantics that were being represented within an n-gram, the higher the number of system calls per dimension and the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector.



(a) The Uni-grams, Di-grams and Tri-grams.



(b) The Uni-grams and Di-grams.



(c) The Tri-grams.

Figure 4.9: These graphs illustrate that the more semantics that were being represented (higher number of system calls per dimension) within an n-gram, the higher the number of dimensions in a vector.

4.3.2 Euclidean Distance

The tables below (Tables 4.32, 4.33, 4.34 and 4.35) display the Euclidean Distance matrices for each system call representation and type of vector. The columns are the distances being represented, the rows are the type of n-gram and the cells represent the number of evidences that had the distance. A small distance between two evidences means that the two evidences are similar to one another (dissimilar for large distances). A distance of 0 between 2 evidences means that the two evidence feature vectors are the same:

1. Full Representation

(a) Bit Vector

Vector	0	1	1-2	2-3	3-4	Largest Distance
Uni-gram	5255	29	13	0	0	1.732
Di-gram	523	49	9	6	0	2.828
Tri-gram	5238	36	13	8	2	3.464

Table 4.32: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Full Representation and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0-5	5-10	10-20	20-50	50-100	100-150	150-200	200-250	Largest Distance
Uni-gram	5187	400	15	10	10	4	0	0	0	78.318
Di-gram	5166	76	11	13	21	9	0	1	0	172.931
Tri-gram	5187	50	11	6	16	19	5	2	1	220.511

Table 4.33: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Full Representation and Vector Representaion of a Frequency Vector.

2. Category

(a) Bit Vector

Vector	0	1	1-2	Largest Distance
Uni-gram	4333	11	2	1.414
Di-gram	4332	14	0	1.0
Tri-gram	4334	11	0	1.414

Table 4.34: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Category and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0 - 5	5 - 10	10 - 60	60 - 150	Largest Distance
Uni-gram	4299	39	5	3	0	54.746
Di-gram	4306	20	8	12	0	55.678
Tri-gram	4315	17	3	10	1	76.145

Table 4.35: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Category and Vector Representaion of a Frequency Vector.

Three trends can be seen. Within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances (especially at 0). In comparison, frequency vectors that have the same n-gram and system call representation have larger range of distances and the largest distances were bigger. For example the table below (Table 4.36) shows this with for Uni-gram Full Representation Vectors (this also applies to Di-grams and Tri-grams). This is because the distance between the feature vectors are larger and so there are many more distances and less vectors that are similar.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Bit Vector	5255	0, 1 and between 1 and 2	1.7
Frequency Vector	5187	0, between 0 and 5, between 5 and 10, between 10 and 20, between 20 and 50 and between 50 and 100	78

Table 4.36: This table displays that within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances compared to frequency vectors.

The same trend was seen within an n-gram (Uni-gram, Di-gram, Tri-gram). Within an n-gram with the same vector representation (bit or frequency), the more semantics that is represented in the feature vector (full representation), the higher the number of evidences at lower distances (especially at 0). In comparison to the same n-gram with the same feature vector representation but less semantics are represented. In this case, less distances are represented and the largest distances were lower. For example, the table below (Table 4.37) shows this for Uni-gram Bit Vectors (this also applies to Di-grams and Tri-grams). This is because the feature space decreases and the number of dimensions of the feature vectors are smaller and the distance between the feature vectors are smaller and so there are less distances and more vectors that are similar.

System Call Representation	Number of evidences at 0	Distances Represented	Largest Distance
Full Representation	5255	0, 1 and between 1 and 2	1.7
Category	4333	0, 1 and between 1 and 2	1.4

Table 4.37: This table displays that less distances were represented and the largest distances were lower.

Another trend could be seen with respect to the type of n-gram being represented. With the same system call representation and type of vector representation (bit or frequency), the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower the number of evidences at lower distances (especially at 0). When there were higher numbers of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram) with the same system call representation and type of vector, more distances were represented and the largest distances were bigger. For example, the table below (Table 4.38) shows this for Full Representation Bit Vectors (this was the same for the category system call representation). This is because the feature space increases and the number of dimensions of the feature vectors are larger and the distance between the feature vectors are larger and so there are many more distances and less vectors that are similar.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Uni-gram	5255	0, 1 and between 1 and 2	1.7
Di-gram	5233	0, 1, between 1 and 2 and between 2 and 3	2.8
Tri-gram	5238	0, 1, between 1 and 2, between 2 and 3 and between 3 and 4	3.5

Table 4.38: This table displays that with the same system call representation and type of vector representation, the lower the number of system calls per dimension, the lower the number of evidences at lower distances.

An exception to this is that with the system call representation of category and vector representation of a bit vector, the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower the number of evidences at lower distances (especially at 0). However, when there were higher numbers of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram) with the same system call representation and type of vector, less distances were represented and the largest distances were lower or the same as the Uni-gram. For example, the table below (Table 4.39) shows this with Trojans for the Full Representation bit vectors.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Uni-gram	4333	0, 1 and between 1 and 2	1.4
Di-gram	4332	0 and 1	1.0
Tri-gram	4334	0 and 1	1.4

Table 4.39: This table displays that the lower the number of system calls per dimension in an n-gram the lower the number of evidences at lower distances.

This graph below (Figure 4.10) shows that within an n-gram, eventhough the system call representations were the same (Full Representation), the number of evidences that had a distance 0 was higher with the bit vectors than the frequency vectors.

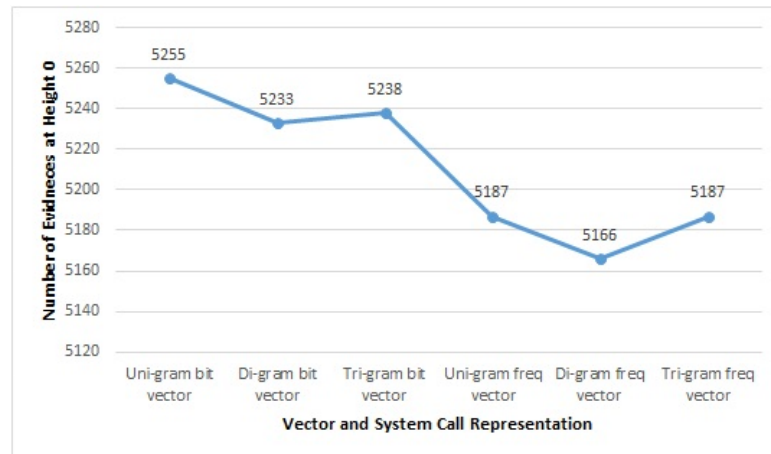
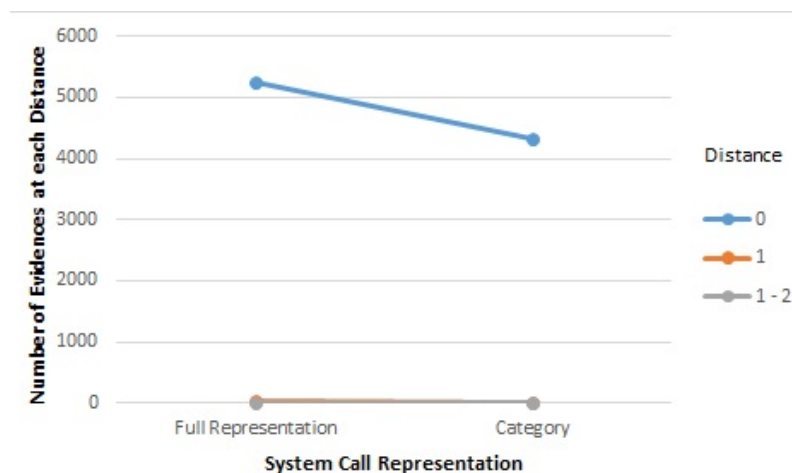
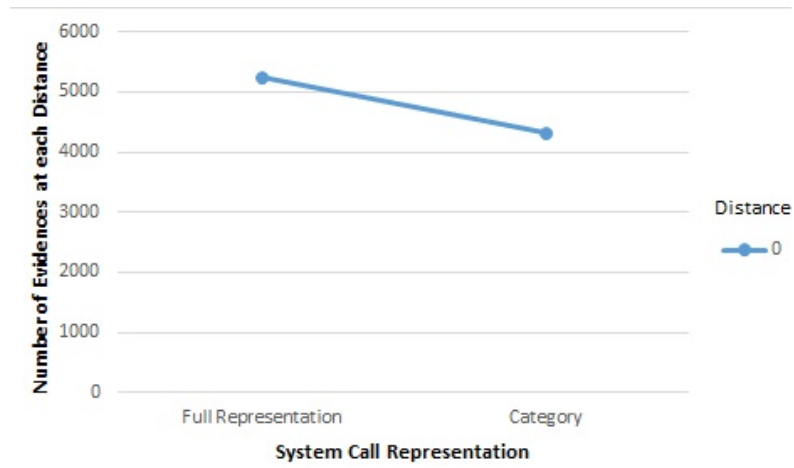


Figure 4.10: This graph illustrates that the number of evidences that had a distance 0 was higher with the bit vectors than the frequency vectors.

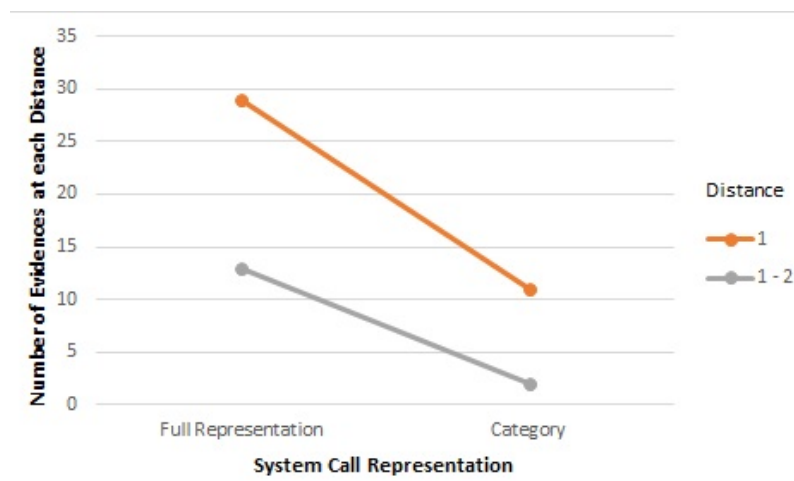
The graphs below (Figure 4.11) show that the fewer the semantics represented in the feature vector by the system call representations (fewer semantics is the category representation), the lower the number of evidences at lower distances compared to the more semantics represented. You will see three graphs. The first shows all the distances all on one graph, the second shows just the distance 0 and the third graph shows all the distances except 0. This was done because the number of evidences was much larger for the distance 0 compared to the other distances so one cannot see what the graph is trying to illustrate.



(a) Distances 0, 1 and 1-2.



(b) Distance 0.



(c) Distances 1 and 1-2.

Figure 4.11: These graphs illustrate that the fewer the semantics represented in the feature vector by the system call representations (fewer semantics is the category representation), the lower the number of evidences at lower distances compared to the more semantics represented.

The graph below (Figure 4.12) shows that the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the higher number of evidences at lower distances (especially at 0).

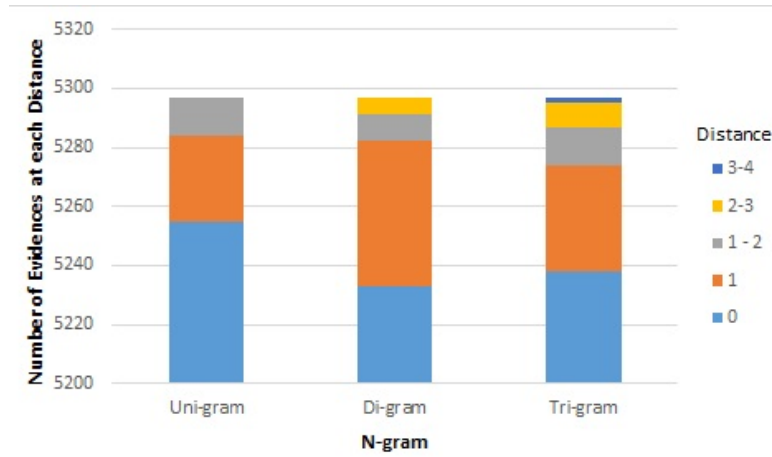


Figure 4.12: The graph illustrates that the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the higher the number of evidences at lower distances (especially at 0).

These trends can be seen because when more features are being represented, the feature space increases and the number of dimensions of the feature vectors are larger and the distance between the feature vectors are larger and so there are many more distances and less vectors that are similar.

4.3.3 Validation Metrics

The table below (Table 4.40) displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at. (Red indicates the highest FMS, F1, ARI or SC score for each n-gram):

Vector and Syscall Representation	Best FMS	Best F1	Best ARI	Best SC
Uni-gram, Full Representation, Bit Vector	0.274 at height 4.173	0.154 at height 4.173	0.0002 at height 0.0	0.998 at height 1.0
Uni-gram, Full Representation, Frequency Vector	0.274 at height 9.356	0.155 at height 1.208	0.002 at height 0.0	0.987 at height 1.0
Uni-gram, Category, Bit Vector	0.237 at height 0.0	0.145 at height 0.0	0.001 at height 0.0	1.0 at height 1.0
Uni-gram, Category, Frequency Vector	0.291 at height 17.160	0.185 at height 1.0	0.001 at height 0.0	0.997 at height 1.0
Di-gram, Full Representation, Bit Vector	0.274 at height 37.335	0.154 at height 22.100	8.4e-05 at height 22.100	0.995 at height 1.0
Di-gram, Full Representation, Frequency Vector	0.274 at height 21.224	0.154 at height 4.834	0.0004 at height 0.0	0.997 at height 1.0
Di-gram, Category, Bit Vector	0.271 at height 1.0	0.175 at height 1.0	0.0001 at height 1.0	0.999 at height 1.0
Di-gram, Category, Frequency Vector	0.291 at height 17.160	0.186 at height 19.381	0.0006 at height 4.225	0.997 at height 1.0
Tri-gram, Full Representation, Bit Vector	0.274 at height 37.335	0.154 at height 4.173	8.5e-05 at height 37.335	0.994 at height 1.0
Tri-gram, Full Representation, Frequency Vector	0.274 at height 4.834	0.155 at height 1.414	0.0003 at height 69.821	0.987 at height 5.475
Tri-gram, Category, Bit Vector	0.291 at height 0.0	0.185 at height 0.0	7.6e-05 at height 0.0	0.999 at height 1.0
Tri-gram, Category, Frequency Vector	0.291 at height 19.381	0.186 at height 10.329	0.0003 at height 10.329	0.997 at height 1.0

Table 4.40: This table displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at.

Looking at this table (Table 4.40), the table below (Table 4.41) shows the experiments with the best clustering according to FMS, F1, ARI and SC.

Metric	N-gram, System Call Representation and Vector Representation	Score and Height
FMS	Tri-gram Category Frequency Vector	0.291 at height 19.381
F1	Tri-gram Category Frequency Vector	0.186 at height 10.329
ARI	Uni-gram Full Representation Frequency Vector	0.002 at height 0.0
SC	Uni-gram Category Bit Vector	1.0 at height 1.0

Table 4.41: This table displays that the best clustering according to FMS, F1, ARI and SC were all different.

In all evaluation metrics, the best clustering method that was chosen was different (see above). The method of feature selection (system call representation) was the same in FMS, F1 and SC category). The method of model construction (feature vector representation) was the same in FMS and F1 and ARI (frequency vector), it was the type of n-gram that was different and the scores produced were different. These experiments are completely different to each other in terms of the number of dimensions in a vector (8 for Uni-gram, 64 for Di-gram and 512 for Tri-gram) and the features abstracted from the behavioural profiles. The type of n-gram was different and the scores produced were different. Only FMS and F1 had the highest score at the same experiment (system call representation, feature vector representation and n-gram). The highest scores for each n-gram were frequency vectors and category with a few exceptions in the ARI (system call representation) and SC (feature vector representation).

In the FMS, F1 and ARI metrics, the scores produced were not high as they were closer to 0. Even though each metric calculated the best method of feature selection and model construction for each dendrogram cut which produced the best clustering, these experiments were not similar to the dataset[41] (true assignments) so these results should be taken with caution. As the scores produced were not near to 1 and the same experiment did not produce the best clustering in all FMS, F1 and ARI, if a clustering system used either one of the best clustering methods and was presented with another dataset, there is no guarantee that the system will group samples from the same Trojan families in the same clusters so the user will not be able to distinguish between the different Trojan families. However, in SC the best clustering was 1.0 (the highest score possible) and most scores produced were close to 1. This means that this model has well defined clusters.

The table below (Table 4.42) shows the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up. For example, for the Tri-gram Frequency Vector with Category, it is the best FMS score so will yield a score of 1 whereas Uni-gram Bit Vector with Category is the worst FMS score so will yield a score of 12. This table is ordered from lowest (best) to highest (worst) total score.

Vector	FMS	F1	ARI	SC	Total
Tri-gram Category Frequency Vector	1	1	6	5	13
Di-gram Category Frequency Vector	3	2	3	7	15
Uni-gram Category Frequency Vector	4	3	4	6	17
Tri-gram Category Bit Vector	2	4	12	3	21
Di-gram Category Bit Vector	11	5	9	2	27
Uni-gram Category Bit Vector	12	12	2	1	27
Tri-gram Full Representation Frequency Vector	5	6	7	10	28
Uni-gram Full Representation Bit Vector	8	9	8	4	29
Uni-gram Full Representation Frequency Vector	10	7	1	11	29
Di-gram Full Representation Frequency Vector	7	8	5	12	32

Tri-gram Full Representation Bit Vector	6	11	10	9	36
Di-gram Full Representation Bit Vector	9	10	11	8	38

Table 4.42: This table displays the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up.

This table (Table 4.42) shows that overall the best clustering method is Tri-gram with the system call representation of Category and vector representation of Frequency Vector.

5

Discussion

This chapter compares the clustering of the three malware families and evaluates the accuracy of clustering-based malware.

In particular:

Section 5.1 critically compares the clustering of Ransomware, Backdoors and Trojans by looking at all the similarities(5.1.1) and differences(5.1.2) between them such as the dataset, features, methods, system call dimensions, euclidean distances, and validation metrics using results obtained from my program as examples.

Section 5.2 discusses the accuracy of clustering-based malware detection using my results and results from other papers, limitations (running times) and future work.

Section 5.3 evaluates my project, discusses the changes made since my project description form, challenges and what I have learnt.

5.1 Comparison between Clustering Ransomware, Backdoor and Trojan

The clustering of Ransomware, Backdoors and Trojans were compared because they can work alongside each other to infect a victim's system. Ransomware can be used to install Backdoors into a system and Trojans can be used to deliver Backdoors or Ransomware into a system.

5.1.1 Similarities

Additionally there are many similarities between Ransomware, Backdoor and Trojans.

Firstly, all the samples of Ransomware, Backdoor and Trojans were from the same dataset[41] and this dataset was produced 3 years ago (December 2016). It represents features extracted from the sample analysis using static and dynamic analysis. I used all the samples available in the dataset (15 Ransomware families, 5 Backdoor families and 9 Trojan families with several sanitized evidences in each family). Please see Description of Program and Justification of Feature Selection and Validation Methods for the rationale for using this dataset.

Secondly, the features within the json files for the different malware families were the same. They all had features consisting of the categories file, pe, reg, sig, str, api, cmd or mutex. As the features were the same, I could compare the accuracy of the clustering between these three types of malware with the different feature selection and vector representation methods. Please see Description of Program and Justification of Feature Selection and Validation Methods for the rationale for using this dataset.

Thirdly, everything in my program (feature selection methods, vector representations and n-grams) was kept the same, it was only the dataset (whether it was Ransomware, Backdoor or Trojan) that changed. Therefore, for each malware family, 12 experiments were conducted (2 experiments per system call representation for bit and frequency vectors per N-gram = $2 * 2 * 3 = 12$). This is because for each N-gram type, of which there are 3, there are two system call representations to be represented and each vector is a bit or frequency vector. Therefore, once I performed feature selection, model construction, hierarchical clustering and evaluation for each malware type, I could compare the accuracy of clustering amongst these three types of malware families because my analysis was kept the same.

Fourthly, with Ransomware and Trojans, the number of system call dimensions for each n-gram for the category feature selection method were the same at 8 for Uni-grams, 64 for Di-grams and 512 for Tri-grams.

Similarities in Results

By doing this comparison, similarities within the results were found.

Firstly, with the system call dimensions (Table 5.1 and 5.2). For Ransomware, Backdoor and Trojans, within a specific n-gram, the more semantics being represented (e.g. full representation compared to category), the higher the number of dimensions in a vector and the larger the feature space. For example, the table below (Table 5.1) shows this with Ransomware and Backdoor for the Uni-grams.

	Vector	System Call Dimensions
Ransomware	Full Representation	45
	Category	8
Backdoor	Full Representation	7
	Category	3

Table 5.1: This table displays that with Ransomware and Backdoor within a specific n-gram, the more semantics being represented the higher the number of dimensions in a vector and the larger the feature space.

Also for each n-gram the higher the number of system calls per dimension (e.g. 1-gram = 1 system call per dimension), the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector. For example, the table below (Table 5.2) shows this with Ransomware and Trojan for the Uni-gram Category Bit Vectors. This is because the feature space increases and the number of dimensions of the feature vectors are larger.

	Vector	System Call Dimensions
Ransomware	Uni-gram	8
	Di-gram	64
	Tri-gram	512
Trojan	Uni-gram	8
	Di-gram	64
	Tri-gram	512

Table 5.2: This table displays that with Ransomware and Trojan the higher the number of system calls per dimension, the higher the number of dimensions in a vector.

Secondly, with the Euclidean Distances (Table 5.3). For Ransomware, Trojan and Backdoor, within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances (especially at 0). In comparison, frequency vectors that have the same n-gram and system call representation have larger range of distances and the largest distances were bigger. For example, the table below (Table 5.3) shows this with Ransomware and Backdoor for Uni-gram Full Representation Vectors (this also applies to Di-grams and Tri-grams). This is because the distance between the feature vectors are larger, so there are more distances and less vectors that are similar.

	Vector	Number of evidences at 0	Distances Represented	Largest Distance
Ransomware	Bit Vector	10646	0, 1, between 1 and 2 and between 2 and 3	2
	Frequency Vector	10524	0, between 0 and 5, between 5 and 10, between 10 and 20, between 20 and 50, between 50 and 100, between 100 and 150 and between 150 and 300	104
Backdoor	Bit Vector	4995	0, 1 and between 1 and 2	1.4
	Frequency Vector	4986	0, between 0 and 5, between 5 and 10 and between 10 and 20	14.8

Table 5.3: This table displays that with Ransomware and Backdoor within a specific n-gram and system call representation, the bit vector has a higher number of evidences at lower distances compared to frequency vectors.

Another similarity with the Euclidean Distances was found. In Ransomware, Backdoors and Trojans, with the same system call representation and type of vector representation (bit or frequency), the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower the number of evidences at lower distances (especially at 0). When there were higher numbers of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram)

with the same system call representation and type of vector, more distances were represented and the largest distances were bigger. For example, the table below (Table 5.4) shows this with Ransomware and Trojan for Full Representation Bit Vectors (this also applies to the category representation). This is because the distance between the feature vectors are larger, so there are more distances and less vectors that are similar.

	Vector	Number of evidences at 0	Distances Represented	Largest Distance
Ransomware	Uni-gram	10646	0, 1, between 1 and 2 and between 2 and 3	2
	Di-gram	10611	0, 1, between 1 and 2, between 2 and 3 and between 3 and 4	3.2
	Tri-gram	10588	0, 1, between 1 and 2, between 2 and 3 and between 3 and 4	3.6
Trojan	Uni-gram	5255	0, 1 and between 1 and 2	1.7
	Di-gram	5233	0, 1, between 1 and 2 and between 2 and 3	2.8
	Tri-gram	5238	0, 1, between 1 and 2, between 2 and 3 and between 3 and 4	3.5

Table 5.4: This table displays that for Ransomware and Trojan with the same system call representation and type of vector representation, the lower the number of system calls per dimension, the lower the number of evidences at lower distances.

Fourthly, with the validation metrics, in Ransomware, Backdoors and Trojans, the scores produced were not high as they were closer to 0. For example the table below (Table 5.5) shows the experiments with the best clustering for Backdoor and Trojan according to FMS, F1 and ARI. Even though each metric calculated the best method of feature selection and model construction which produced the best clustering, these experiments were not similar to the dataset[41] so these results should be taken with caution. This means that if a clustering system used either one of the best clustering methods and was presented with another dataset, there is no guarantee that the system will group samples from the same malware families into the same clusters so the user will not be able to distinguish between the different malware families. However, in SC the best clustering was 1.0 (the highest score possible) and most of the scores that were produced were close to 1. This means that this model has well defined clusters.

	Metric	Score
Backdoor	FMS	0.457
	F1	0.223
	ARI	0.003
Trojan	FMS	0.291
	F1	0.186
	ARI	0.002

Table 5.5: This table displays the experiments with the best clustering for Backdoor and Trojan according to FMS, F1 and ARI.

5.1.2 Differences

There were differences between Ransomware, Backdoor and Trojans.

Firstly, the number of samples and sanitized evidences for each family. There were 15 Ransomware families with 10727 evidences, 5 Backdoor families with 5003 evidences and 9 Trojan families with 5298 evidences. One could argue that this dataset is unbalanced e.g. 15 vs 5 families. However, as all the features were the same and the whole program was the same for each family, it does not matter whether one malware family had more samples or evidences than another and this should not affect the results.

Secondly with Backdoors, the number of system call dimensions for each n-gram for the category feature selection method was different to that of Ransomware and Trojans at 3 for Uni-grams, 9 for Di-grams and 27 for Tri-grams compared to 8 for Uni-grams, 64 for Di-grams and 512 for Tri-grams with Ransomware and Trojans. This however should not affect the results because even though the system call dimensions are different, it does not affect the accuracy of the clusterings as that is dependent on the Euclidean Distances and cuts of the dendrogram and not the number of system call dimensions. This was because of the number of different operations that existed in the files.

Thirdly, the true-assignments, the feature vectors, the evidences, Euclidean Distances, the heights of the dendrogram and the FMS, F1, ARI and SC scores differed between the malware families. This was expected as each malware family has different evidences, each evidence has different features observed and the number of features observed are different. This means that the Euclidean Distances, the dendrograms and the cuts on the dendrogram will differ. Therefore, the FMS, F1, ARI and SC scores will be different.

Differences in Results

Due to these differences, there were differences within the results.

Firstly, with the Euclidean Distances. By comparing the distances represented in Ransomware, Trojans and Backdoor, it was found that Backdoor and Trojans had most of the evidences at lower distances (especially at 0), whereas Ransomware had larger ranges of distances and the largest distances were bigger. For example, the table below (Table 5.6) shows this with Uni-gram Full Representation Bit Vectors (Frequency Vector for Backdoor) (this was the same for all feature selection methods, vector representations and types of n-grams). This could be because there were many more files and evidences in Ransomware (15 families with 10727 evidences) compared to Backdoor (5 families with 5003 evidences) and Trojan (21 families with 5298 evidences). Please see Appendix C for the collated results of the Euclidean Distances across the three different types of malware.

	Number of evidences at 0	Distances Represented	Largest Distance
Ransomware	10646	0, 1, between 1 and 2 and between 2 and 3	2
Backdoor	4995	0, 1 and between 1 and 2	1.4
Trojan	5255	0, 1 and between 1 and 2	1.7

Table 5.6: This table displays that Backdoor and Trojans had most of the evidences at lower distances, whereas Ransomware had larger ranges of distances and the largest distances were bigger.

Another difference was found with the Euclidean Distances. Within an n-gram (Uni-gram, Di-gram, Tri-gram) that has same vector representation (bit or frequency), it was found that the more semantics that are represented in the feature vector (full representation), the higher the number of evidences at lower distances (especially at 0). This pattern was found for all three types of malware. However different results were found between the three types of malware with the same n-gram and feature vector representation but less semantics. For Ransomware and Trojans less distances were represented and the largest distances were lower but in Backdoor more distances were represented and the largest distances were higher. For example, the table below (Table 5.7) shows this with Ransomware and Backdoors for the Uni-gram Bit Vectors. This is because in Ransomware and Trojans, the feature space decreases, the number of dimensions in the feature vectors are smaller and the distance between the feature vectors are smaller, so there are less distances and more vectors that are similar. However in Backdoor, even though the feature space decreases and the number of dimensions in the feature vectors are smaller, these results suggests that the distance between the feature vectors are larger so there are more distances and less vectors that are similar.

	System Call Representation	Number of evidences at 0	Distances Represented	Largest Distance
Ransomware	Full Representation	10646	0, 1, between 1 and 2 and between 2 and 3	2
	Category	9846	0, 1 and between 1 and 2	1.4
Backdoor	Full Representation	4986	0, between 0 and 5, between 5 and 10 and between 10 and 20	14.8
	Category	4865	0, between 0 and 5, between 5 and 10 and between 10 and 60	16.5

Table 5.7: This table displays that for Ransomware less distances were represented and the largest distances were lower but in Backdoor more distances were represented and the largest distances were higher.

Another difference was found with the Euclidean Distances. With Trojans but not Ransomware or Backdoor, it was found that for the system call representation of category and vector representation (bit), the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the lower the number of evidences at lower distances (especially at 0). However, when there were higher numbers of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram) with the same system call representation and type of vector, less distances were represented and the largest distances were lower or the same as the Uni-gram. For example, the table below (Table 5.8) shows this with Trojans for the Full Representation bit vectors.

Vector	Number of evidences at 0	Distances Represented	Largest Distance
Uni-gram	4333	0, 1 and between 1 and 2	1.4
Di-gram	4332	0 and 1	1.0
Tri-gram	4334	0 and 1	1.4

Table 5.8: This table displays that the lower the number of system calls per dimension in an n-gram the lower the number of evidences at lower distances.

Another difference was found with the Euclidean Distances. It was found that with Backdoors but not Ransomware or Trojans, when there were different system call representations but the same vector representation of a bit vector and n-gram, the same distances were represented and the largest distances were the same. For example, the table below (Table 5.9) shows this with Ransomware and Backdoor for the Uni-gram bit vectors.

	System Call Representation	Number of evidences at 0	Distances Represented	Largest Distance
Ransomware	Full Representation	10646	0, 1, between 1 and 2 and between 2 and 3	2
	Category	9846	0, 1 and between 1 and 2	1.4
Backdoor	Full Representation	4995	0, 1 and between 1 and 2	1.4
	Category	4870	0, 1 and between 1 and 2	1.4

Table 5.9: This table displays that for Backdoors but not Ransomware, when there were different system call representations but the same vector representation of a bit vector and n-gram, the same distances were represented and the largest distances were the same.

Secondly, with Backdoors but not Ransomware or Trojans, it was found that the higher the number of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram), with the same system call representation and type of vector, less distances were represented and the largest distances were

lower. For example, the table below (Table 5.10) shows this with Ransomware and Backdoor for the Full Representation bit vectors.

	Vector	Number of evidences at 0	Distances Represented	Largest Distance
Ransomware	Uni-gram	10646	0, 1, between 1 and 2 and between 2 and 3	2
	Di-gram	10611	0, 1, between 1 and 2, between 2 and 3 and between 3 and 4	3.2
Backdoor	Uni-gram	4995	0, 1 and between 1 and 2	1.4
	Di-gram	4996	0 and 1	1.0

Table 5.10: This table displays that for Backdoors but not Ransomware, the higher the number of system calls per dimension in an n-gram with the same system call representation and type of vector, less distances were represented and the largest distances were lower.

Thirdly, a difference could be found with the validation metrics as the experiments with the best clustering according to FMS, F1, ARI and SC for Ransomware, Backdoors and Trojans were different. The Table below (Table 5.11) shows this with the best clustering according to the FMS, F1, ARI and SC scores. Therefore, these results do not agree on the feature selection (system call representation), n-gram or model construction methods (feature vector representation).

Metric	Malware	N-gram, System Call Representation and Vector Representation
FMS	Ransomware	Uni-gram Category Frequency Vector
	Backdoor	Tri-gram Full Representation Frequency Vector
	Trojan	Tri-gram Category Frequency Vector
F1	Ransomware	Di-gram Full Representation Frequency Vector
	Backdoor	Uni-gram Full Representation Frequency Vector
	Trojan	Tri-gram Category Frequency Vector
ARI	Ransomware	Uni-gram Full Representation Frequency Vector
	Backdoor	Di-gram Full Representation Bit Vector
	Trojan	Uni-gram Full Representation Frequency Vector
SC	Ransomware	Uni-gram Category Bit Vector
	Backdoor	Uni-gram Full Representation Bit Vector, Uni-gram Category Bit Vector, Di-gram Full Representation Bit Vector, Di-gram Category Bit Vector and Tri-gram Full Representation Bit Vector
	Trojan	Uni-gram Category Bit Vector

Table 5.11: This table displays that the best clustering according to FMS, F1, ARI and SC for Ransomware, Backdoors and Trojans were all different.

Finally, with regards to the validation metrics, the best clustering method for each malware family was different. The table below (Table 5.12) shows the best clustering methods for the three malware families.

Malware	N-gram, System Call Representation and Vector Representation
Ransomware	Uni-gram Category Frequency Vector
Backdoor	Di-gram Full Representation Frequency Vector
Trojan	Tri-gram Category Frequency Vector

Table 5.12: This table displays that the best clustering methods for the three malware families.

5.2 Accuracy of Clustering-Based Malware Detection

The accuracy of clustering-based malware detection depends on many factors including the type of machine learning algorithm, the features selected, the feature selection methods, the model construction methods and evaluation metrics. This is illustrated in my results where the different methods of feature selection and vector representation yielded different results (scores and best clustering methods) for the validation metrics. Therefore, the accuracy of clustering-based malware detection is highly subjective as it is dependent on many factors.

My results suggest that clustering-based malware detection has a low validity and accuracy. This is because each different type of malware (Ransomware, Backdoor or Trojan) produced a different best clustering method. Please see the table below (Table 5.13) for the best clustering methods for the three malware families. The model construction method (feature vector representation) is the same but the feature selection (system call representation) and n-gram is different. Therefore, these three types of malware did not agree on the feature selection or n-gram but they did agree on the model construction method.

Malware	N-gram, System Call Representation and Vector Representation
Ransomware	Uni-gram Category Frequency Vector
Backdoor	Di-gram Full Representation Frequency Vector
Trojan	Tri-gram Category Frequency Vector

Table 5.13: This table displays that the best clustering methods for the three malware families.

However, my program does not reflect reality as the true assignments and testing labels were known. In malware detection these will not be known as new malware or variants of previously seen malware (e.g. polymorphism or metamorphism)[9] can be produced. This makes detection hard as one may not be able to identify what malware family the sample belongs to. This means that validation techniques other than FMS, F1 and ARI (like SC) should be used because the ground truth is unknown. The table below (Table 5.14) displays the clustering methods according to the best clustering method determined by SC. This is because they all yielded a score of 1.0 which is the highest possible score that can be obtained. This suggests that these models had well defined non-overlapping clusters and no samples were assigned to the wrong clusters. Therefore, there was no cluster that was more similar to the cluster that the samples were assigned to.

Malware	N-gram, System Call Representation and Vector Representation
Ransomware	Uni-gram Category Bit Vector
Backdoor	Uni-gram Full Representation Bit Vector, Uni-gram Category Bit Vector, Di-gram Full Representation Bit Vector, Di-gram Category Bit Vector and Tri-gram Full Representation Bit Vector
Trojan	Uni-gram Category Bit Vector

Table 5.14: This table displays the clustering methods according to the best clustering method determined by SC.

However, using SC can introduce problems, therefore it should be used with caution. As shown above, for each malware family the best clustering was different and five experiments for Backdoor had the same SC value of 1.0 and this was the highest SC score. This is not beneficial as this means that a programmer can use any of these experiments, despite them being different to each other in terms of the number of dimensions in a vector and the features abstracted from the behavioural profiles.

Secondly, as I had access to the ground truth (labelled dataset) I was able to see if this was correct using three evaluation metrics: FMS, F1 and ARI (which are used when the true labels are known). Scores nearer 1.0 mean that the true class assignments are equal to the predictive assignments (clustering algorithm assignments) and the higher the score (nearer to 1) the more similar the true and predictive assignments are and the lower the score (near to 0), the more dissimilar

the true and predictive assignments are. Scores nearer 0 suggest that the assignments are independent. The table below (Table 5.15) shows the FMS, F1 and ARI for the experiment which produced the best clustering according to SC. It illustrates that even though the SC scores were high, the predictive assignments were not similar to the true assignments because all scores were closer to 0 than 1. This suggests that the clustering assignments were independent.

	N-gram, System Call Representation and Vector Representation	Metric	Score
Ransomware	Uni-gram Category Bit Vector	FMS	0.275
		F1	0.076
		ARI	0.002
Backdoor	Uni-gram Full Representation Bit Vector, Uni-gram Category Bit Vector, Di-gram Full Representation Bit Vector, Di-gram Category Bit Vector and Tri-gram Full Representation Bit Vector	FMS	0.443, 0.453, 0.402, 0.433 and 0.439
		F1	0.221, 0.216, 0.213, 0.218 and 0.218
		ARI	0.003, 0.0002, 0.003, 0.001 and 0.002
Trojan	Uni-gram Category Bit Vector	FMS	0.237
		F1	0.145
		ARI	0.001

Table 5.15: This table displays the FMS, F1 and ARI scores for the experiment which produced the best clustering according to SC.

The table below (Table 5.16) shows the SC scores for the experiment which produced the best clustering according to FMS, F1 and ARI. It shows that the clusterings obtained were not well defined. This could be because there were many overlapping clusters or because there were samples that were assigned to the wrong cluster.

	Metric	N-gram, System Call Representation and Vector Representation	Score
Ransomware	FMS	Uni-gram Category Frequency Vector	0.998
	F1	Di-gram Full Representation Frequency Vector	0.991
	ARI	Uni-gram Full Representation Frequency Vector	0.992
Backdoor	FMS	Tri-gram Full Representation Frequency Vector	0.999
	F1	Uni-gram Full Representation Frequency Vector	0.999
	ARI	Di-gram Full Representation Bit Vector	1.0
Trojan	FMS	Tri-gram Category Frequency Vector	0.997
	F1	Tri-gram Category Frequency Vector	0.997
	ARI	Uni-gram Full Representation Frequency Vector	0.987

Table 5.16: This table displays the SC scores for the experiment which produced the best clustering according to FMS, F1 and ARI.

Taking all these results into consideration, these results suggest that if either FMS, F1 or ARI have high scores (obtain a best clustering) the SC scores will be lower (not 1.0) and if the SC scores are high, the FMS, F1 and ARI scores will be low (less than 0.5). These evaluation metrics do not agree on the best clusterings, therefore it can be concluded that clustering-based malware detection has a low validity and accuracy.

This conclusion is supported by my “Clustering Android Application Behaviour” BSc Project. In this project most of the program aspects were the same (e.g. the process of clustering such as the vector representations, dissimilarity metrics, hierarchical clustering and evaluation metrics). The only aspect that differed was the system call representations (feature selection methods) as Ransomware and Android have different system calls and behaviours (android had system calls and ioctl calls such as Binder or Intent Transactions and the Ransomware dataset consisted of categories such as file, signature, api and registry). However, the results (please see Appendix D for the full list of results) draw the same conclusions, as detailed above.

To summarise the results from Appendix D, the table below (Table 5.17) shows the FMS, F1 and ARI scores for the experiment which produced the best clustering according to SC. This shows that the predictive assignments were not similar to the true assignments. The table below (Table 5.18) shows the SC scores for the best clusterings that were determined by FMS, F1 and ARI. Therefore, as stated above, these results suggest that clustering-based malware detection has a low validity and accuracy.

Metric	Score
FMS	0.150
F1	0.035
ARI	0.0004

Table 5.17: This table displays the SC scores for the experiment which produced the best clustering according to FMS, F1 and ARI.

Metric	N-gram, System Call Representation and Vector Representation	Score
FMS	Di-gram Composite Behaviours and Ioctl calls Bit Vector	0.116
F1	Uni-gram Composite Behaviours and Ioctl Calls Frequency Vector	0.872
ARI	Di-gram Composite Behaviours and Binder Semantics Frequency Vector	0.388

Table 5.18: This table displays the SC scores for the experiment which produced the best clustering according to FMS, F1 and ARI.

However, some researchers have found a high accuracy and validity for clustering-based malware.

For example, Bayer et al.[20] produced results with a precision of 0.984 and a recall of 0.930 when using dynamic analysis, extraction of behavioural profile (feature selection) and hierarchical clustering. This suggests that the clustering that they produced was very close to the reference set (true assignments) and the system could differentiate the different malware classes as the majority of the samples were grouped into the same family.

Hamid et al.[38] used k-means clustering to separate the android malware into ransomware, scareware or goodware depending on 2 features, presence or absence of locks and encryption. If the trace had a lock and encryption then it was classified as ransomware, if the trace had a lock but not encryption then it was classified as scareware and if the trace did not have both a lock and encryption then it was classified as goodware. It was performed on two datasets with one having accuracy at 98.1% and the other at 74.7%. This means that clustering malware can be accurate but it can be made more accurate depending on the number features used. Here only 2 features were used (presence or absence of locks and encryption), so in theory the more features used the more accurate the classification will be.

However, different analysis methods produce different features. For example, if Cuckoo sandbox was used to perform the analysis on the malware samples, many features can be produced to perform feature selection, model construction and hierarchical clustering on. If there are many features then the dimensionality of the feature vectors will be very high. However, it could be that many of these features are irrelevant and only some fully distinguish one sample from another. Therefore, one should use a dimensionality reduction algorithm (e.g. PCA) to only select the features that are representative of the malware family. The feature vectors I used had a high dimensionality (e.g. 91125 system call dimensions for Ransomware Tri-grams Full Representation), many dimensions of the feature vectors has the value 0 (feature (system call) not present) and as I did not use a dimensionality reduction algorithm they could contain irrelevant features. This means if I had used a dimensionality reduction algorithm, my results may have been more accurate. So future work should run a dimensionality reduction algorithm on my program before hierarchical clustering to see whether my results become more accurate.

Also, Zhang et al.[25] performed classification of Ransomware families with machine learning based on N-grams of opcodes. Please see Feature Selection and Model Construction Report for a

description of their feature selection method. Their different classifiers produced accuracy results in the range of 46% - 91% (Decision Tree 83.42% to 86.57%, K-Nearest Neighbour 82.52% to 88.19 Random Forest 85.15% to 91.43%, Naive Bayes 45.89% to 70.34% and Gradient Boosting Decision Tree 84.03% to 89.98%). Therefore, the research by Bayer et al.[20], Hamid et al.[38] and Zhang et al.[25] suggests that clustering based malware detection is accurate.

A reason for the differences in results can be explained by the analysis, feature selection, model construction methods and machine learning methods as they all differed between the papers. In both my BSc project and MSc project I used the operations from the CopperDroid[43] (dynamic analysis) or Ramilli[41](static and dynamic analysis) datasets as my features for the vectors and then used the operation or the category of the operation to construct the bit or frequency n-gram feature vectors (uni-grams, di-grams and tri-grams). I used an unsupervised machine learning method called hierarchical clustering in order to perform my malware classification (please see Description of Program and Justification of Feature Selection and Validation Methods for a description of my feature selection and model construction process). In contrast Bayer et al.[20] used their own ANUBIS system (please see [44] for a detailed description of their system) to perform dynamic analysis using data tainting to gather execution traces of malware programs, extraction of traces into behavioural profiles (feature selection) and then hierarchical clustering and Hamid et al.[38] used k-means clustering. However, Zhang et al.[25] performed static analysis to analyse the Ransomware samples using the IDAPRO Disassembler and then converted the opcodes into n-grams (1-gram, 2-gram, 3-gram and 4-gram) and constructed feature vectors using the term frequency-Inverse document frequency (TF-IDF) for each n-gram. They used supervised machine-learning methods (Decision Tree, Random Forest, K-Nearest Neighbour, Naive Bayes, and Gradient Boosting Decision Tree) to perform Ransomware classification. See Table 5.19 below for these differences.

Project/Research Paper	Techniques For Analysis	Feature Selection and Model Construction Methods	Machine Learning Algorithms
BSc (7.4)	CopperDroid[43]	System_call bit/frequency vector n-grams (1-gram, 2-gram and 3-gram)	Hierarchical unsupervised machine learning method
MSc (this document)	Static and dynamic analysis[41]	Operation/ category of operation bit/ frequency vector n-grams (1-gram, 2-gram and 3-gram)	Hierarchical unsupervised machine learning method
Bayer et al.[20]	Dynamic analysis	Behavioural profiles	Hierarchical unsupervised machine learning method
Hamid et al.[38]	-	-	K-means unsupervised machine learning method
Zhang et al.[25]	Static analysis	TF-IDF n-grams (1-gram, 2-gram, 3-gram and 4-gram)	Supervised machine learning methods (Decision Tree, Random Forest, K-Nearest Neighbour, Naive Bayes, and Gradient Boosting Decision Tree)

Table 5.19: This table displays that all five projects/research papers used different techniques for analysis, feature selection and model construction methods and machine learning algorithms.

This confirms my initial conclusion that the accuracy of clustering-based malware detection is highly subjective; it is dependent on many factors such as the analysis to generate the features, the method that is used to select the features and the machine learning algorithm.

However Mehnaz et al.[27] said that there are several mechanisms for malware detection but a few for Ransomware. They claim that analysing Ransomware for file usage patterns or behaviours results in delayed detections and “monitoring only the process activities ... or file changes is not sufficient for effective detection”. This is because it yields high false positive and false negative results. This suggests that a combination of features should be selected to achieve the most accurate results.

Therefore, more research is needed to find out whether this difference (and/or other reasons) affect the accuracy of clustering malware. For example, researches should conduct a validation study where they use the Ramilli dataset that I used ([41]) in this project using the feature selection, model construction and machine learning methods that Bayer et al.[20], Hamid et al.[38] or Zhang et al.[25] used and see if they produce a similar result (high precision and recall as in Bayer et al.[20] or high accuracy results as in Zhang et al.[25]). This will then help to conclude whether clustering improves malware detection or not.

Li et al.[40] found that the make-up of the ground-truth data (method of selection i.e. distribution sizes of the malware families) from prior evaluations “biases their results toward high accuracy” specifically increasing the likelihood of good precision and recall. They used the sequences of system calls and API calls of malware and performed single-linkage hierarchical clustering, precision, recall and F-measure. They concluded and proved that as the plagiarism detectors and BCHKK-algo (used by Bayer et al.[20]) performed well on the clustering (greater precision, worse recall and similar f-measure) with ground truth labels inferred by antivirus tools and was better with the BCHKK-data dataset (used by Bayer et al.[20]) compared to the VXH-data dataset (malware instances see[40]). This may be due to way the systems gathered the malware traces because they can vary in the length and composition of API sequences or due to the differences in the presence and frequencies of certain activities in each dataset. The main difference noted in the datasets were the distribution of cluster sizes as the BCHKK-data dataset is highly biased with 2 large clusters covering more than half of the malware instances.

Finally, the running times of the experiments need to be considered. When these experiments were run on a local machine (CPU), unreasonable running times were produced (the shortest time to perform one experiment (of the twelve) was 2 days and the longest time was over 4 weeks without a break and the vectors were still being constructed). This long run time could be due to the large number of system call dimensions in the feature vectors (e.g. 2025 system call dimensions for Ransomware Di-grams Full Representation) and the different heights obtained for the dendrogram. This is because the feature space increases, the number of dimensions in the feature vectors are larger and the distance between the feature vectors are larger and so there are more distances and less vectors that are similar. This also meant that it was infeasible for me to compute experiments with n-grams of more than tri-grams (3-grams). However, if I had access to a GPU, the experiments would execute much quicker. I would be able to expand this project and look at larger datasets or other feature selection and vector representation methods in order to produce more accurate results. This means that the best clustering may not reflect completely what the evaluation metrics say is the best clustering. Instead, the best clustering should be based on the ground truths (evaluation metrics) and running times. This is because if it takes too long to run, people will be reluctant to use it to detect malware.

5.3 Self-Evaluation

Overall the project went well and I was able to achieve all the project aims and deliverables that I had intended to achieve. I created a program which extracted features from behavioural profiles of three different malware families and implemented clustering on the resulting data.

The objectives of my project are slightly different to the objectives stated in my project description form. Initially I was going to compare clustering results between Ransomware and Android applications (BSc Project). However, both datasets did not have similar features and are not from the same year and both malware types are independent therefore it did not make sense to compare them to each other. For example, if it was Ransomware for Android or both Ransomware and Android had similar families then it would make sense to compare them. Instead I looked at the

other malware types in the dataset and I noticed that similar features were occurring in Backdoor and Trojan. Therefore, I decided to cluster each malware type individually into families and then compare the results between them. This also meant that my title changed from “A Novel Approach To Clustering: Clustering Ransomware Behaviour As An Aid For Ransomware Detection In The Future” to “A Novel Approach To Clustering Malware Behaviour To Improve Malware Detection”. As I added two more malware families and had to run the analysis and process the results for each family, I did not have time to optimise both the Ransomware and Android programs to see if this makes a difference to the result. All other objectives (including the optional ones) in my project description form were completed.

This project enabled me to learn and develop new and existing skills such as expand my knowledge of Python (a programming language I learnt for my BSc project last year) and Latex (a report writing software which I learnt about last year but learnt more features such as captions on figures and tables, chapter headings, bibtex etc). I stuck to my project plan timeline which meant I finished my tasks on time and then I could concentrate on my report structure in August. I was not rushing or struggling to complete my work on time. As I stuck to my timings well, it meant that I could send most of my report to my supervisor for feedback instead of sending him incomplete sections.

A difficulty I encountered during my project was the running-times of my program. The feature selection, model construction and the validation parts of my program took time to compute and produced unreasonable running times. Therefore, the program was taking a long time to run one experiment (12 all together) and so the overall programming time was large. This long run time could be due to the large number of system call dimensions in the feature vectors (e.g. 2025 system call dimensions for Ransomware Di-grams Full Representation) and the different heights obtained for the dendrogram. My solution to this was to use google colabatory (use the GPU) to produce all the text files needed to compute the best clustering. However, this was also limited to a 12 hour runtime and had to be reconnected to a hosted runtime every few hours.

6

Conclusion

This chapter concludes this project by describing the main finding of this project.

6.1 Conclusion

In conclusion the results in this and other projects suggest that there is a discrepancy with the accuracy of clustering-based malware and whether clustering improves malware detection. The accuracy of clustering-based malware detection is highly subjective as it depends on many factors including the type of machine learning algorithm, the features selected, the feature selection methods, the model construction methods and evaluation metrics (as shown in my results where the different methods of feature selection and vector representation yielded different results (scores and best clustering methods) for the validation metrics and the accuracy was low and other papers using different methods where the accuracy was high). Therefore, future research should be conducted to find out all the reasons that may affect the accuracy of clustering malware and discover the best methods in terms of accuracy and a good run time for clustering malware to improve malware detection.

Bibliography

- [1] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Massachusetts Institute of Technology, 2012.
- [2] C. Chio and D. Freeman, *Machine Learning and Security*. O'Reilly Media, 2018.
- [3] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*. Springer, 2013.
- [4] M. Tomlinson, "Outperforming the human: How can machine learning help your business?." <https://pwc.blogs.com/data/2018/03/outperforming-the-human-how-can-machine-learning-help-your-business.html>, 2018.
- [5] F. Ciaia, "Meet our new head of machine learning." <https://www.pwc.co.uk/services/audit-assurance/actuarial/insights/meet-new-head-of-machine-learning.html>, 2016.
- [6] PWC, "Uk gdp growth projected to be 0.5% in the third quarter of 2018 according to pwc's "nowcasting" model." <https://www.pwc.co.uk/press-room/press-releases/UK-GDP-growth-projected-to-be-0.5-in-the-third-quarter-of-2018-according-to-PwCs-nowcasting-model.html>, 2018.
- [7] K. Savage, P. Coogan, and H. Lau, "The evolution of ransomware." http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf, 2015.
- [8] W. Zamora, "How to protect your business from ransomware." <https://blog.malwarebytes.com/101/2016/04/how-to-protect-your-business-from-ransomware/>, 2017.
- [9] D. O'Brien, "Ransomware." <https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-ransomware-2017-en.pdf>, 2017.
- [10] E. Wilding, "Virus bulletin." <https://www.virusbulletin.com/uploads/pdf/magazine/1990/199001.pdf>, 1990.
- [11] J. Zorabedian, "Anatomy of a ransomware attack: Cryptolocker, cryptowall, and how to stay safe." <https://news.sophos.com/en-us/2015/03/03/anatomy-of-a-ransomware-attack-cryptolocker-cryptowall-and-how-to-stay-safe-infographic/>, 2015.
- [12] FBI, "Incidents of ransomware on the rise." <https://www.fbi.gov/news/stories/incidents-of-ransomware-on-the-rise>, 2016.
- [13] R. Lipovsky, L. Stefanko, and G. Branisa, "The rise of android ransomware." https://www.welivesecurity.com/wp-content/uploads/2016/02/Rise_of_Android_Ransomware.pdf, 2016.
- [14] S. Joachim, "Unix: A game changer in the ransomware landscape?." <https://blog.trendmicro.com/trendlabs-security-intelligence/unix-a-game-changer-in-the-ransomware-landscape/>, 2017.
- [15] O. Ziv, "0.2 btc strikes back, now attacking mysql databases." <https://www.guardicore.com/2017/02/0-2-btc-strikes-back-now-attacking-mysql-databases/>, 2017.
- [16] C. Wueest, "How my tv got infected with ransomware and what you can learn from it." <https://www.symantec.com/connect/blogs/how-my-tv-got-infected-ransomware-and-what-you-can-learn-it>, 2015.
- [17] M. Rouse, "What is backdoor (computing)?." <https://searchsecurity.techtarget.com/definition/back-door>, 2017.
- [18] MalwareBytes, "Backdoor." <https://www.malwarebytes.com/backdoor/>.
- [19] MalwareBytes, "Trojan." <https://www.malwarebytes.com/trojan/>.

- [20] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," *16th Annual Network and Distributed System Security Symposium*, vol. 9, pp. 1–18, 2009.
- [21] A. X. Zheng, *Mastering Feature Engineering*. O'Reilly, Pre-Release edition, 2016.
- [22] C. Boutsidis, M. W. Mahoney, and P. Drineas, "Unsupervised feature selection for the k-means clustering problem," *Proceedings of the 22nd International Conference on Neural Information Processing Systems (NIPS'09)*, pp. 153–161, 2009.
- [23] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "A quantitative study of accuracy in system call-based malware detection," *Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA 2012)*, pp. 122–132, 2012.
- [24] D. Mutz, F. Valeur, C. Kruegel, and G. Vigna, "Anomalous system call detection," *ACM Transactions on Information and System Security (TISSEC)*, pp. 61–93, 2006.
- [25] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, and F. Martinelli, "Classification of ransomware families with machine learning based on n-gram of opcodes," *Future Generation Computer Systems*, vol. 90, pp. 211–221, 2019.
- [26] Y.-L. Wan, J.-C. Chang, R.-J. Chen, and W. Shiuh-Jeng, "Feature-selection-based ransomware detection with machine learning of data," *3rd International Conference on Computer and Communication Systems*, pp. 85–88, 2018.
- [27] S. Mehnaz, A. Mudgerikar, and E. Bertino, "Rwguard: A real-time detection system against cryptographic ransomware," *International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 114–136, 2018.
- [28] Z. A. Genc, G. Lenzini, and P. Y. Ryan, "No random, no ransom: A key to stop cryptographic ransomware," *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 234–255, 2018.
- [29] S. Raschka, "About feature scaling and normalization." http://sebastianraschka.com/Articles/2014_about_feature_scaling.html, 2014.
- [30] S. Learn, "Importance of feature scaling scikit-learn 0.19.0 documentation." http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html.
- [31] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.
- [32] Y. Takeuchi, K. Sakai, and S. Fukumoto, "Detecting ransomware using support vector machines," *Proceedings of the 47th International Conference on Parallel Processing Companion*, pp. 1–6, 2018.
- [33] S. S. im Walde, "Experiments on the automatic induction of german semantic verb classes," *Arbeitspapiere des Instituts fr Maschinelle Sprachverarbeitung (AIMS) report*, pp. 1–343, 2003.
- [34] T. S. Madhulatha, "An overview of clustering methods," *IOSR Journal of Engineering*, vol. 2, no. 4, pp. 719–725, 2012.
- [35] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson, 2005.
- [36] Y. Lu, I. Cohen, X. S. Zhou, and Q. Tian, "Feature selection using principal feature analysis," *Proceedings of the Fifteenth ACM International Conference on Multimedia*, pp. 301–304, 2007.
- [37] M. Namratha, "A comprehensive overview of clustering algorithms in pattern recognition," *IOSR Journal of Computer Engineering*, vol. 4, no. 6, pp. 23–30, 2012.
- [38] I. R. A. Hamid, N. S. Khalid, N. A. Abdullah, N. H. A. Rahman, and C. C. Wen, "Android malware classification using k-means clustering algorithm," *IOP Conference Series: Materials Science and Engineering*, vol. 226, 2017.
- [39] L. Chen, Y. Chih-Yuan, P. Anindya, and R. Sahita, "Towards resilient machine learning for ransomware detection," *Computing Research Repository*, 2018.

- [40] P. Li, L. Liu, D. Gao, and M. Reiter, "On challenges in evaluating malware clustering," *Recent Advances in Intrusion Detection, Springer Berlin Heidelberg*, pp. 238–25, 2010.
- [41] M. Ramilli, "Malware training sets: a machine learning dataset for everyone." <https://marcoramilli.com/2016/12/16/malware-training-sets-a-machine-learning-dataset-for-everyone/>, 2016.
- [42] P. Trinius, C. Willems, T. Holz, and K. Rieck, "A malware instruction set for behavior-based analysis," *Proceedings of 5th GI Conference "Safety, Protection and Reliability"*, 2010.
- [43] K. Tam, S. J. Khan, A. Fattoriy, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors," *NDSS Symposium*, 2015.
- [44] U. Bayer, C. Kruegel, and E. Kirda, "Ttanalyze: A tool for analyzing malware," *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, 2006.



Appendices

7.1 Appendix A - Program

7.1.1 Installation Requirements

A Ransomware dataset was provided by Ramilli[41] (please see Description of Program and Justification of Feature Selection and Validation Methods for a description of this dataset and the rationale for using it. The JSON files were used to perform hierarchical clustering. The program consisted of feature selection, vector representation (model construction), hierarchical clustering and validation. It was written in python 2.7 with the libraries Scikit-learn, Numpy, Scipy, Pandas, Matplotlib.

I used a git repository (which was used to run on google colabatory) to store the code which only produces and downloads all the text files for running the individual experiments for one malware family at a time (please see <https://github.com/Rebecca22/Proj2> for the code that produces all the text files). **Please note:** The text files were too large to store on git hub which meant that I could not run my best cluster algorithm to find the best clustering using the text files according to the FMS, F1, ARI and SC scores. This meant only the code for the production of the text files is on github, all the rest of the program code is stored on my local machine as it is too large to store on github.

To install the various applications/ libraries:

1. Python 2.7.13 - Install the correct python version for <https://www.Python.org/downloads/release/Python-2713/> e.g. for a 64-bit operating system in windows install the Windows x86-64 MSI installer. **Note:** Make sure the "Add to path" task is ticked so the Python programs are easier to type into cmd.
2. "Scikit-learn" library (including numpy and scipy that scikit-learn depends on):
 - (a) Python has a special library installer program called "pip", install and update it to its latest version by typing in cmd: `Python -m pip install -U pip setuptools`
 - (b) The Scikit-learn website needs the "numpy" and "scipy" libraries to be installed. For windows (other versions of OS may be readily available on the pip website) as there is no version available on the pip website, special files that pip can interpret need to be installed. Download Numpy: <http://www.lfd.uci.edu/~gohlke/Pythonlibs/#numpy> ((*numpy* - 1.13.1 + *mkl* - *cp27* - *cp27m* - *win_amd64.whl*)) and Scipy: <http://www.lfd.uci.edu/~gohlke/Pythonlibs/#scipy> ((*scipy* - 0.19.1 - *cp27* - *cp27m* - *win_amd64.whl*))
 - (c) Using cmd, navigate to "downloads" folder and type `pip install numpy.file.name` and after this repeat this for scipy (as scipy depends on numpy).
 - (d) Finally type in cmd: `pip install -U scikit-learn`

3. Any other Python libraries that are not supported by pip e.g. pandas
 - (a) Go to <https://www.lfd.uci.edu/~gohlke/pythonlibs/#pandas> and download the correct version(the same version as numpy and scipy (*pandas – 0.24.2 – cp27 – cp27m – win_amd64.whl*)).
 - (b) Using cmd, navigate to the “downloads” folder.
 - (c) Type in cmd: “pip install “pandas file_name”
4. Any other Python libraries that are supported by pip e.g. matplotlib In cmd type:
 - (a) In cmd type: Python *-mpip* install *-U* pip
 - (b) In cmd type: Python *-mpip* install *-U* matplotlib

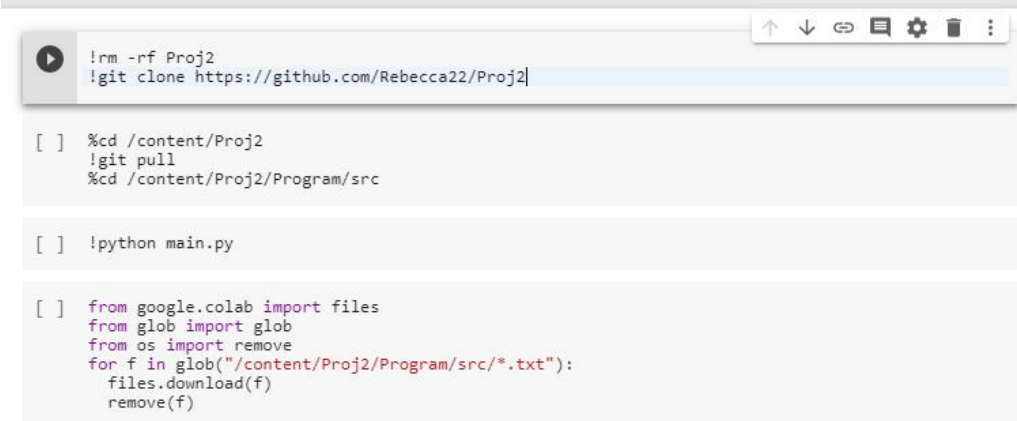
7.1.2 User Manual

Google Colabatory and Git Hub(to produce the text files)

Description of the classes in src package:

1. main.py - used to run my program by running the main method which runs experiments for each system call and vector representations.
2. HierarchicalClustering package - contains a class that standardises the frequency feature vectors (not bit vectors), calculates the Euclidean Distances of each of the feature vectors and then produces a dendrogram.
3. JsonFiles package - contains a class that gets all the folders in the specified samples folder (either Ransomware, Backdoor or Trojan), and for each family of the specific malware, gets the json files and stores the contents of the files in a list and then stores the list for each files in a family in a dictionary and returns the dictionary.
4. ProduceVector package - contains a class which produces a vector depending on if it is a bit vector or frequency vector and the type n-gram and returns it.
5. SystemCallRepresentation package - contains a class which produces a list of system calls for each file and returns it depending on the type of syscall behaviour being represented ie full representation or category.
6. Validation package - contains a class that evaluates the cuts on the dendrogram at different heights using metrics (FMS, F1, ARI and SC) by finding the best clustering for the matrix passed in.

I used google colabatory to produce all the text files that the best clustering method on my local machine will use to find the best clustering for each family.



```

!rm -rf Proj2
!git clone https://github.com/Rebecca22/Proj2

[ ] %cd /content/Proj2
!git pull
%cd /content/Proj2/Program/src

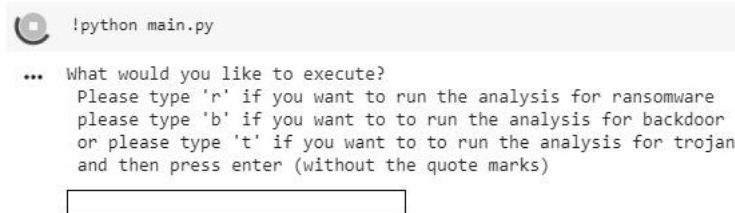
[ ] !python main.py

[ ] from google.colab import files
from glob import glob
from os import remove
for f in glob("/content/Proj2/Program/src/*.txt"):
    files.download(f)
    remove(f)
  
```

Figure 7.1: This picture displays the code written in google colabatory.

The screenshot above (Figure 7.1) shows the code for the running of my program. Please see <https://github.com/Rebecca22/Proj2> for the code that is being run. Google colab clones my repository and then runs the main method within the program. After it has run the main method, the text files produced by my program will be downloaded.

Below (Figure 7.2) is a screenshot of google colab once the main method is run:



```
!python main.py
... What would you like to execute?
Please type 'r' if you want to run the analysis for ransomware
please type 'b' if you want to to run the analysis for backdoor
or please type 't' if you want to to run the analysis for trojan
and then press enter (without the quote marks)

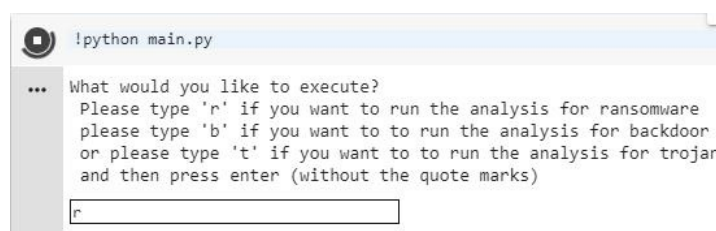
```

Figure 7.2: This picture is a screenshot of google colabatory once the main method is run.

This picture (Figure 7.2) asks the user what they would like to execute. If they enter 'r', every experiment for each type of system call and feature vector representation will be executed for the Ransomware malware family where the feature vectors will be constructed, standardised (if a frequency vector), a dendrogram will be produced and the cuts of the dendrogram at the various heights will be validated using FMS, F1, ARI and SC scores to determine the best cluster. If they enter 'b' or 'c' then the same thing will happen as described above for Ransomware but for Backdoor ('b') or Trojan ('t').

1. User enters 'r' (same thing will happen for when the user enters 'b' or 't'): Here experiments for each type of vector (Uni-gram bit vector, Uni-gram frequency vector, Di-gram bit vector, Di-gram frequency vector, Tri-gram bit vector or Tri-gram frequency vector) and for each type of behaviour Full representation or Category). In each experiment, the feature vectors will be extracted from all the json files in the malware families (1 vector for every json file), standardised (if frequency vector), the Euclidean Distance is calculated on them and a dendrogram is created of the malware samples against Euclidean Distances. Next the dendrograms are cut at different heights to obtain different clusterings and the clusterings obtained are compared to the labelled dataset from Ramilli[41] and the best clustering obtained (the method of feature selection and model construction and the cut that is the most similar as the labelled dataset) is when the Fowlkes Mallows Score, the F1-Score, the Adjusted Rand Index or the Silhouette Coefficient is the closest to 1.

User enters 'r' (Figure 7.3):



```
!python main.py
... What would you like to execute?
Please type 'r' if you want to run the analysis for ransomware
please type 'b' if you want to to run the analysis for backdoor
or please type 't' if you want to to run the analysis for trojan
and then press enter (without the quote marks)
r
```

Figure 7.3: This picture is a screenshot of google colabatory where the use enters 'r'.

The experiments for each feature vector will be run:

- (a) Firstly the program, explains the type of feature vector and system call representation that it being run e.g. Running experiment for Uni-gram bit vector with full representation (Figure 7.4 a).
- (b) Next it says that it is producing the feature vectors. Here the program extracts the system calls from the malware samples, creates the dimensions of the feature vectors

according to the Uni-gram, Di-gram or Tri-gram. The program prints compiled keys to tell the user that the dimensions of the feature vectors have been produced (Figure 7.4 b).

- (c) It then populates the feature vectors according to the malware samples and whether the vector is a bit vector (1 if the system call is present in the behavioural profile or 0 otherwise) or frequency vector (the number of system calls observed in the behavioural profile according to the Uni-gram, Di-gram or Tri-gram). The program prints the number of system call dimensions of the feature vector to the user (Figure 7.4 c).
- (d) Then a dendrogram is produced for the feature vector. Firstly if the feature vector is a frequency vector, it is standardised and then Euclidean distances are calculated and a dendrogram is constructed (Figure 7.4 d).

```

a Running experiment for Uni-gram bit vector with full representation
b Producing the feature vectors
  compiled keys
c The number of system call dimensions is: 32
d Producing a dendrogram

```

Figure 7.4: This picture is a screenshot of google colabatory illustrating the steps from a to d above.

- (e) Next the dendrogram produced is cut at different heights (the number of heights is displayed in the output to the user) to obtain different clusterings. The clusterings obtained are compared to the labelled dataset from Ramilli[41] and are validated against the FMS, F1, ARI and SC scores and a graph is produced to display the scores at the different heights (Figure 7.5).

```

Validating the clusters obtained
The number of different heights are = 5

```

Figure 7.5: This picture is a screenshot of google colabatory illustrating the step e above.

This process (steps a - e) continues for each feature vector (repeats 12 times).

Under the files tab on the left hand side of the screen, under the folder /content/ Proj2/ Program/ src, it has a list of all the text files to be produced in the format of "malware_type bit/frequency_vector file_produced n-gram_and_system_call_representation". For example the file /content/Proj2/Program/src/Ransomware b ARI uniFull Representation Bit Vector.txt (see the file circled below) means that the file contains the ARI list for Ransomware of Uni-gram with the system call representation of Full Representation and the vector representation of Bit Vector (Figure 7.6).

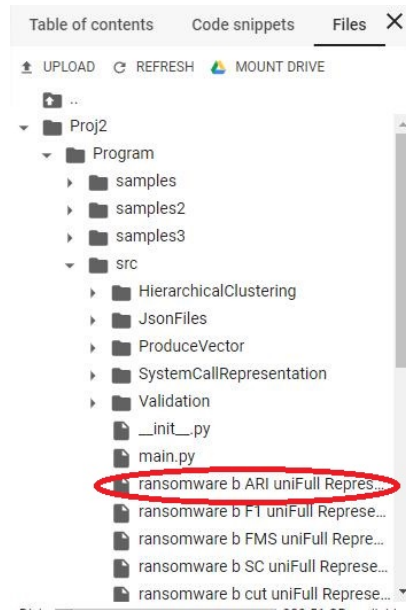


Figure 7.6: This picture is a screenshot of google colabatory illustrating all the text files produced under the files tab.

Then the program downloads all the files.

2. If the user does not enter either 'r', 'b' or 't' then an error will be displayed to the user explaining to them what they did wrong (Figure 7.7).

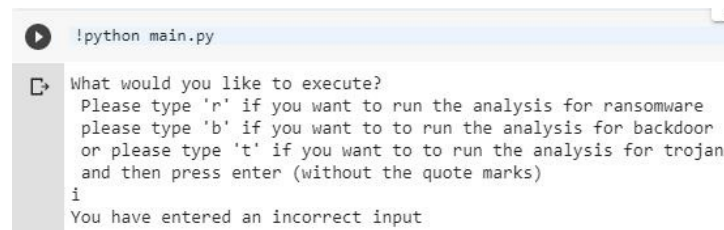


Figure 7.7: This picture is a screenshot of google colabatory illustrating the error if the user enters a wrong input.

Local Machine (Best Clustering)

1. User runs main:

For the best cluster for an individual malware family or for all the malware families, from the lists of the FMS, F1, ARI and SC scores for each height stored in text files, the program will calculate the best clustering obtained (the method of feature selection and model construction and the cut that is the most similar as the labelled dataset). This is when the Fowlkes Mallows Score, the F1-Score, the Adjusted Rand Index or the Silhouette Coefficient is the closest to 1.

User runs main (Figure 7.8):

```
What would you like to execute?
Please type 'r' if you want to run the analysis for ransomware,
please type 'b' if you want to to run the analysis for backdoor,
please type 't' if you want to to run the analysis for trojan
or please type 'all' if you want to to run the analysis for all the malware families
and then press enter (without the quote marks)
```

Figure 7.8: This picture is a screenshot of the program once the main method is run.

The best clustering from the stored FMS, F1, ARI and SC scores will be produced:

Firstly the program, gets all the heights from the cut text files (cut.txt) for each experiment and then creates and stores the heights in a list in a text file. It then asks the user what they would like to run. If the user enters 'r' then the best cluster program will only run for Ransomware, if they enter 'b' then the best cluster program will only run for Backdoor and if they enter 't' then the best cluster program will only run for Trojan. If the user enters "all" then the best cluster program will run for all the malware families (Ransomware, Backdoor and Trojan).

2. User enters 'r' (same thing will happen for when the user enters 'b' or 't') (Figure 7.9):

```
What would you like to execute?
Please type 'r' if you want to run the analysis for ransomware,
please type 'b' if you want to to run the analysis for backdoor,
please type 't' if you want to to run the analysis for trojan
or please type 'all' if you want to to run the analysis for all the malware families
and then press enter (without the quote marks)
r
```

Figure 7.9: This picture is a screenshot my program where the use enters 'r'.

- (a) The program explains that it is getting all the FMS, F1, ARI and SC scores for the different vectors. The text files which store the lists containing the FMS, F1, ARI and SC scores for each height were produced from the google colabatory program (see above) for the corresponding feature vector and the text files which store the lists containing the heights are produced in step 2 (directly above) for the corresponding feature vector (Figure 7.10).

```
a Getting all the FMS, F1, ARI and SC scores for each vector of ransomware
```

Figure 7.10: This picture is a screenshot of my program illustrating the step a above.

- (b) Next graphs are produced to display all the FMS, F1, ARI and SC scores for all the feature vectors for Ransomware at the different heights. For example for the for Uni-gram Bit Vector with Full Representation (Figure 7.11):

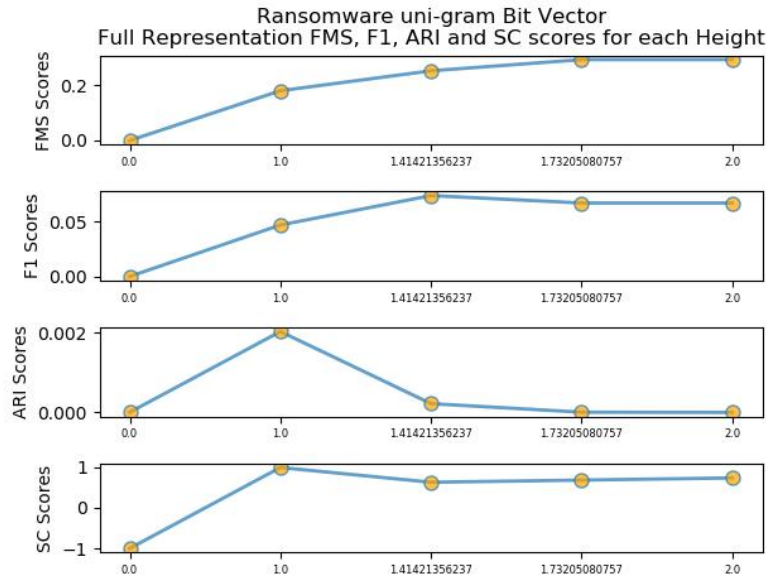


Figure 7.11: This graph displays the FMS, F1, ARI and SC scores for the Ransomware Uni-gram Bit Vector with Full Representation vector at the different heights.

- (c) The best clustering obtained for the specific feature vector is when the Fowlkes Mallows Score, the F1-Score, the Adjusted Rand Index or Silhouette Coefficient is the nearest to 1. The program calculates the best clustering for all the feature vectors using the various metrics and then displays the results (in the form of experiment, height and score) in the output. For example the first line ('Ransomware uni-gram Category Frequency Vector', 'at height', 52.593 with FMS score 0.301530665781 means that the best FMS score for Ransomware was the uni-gram with the system call representation of Category and vector representation of frequency vector at the height of 52.593 with score of 0.3) (Figure 7.12).

```
The best clusters according to the FMS, F1, ARI and SC scores are
[('Ransomware uni-gram Category Frequency Vector', 'at height', 52.59338200519851)] with FMS score 0.301530665781
[('Ransomware di-gram Full Representation Frequency Vector', 'at height', 1.4056599374977345)] with F1 score 0.147944439265
[('Ransomware uni-gram Full Representation Frequency Vector', 'at height', 0.5037587755160224)] with ARI score 0.00289659136731
[('Ransomware uni-gram Category Bit Vector', 'at height', 1.0)] with SC score 0.9996959254
```

Figure 7.12: This picture is a screenshot of my program illustrating the step c above.

- (d) Then the program prints out the order of the Ransomware experiments from best to worst for each experiment for each validation metric scores (Figure 7.13).

Ransomware Scores of each vector by their respective positions in scores from best to worst

<pre>FMS: uni-gram Category Frequency Vector di-gram Category Bit Vector di-gram Category Frequency Vector tri-gram Category Frequency Vector tri-gram Category Bit Vector uni-gram Full Representation Frequency Vector di-gram Full Representation Frequency Vector uni-gram Full Representation Bit Vector di-gram Full Representation Bit Vector tri-gram Full Representation Frequency Vector tri-gram Full Representation Bit Vector uni-gram Category Bit Vector</pre>	<pre>F1: di-gram Full Representation Frequency Vector uni-gram Full Representation Frequency Vector tri-gram Category Frequency Vector tri-gram Category Bit Vector uni-gram Category Bit Vector uni-gram Category Frequency Vector uni-gram Full Representation Bit Vector di-gram Category Frequency Vector di-gram Category Bit Vector di-gram Full Representation Bit Vector tri-gram Full Representation Frequency Vector tri-gram Full Representation Bit Vector</pre>
(a) FMS.	(b) F1.
<pre>ARI: uni-gram Full Representation Frequency Vector uni-gram Category Frequency Vector uni-gram Full Representation Bit Vector uni-gram Category Bit Vector di-gram Full Representation Frequency Vector di-gram Full Representation Bit Vector di-gram Category Bit Vector di-gram Category Frequency Vector tri-gram Full Representation Frequency Vector tri-gram Full Representation Bit Vector tri-gram Category Frequency Vector tri-gram Category Bit Vector</pre>	<pre>SC: uni-gram Category Bit Vector tri-gram Category Bit Vector di-gram Category Bit Vector tri-gram Category Frequency Vector uni-gram Category Frequency Vector di-gram Category Frequency Vector uni-gram Full Representation Bit Vector di-gram Full Representation Bit Vector tri-gram Full Representation Bit Vector uni-gram Full Representation Frequency Vector di-gram Full Representation Frequency Vector tri-gram Full Representation Frequency Vector</pre>
(c) ARI.	(d) SC.

Figure 7.13: These pictures illustrate my program printing out the order of the Ransomware experiments from best to worst for each experiment for each validation metric scores.

The same thing will happen as described above for Ransomware but for Backdoor ('b') or Trojan ('t').

3. User enters 'all' (Figure 7.14):

```
What would you like to execute?
Please type 'r' if you want to run the analysis for ransomware,
please type 'b' if you want to to run the analysis for backdoor,
please type 't' if you want to to run the analysis for trojan
or please type 'all' if you want to to run the analysis for all the malware families
and then press enter (without the quote marks)
all
```

Figure 7.14: This picture is a screenshot my program where the use enters 'all'.

- (a) The program explains that it is getting all the FMS, F1, ARI and SC scores for the different vectors. The text files which store the lists containing the FMS, F1, ARI and SC scores for each height were produced from the google colabatory program (see above) for the corresponding feature vector and the text files which store the lists containing the heights are produced in step 2 (directly above) for the corresponding feature vector (Figure 7.15).

```
a Getting all the FMS, F1, ARI and SC scores for each vector
```

Figure 7.15: This picture is a screenshot of my program illustrating the step a above.

- (b) Next graphs are produced to display all the FMS, F1, ARI and SC scores for all the feature vectors (for Ransomware, Backdoor and Trojans) at the different heights. For

example for Ransomware for the Uni-gram Bit Vector with Full Representation (Figure 7.16):

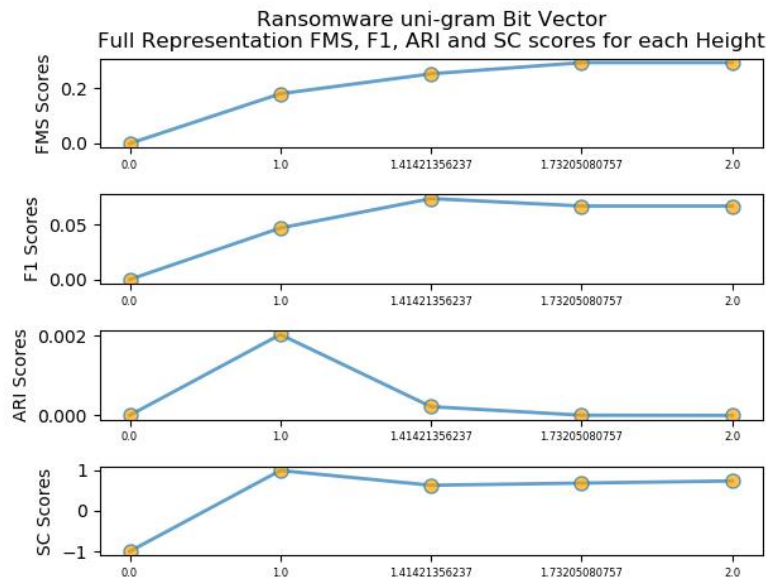


Figure 7.16: This graph displays the FMS, F1, ARI and SC scores for the Ransomware Uni-gram Bit Vector with Full Representation vector at the different heights.

- (c) The best clustering obtained for the specific feature vector is when the Fowlkes Mallows Score, the F1-Score, the Adjusted Rand Index or Silhouette Coefficient is the nearest to 1. The program calculates the best clustering for all the feature vectors using the various metrics and then displays the results (in the form of experiment, height and score) in the output. For example the first ('Backdoor tri-gram Full Representation Frequency Vector', 'at height', 34.320) with FMS score 0.457323554014 means that the best FMS score for Ransomware, Backdoor and Trojan was the tri-gram with the system call representation of Full Representation and vector representation of frequency vector at the height of 34.320 with score of 0.46) (Figure 7.17).

```
The best clusters according to the FMS, F1, ARI and SC scores are
[('Backdoor tri-gram Full Representation Frequency Vector', 'at height', 34.3198930976457)] with FMS score 0.457323554014
[('Backdoor uni-gram Full Representation Frequency Vector', 'at height', 9.7872394386919)] with F1 score 0.223066160304
[('Ransomware uni-gram Full Representation Frequency Vector', 'at height', 0.5037587755160224)] with ARI score 0.00269659136731
[('Backdoor tri-gram Full Representation Bit Vector', 'at height', 1.0), ('Backdoor di-gram Full Representation Bit Vector', 'at height', 1.0), ('Trojan uni-gram Category Bit Vector', 'at height', 1.0), ('Backdoor uni-gram Category Bit Vector', 'at height', 1.0), ('Backdoor di-gram Category Bit Vector', 'at height', 1.0)] with SC score 1.0
```

Figure 7.17: This picture is a screenshot of my program illustrating the step c above.

- (d) Then the program prints out the order of all the experiments from best to worst for each experiment for each validation metric scores (Figures 7.18, 7.19 and 7.20).

Ransomware Scores of each vector by their respective positions in scores from best to worst

FMS:

```

uni-gram Category Frequency Vector
di-gram Category Bit Vector
di-gram Category Frequency Vector
tri-gram Category Frequency Vector
tri-gram Category Bit Vector
uni-gram Full Representation Frequency Vector
di-gram Full Representation Frequency Vector
uni-gram Full Representation Bit Vector
di-gram Full Representation Bit Vector
tri-gram Full Representation Frequency Vector
tri-gram Full Representation Bit Vector
uni-gram Category Bit Vector

```

(a) FMS.

F1:

```

di-gram Full Representation Frequency Vector
uni-gram Full Representation Frequency Vector
tri-gram Category Frequency Vector
tri-gram Category Bit Vector
uni-gram Category Bit Vector
uni-gram Category Frequency Vector
uni-gram Full Representation Bit Vector
di-gram Category Frequency Vector
di-gram Category Bit Vector
di-gram Full Representation Bit Vector
tri-gram Full Representation Frequency Vector
tri-gram Full Representation Bit Vector

```

(b) F1.

ARI:

```

uni-gram Full Representation Frequency Vector
uni-gram Category Frequency Vector
uni-gram Full Representation Bit Vector
uni-gram Category Bit Vector
di-gram Full Representation Frequency Vector
di-gram Full Representation Bit Vector
di-gram Category Bit Vector
di-gram Category Frequency Vector
tri-gram Full Representation Frequency Vector
tri-gram Full Representation Bit Vector
tri-gram Category Frequency Vector
tri-gram Category Bit Vector

```

(c) ARI.

SC:

```

uni-gram Category Bit Vector
tri-gram Category Bit Vector
di-gram Category Bit Vector
tri-gram Category Frequency Vector
uni-gram Category Frequency Vector
di-gram Category Frequency Vector
uni-gram Full Representation Bit Vector
di-gram Full Representation Bit Vector
tri-gram Full Representation Bit Vector
uni-gram Full Representation Frequency Vector
di-gram Full Representation Frequency Vector
tri-gram Full Representation Frequency Vector

```

(d) SC.

Figure 7.18: These pictures illustrate my program printing out the order of the Ransomware experiments from best to worst for each experiment for each validation metric scores.

Backdoor Scores of each vector by their respective positions in scores from best to worst

FMS:

```

tri-gram Full Representation Frequency Vector
di-gram Full Representation Frequency Vector
tri-gram Category Frequency Vector
di-gram Category Frequency Vector
uni-gram Category Frequency Vector
tri-gram Category Bit Vector
uni-gram Full Representation Frequency Vector
uni-gram Category Bit Vector
uni-gram Full Representation Bit Vector
tri-gram Full Representation Bit Vector
di-gram Category Bit Vector
di-gram Full Representation Bit Vector

```

(a) FMS.

F1:

```

uni-gram Full Representation Frequency Vector
uni-gram Full Representation Bit Vector
di-gram Full Representation Frequency Vector
tri-gram Full Representation Frequency Vector
di-gram Category Bit Vector
tri-gram Full Representation Bit Vector
di-gram Category Frequency Vector
uni-gram Category Frequency Vector
tri-gram Category Frequency Vector
uni-gram Category Bit Vector
tri-gram Category Bit Vector
di-gram Full Representation Bit Vector

```

(b) F1.

ARI:

```

di-gram Full Representation Bit Vector
di-gram Full Representation Frequency Vector
uni-gram Full Representation Bit Vector
uni-gram Full Representation Frequency Vector
tri-gram Full Representation Bit Vector
tri-gram Full Representation Frequency Vector
uni-gram Category Frequency Vector
di-gram Category Bit Vector
di-gram Category Frequency Vector
uni-gram Category Bit Vector
tri-gram Category Bit Vector
tri-gram Category Frequency Vector

```

(c) ARI.

SC:

```

uni-gram Full Representation Bit Vector
uni-gram Category Bit Vector
di-gram Category Bit Vector
di-gram Full Representation Bit Vector
tri-gram Full Representation Bit Vector
tri-gram Category Bit Vector
tri-gram Full Representation Frequency Vector
di-gram Full Representation Frequency Vector
uni-gram Full Representation Frequency Vector
tri-gram Category Frequency Vector
di-gram Category Frequency Vector
uni-gram Category Frequency Vector

```

(d) SC.

Figure 7.19: These pictures illustrate my program printing out the order of the Backdoor experiments from best to worst for each experiment for each validation metric scores.

Trojan Scores of each vector by their respective positions in scores from best to worst

<p>FMS:</p> <ul style="list-style-type: none"> tri-gram Category Frequency Vector tri-gram Category Bit Vector di-gram Category Frequency Vector uni-gram Category Frequency Vector tri-gram Full Representation Frequency Vector tri-gram Full Representation Bit Vector di-gram Full Representation Frequency Vector uni-gram Full Representation Bit Vector di-gram Full Representation Bit Vector uni-gram Full Representation Frequency Vector di-gram Category Bit Vector uni-gram Category Bit Vector 	<p>F1:</p> <ul style="list-style-type: none"> tri-gram Category Frequency Vector di-gram Category Frequency Vector uni-gram Category Frequency Vector tri-gram Category Bit Vector di-gram Category Bit Vector tri-gram Full Representation Frequency Vector uni-gram Full Representation Frequency Vector di-gram Full Representation Frequency Vector uni-gram Full Representation Bit Vector di-gram Full Representation Bit Vector tri-gram Full Representation Bit Vector uni-gram Category Bit Vector
--	---

(a) FMS.

(b) F1.

<p>ARI:</p> <ul style="list-style-type: none"> uni-gram Full Representation Frequency Vector uni-gram Category Bit Vector di-gram Category Frequency Vector uni-gram Category Frequency Vector di-gram Full Representation Frequency Vector tri-gram Category Frequency Vector tri-gram Full Representation Frequency Vector uni-gram Full Representation Bit Vector di-gram Category Bit Vector tri-gram Full Representation Bit Vector di-gram Full Representation Bit Vector tri-gram Category Bit Vector 	<p>SC:</p> <ul style="list-style-type: none"> uni-gram Category Bit Vector di-gram Category Bit Vector tri-gram Category Bit Vector uni-gram Full Representation Bit Vector tri-gram Category Frequency Vector uni-gram Category Frequency Vector di-gram Category Frequency Vector di-gram Full Representation Bit Vector tri-gram Full Representation Bit Vector tri-gram Full Representation Frequency Vector uni-gram Full Representation Frequency Vector di-gram Full Representation Frequency Vector
--	---

(c) ARI.

(d) SC.

Figure 7.20: These pictures illustrate my program printing out the order of the Trojan experiments from best to worst for each validation metric scores.

4. If the user does not enter either 'r', 'b', 't' or 'all' then an error will be displayed to the user explaining to them what they did wrong (Figure 7.21).

You have entered an incorrect input

Figure 7.21: This picture is a screenshot of my program illustrating the error if the user enters a wrong input.

7.2 Appendix B - Text Files Of Results

The results for each experiment across the three different types of malware are stored in text files located at <https://www.dropbox.com/sh/snlwhqjzh8bwc2o/AAAnh9BoAfUeKo4Vax9aBKQ7a?dl=0>.

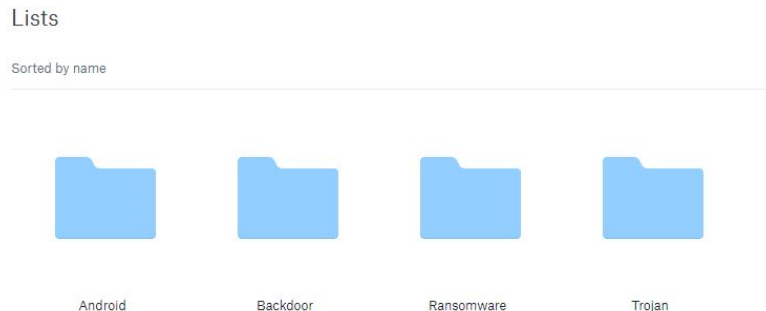


Figure 7.22: This picture is a screenshot of the start page when clicking on the url link above.



Figure 7.23: This picture is a screenshot displaying all the text files that one can click on to view the results.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_full_representation_frequency_vector](#). The screenshot below (7.24) displays a screenshot of the webpage.



Figure 7.24: This picture is a screenshot displaying all the text files that one can click on to view the results.

2. Uni-gram with Category

There were 8 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Unigram_with_category](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

3. Di-gram with Full Representation

There were 2025 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Digram_with_full_representation`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), heights of the dendrogram (`heights.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Digram_with_full_representation_bit_vector`. Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), heights of the dendrogram (`heights.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Digram_with_full_representation_frequency_vector`. Please see the screenshot (7.24) which displays a screenshot of the webpage.

4. Di-gram with Category

There were 64 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Digram_with_category`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), heights of the dendrogram (`heights.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Digram_with_category_bit_vector`. Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), heights of the dendrogram (`heights.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Digram_with_category_frequency_vector`. Please see the screenshot (7.24) which displays a screenshot of the webpage.

5. Tri-gram with Full Representation

There were 91125 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Trigram_with_full_representation`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), heights of the dendrogram (`heights.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Trigram_with_full_representation_bit_vector`. Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), heights of the dendrogram (`heights.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Trigram_with_full_representation_frequency_vector`. Please see the screenshot (7.24) which displays a screenshot of the webpage.

6. Tri-gram with Category

There were 512 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Trigram_with_category`.

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

7.2.2 Backdoor

The true assignments are attached in this text file: [true_assignments](#)

1. Uni-gram with Full Representation

There were 7 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Unigram_with_full_representation](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_full_representation_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_full_representation_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

2. Uni-gram with Category

There were 3 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Unigram_with_category](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

3. Di-gram with Full Representation

There were 49 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Digram_with_full_representation](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_full_representation_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_full_representation_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

4. Di-gram with Category

There were 9 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Digram_with_category](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

5. Tri-gram with Full Representation

There were 343 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Trigram_with_full_representation](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_full_representation_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_full_representation_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

6. Tri-gram with Category

There were 27 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Trigram_with_category](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Tri-gram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

7.2.3 Trojan

The true assignments are attached in this text file: [true_assignments](#).

1. Uni-gram with Full Representation

There were 27 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Unigram_with_full_representation](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_full_representation_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_full_representation_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

2. Uni-gram with Category

There were 8 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Unigram_with_category](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Unigram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

3. Di-gram with Full Representation

There were 729 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Digram_with_full_representation](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_full_representation_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_full_representation_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

4. Di-gram with Category

There were 64 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Digram_with_category](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Digram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

5. Tri-gram with Full Representation

There were 19683 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Trigram_with_full_representation](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_full_representation_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_full_representation_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

6. Tri-gram with Category

There were 512 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Trigram_with_category](#).

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link [Trigram_with_category_bit_vector](#). Please see the screenshot (7.23) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), heights of the dendrogram (heights.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list

(SC.txt) can be found by clicking on the relevant text files on the website link [Tri-gram_with_category_frequency_vector](#). Please see the screenshot (7.24) which displays a screenshot of the webpage.

7.3 Appendix C - Euclidean Distance Results for Ransomware, Backdoor and Trojans

The tables below show collated results of the Euclidean Distances across the three different types of malware.

1. Full Representation

(a) Bit Vector

	Vector	0	1	1-2	2-3	3-4	Largest Distance
Ransomware	Uni-gram	10646	58	21	1	0	2
	Di-gram	10611	66	36	11	1	3.162
	Tri-gram	10588	77	38	19	4	3.606
Backdoor	Uni-gram	4995	4	3	0	0	1.414
	Di-gram	4996	6	0	0	0	1.0
	Tri-gram	4998	4	0	0	0	1.0
Trojan	Uni-gram	5255	29	13	0	0	1.732
	Di-gram	523	49	9	6	0	2.828
	Tri-gram	5238	36	13	8	2	3.464

Table 7.1: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for each malware for the System Call Representation of Full Representation and Vector Representaion of a Bit Vector.

(b) Frequency Vector

	Vector	0	0-5	5-10	10-20	20-50	50-100	100-150	150-200	200-300	Largest Distance
Ransomware	Uni-gram	10524	117	38	24	16	6	1	0	0	103.577
	Di-gram	10531	68	29	21	39	26	10	2	0	174.894
	Tri-gram	10522	51	20	25	33	44	21	7	3	295.971
Backdoor	Uni-gram	4986	10	2	4	0	0	0	0	0	14.807
	Di-gram	4991	6	0	2	3	0	0	0	0	31.648
	Tri-gram	4996	2	0	0	3	0	0	0	0	34.320
Trojan	Uni-gram	518	400	15	10	10	4	0	0	0	78.318
	Di-gram	5166	76	11	13	21	9	0	1	0	172.931
	Tri-gram	5187	50	11	6	16	19	5	2	1	220.511

Table 7.2: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for each malware for the System Call Representation of Full Representation and Vector Representaion of a Frequency Vector.

2. Category

(a) Bit Vector

	Vector	0	1	1-2	Largest Distance
--	--------	---	---	-----	------------------

Ransomware	Uni-gram	9846	17	2	1.414
	Di-gram	9839	23	3	1.732
	Tri-gram	9846	17	2	1.732
Backdoor	Uni-gram	4870	2	1	1.414
	Di-gram	4870	3	0	1.0
	Tri-gram	4871	2	0	1.0
Trojan	Uni-gram	4333	11	2	1.414
	Di-gram	4332	14	0	1.0
	Tri-gram	4334	11	0	1.414

Table 7.3: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for each malware for the System Call Representation of Category and Vector Representaion of a Bit Vector.

(b) Frequency Vector

	Vector	0	0 - 5	5 - 10	10 - 60	60 - 150	Largest Distance
Ransomware	Uni-gram	9802	40	19	4	0	52.593
	Di-gram	9809	25	9	17	4	107.300
	Tri-gram	9822	19	4	12	7	143.378
Backdoor	Uni-gram	4865	4	2	2	0	16.514
	Di-gram	4867	3	0	3	0	37.335
	Tri-gram	4870	0	0	2	1	69.821
Trojan	Uni-gram	4299	39	5	3	0	54.746
	Di-gram	4306	20	8	12	0	55.678
	Tri-gram	4315	17	3	10	1	76.145

Table 7.4: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for each malware for the System Call Representation of Category and Vector Representaion of a Frequency Vector.

7.4 Appendix D - Results from Clustering Android Application Behaviour

In total, 24 experiments were run for each system call representation and feature vector representation. The results for each experiment are detailed below:

There are 49 different malware families from the CopperDroid dataset with 1230 json files in total. The table below (Table 7.5) displays the Android families and the number of json files for each family:

Name of Family	Number of Json Files
ADRD_genome_stimulated	21
AnserverBot_genome_stimulated	187
Asroot_genome_stimulated	8
BaseBridge_genome_stimulated	122
BeanBot_genome_stimulated	8
Bgserv_genome_stimulated	9
CoinPirate_genome_stimulated	1
CruseWin_genome_stimulated	2
DogWars_genome_stimulated	1

The text files containing the standardised feature vectors are stored in lists where each list is a feature vector.

The text files containing the Euclidean Distance matrices are stored in lists of each iteration in the form [vector1, vector2, dist, sample_count].

The text files containing the cuts of the dendrogram at the different heights are stored in lists where each cut is stored in a list. The number in the list denotes the cluster number that the file is assigned to and the frequency of the cluster number is denoted by the cut of the dendrogram. The number above each list in the file is the height of the dendrogram where it was cut.

The text files containing the Fowlkes Mallows Score (FMS), F1-Score (F1), Adjusted Rand Index Score (ARI) and Silhouette Scores (SC) computed at each cut for each dendrogram between the true and predictive assignments are stored in a list.

1. Uni-gram with syscalls and ioctls

There were 6 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: [Unigram_with_syscalls_and_ioctls](#)

(a) Bit vector

The feature vector ([vector.txt](#)), Euclidean Distance matrix ([matrix.txt](#)), cluster assignments ([cut.txt](#)), FMS list ([FMS.txt](#)), F1 list ([F1.txt](#)), ARI list ([ARI.txt](#)) and SC list ([SC.txt](#)) can be found by clicking on the relevant text files on the website link [Unigram_with_syscalls_and_ioctls_bit_vector](#). The screenshot below (7.25) displays a screenshot of the webpage.



Figure 7.25: This picture is a screenshot displaying all the text files that one can click on to view the results.

(b) Frequency vector

The feature vector ([vector.txt](#)), standardised feature vector ([standardisation.txt](#)), Euclidean Distance matrix ([matrix.txt](#)), cluster assignments ([cut.txt](#)), FMS list ([FMS.txt](#)), F1 list ([F1.txt](#)), ARI list ([ARI.txt](#)) and SC list ([SC.txt](#)) can be found by clicking on the relevant text files on the website link [Unigram_with_syscalls_and_ioctls_frequency_vector](#). The screenshot below (7.26) displays a screenshot of the webpage.

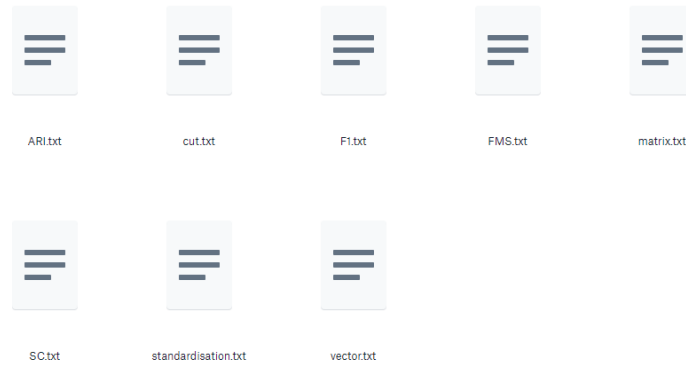


Figure 7.26: This picture is a screenshot displaying all the text files that one can click on to view the results.

2. Uni-gram with syscalls and binder semantics

There were 27 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Unigram_with_syscalls_and_binder_semantics`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link

`Unigram_with_syscalls_and_binder_semantics_bit_vector`. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Unigram_with_syscalls_and_binder_semantics_frequency_vector`. Please see the screenshot (7.26) which displays a screenshot of the webpage.

3. Uni-gram with composite behaviours and ioctls

There were 4 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Unigram_with_composite_behaviours_and_ioctls`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link

`Unigram_with_composite_behaviours_and_ioctls_bit_vector`. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Unigram_with_composite_behaviours_and_ioctls_frequency_vector`. Please see the screenshot (7.26) which displays a screenshot of the webpage.

4. Uni-gram with composite behaviours and binder semantics

There were 25 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Unigram_with_composite_behaviours_and_binder_semantics`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link

`Unigram_with_composite_behaviours_and_binder_semantics_bit_vector`. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Unigram_with_composite_behaviours_and_binder_semantics_frequency_vector. Please see the screenshot (7.26) which displays a screenshot of the webpage.

5. Di-gram with syscalls and ioctls

There were 6 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: Digram_with_syscalls_and_ioctls

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Digram_with_syscalls_and_ioctls_bit_vector. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link Digram_with_syscalls_and_ioctls_frequency_vector. Please see the screenshot (7.26) which displays a screenshot of the webpage.

6. Di-gram with syscalls and binder semantics

There were 27 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: Digram_with_syscalls_and_binder_semantics.

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Digram_with_syscalls_and_binder_semantics_bit_vector. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link Digram_with_syscalls_and_binder_semantics_frequency_vector. Please see the screenshot (7.26) which displays a screenshot of the webpage.

7. Di-gram with composite behaviours and ioctls

There were 4 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: Digram_with_composite_behaviours_and_ioctls.

(a) Bit vector

The feature vector (vector.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Digram_with_composite_behaviours_and_ioctls_bit_vector. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Distance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link Digram_with_composite_behaviours_and_ioctls_frequency_vector. Please see the screenshot (7.26) which displays a screenshot of the webpage.

8. Di-gram with composite behaviours and binder semantics

There were 25 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Digram_with_composite_behaviours_and_binder_semantics`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Digram_with_composite_behaviours_and_binder_semantics_bit_vector`. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Distance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Digram_with_composite_behaviours_and_binder_semantics_frequency_vector`. Please see the screenshot (7.26) which displays a screenshot of the webpage.

9. Tri-gram with syscalls and ioctls

There were 6 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Trigram_with_syscalls_and_ioctls`

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Tristance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Trigram_with_syscalls_and_ioctls_bit_vector`. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Tristance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Trigram_with_syscalls_and_ioctls_frequency_vector`. Please see the screenshot (7.26) which displays a screenshot of the webpage.

10. Tri-gram with syscalls and binder semantics

There were 27 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Trigram_with_syscalls_and_binder_semantics`.

(a) Bit vector

The feature vector (`vector.txt`), Euclidean Tristance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Trigram_with_syscalls_and_binder_semantics_bit_vector`. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (`vector.txt`), standardised feature vector (`standardisation.txt`), Euclidean Tristance matrix (`matrix.txt`), cluster assignments (`cut.txt`), FMS list (`FMS.txt`), F1 list (`F1.txt`), ARI list (`ARI.txt`) and SC list (`SC.txt`) can be found by clicking on the relevant text files on the website link `Trigram_with_syscalls_and_binder_semantics_frequency_vector`. Please see the screenshot (7.26) which displays a screenshot of the webpage.

11. Tri-gram with composite behaviours and ioctls

There were 4 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: `Trigram_with_composite_behaviours_and_ioctls`.

(a) Bit vector

The feature vector (vector.txt), Euclidean Tristance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Trigram_with_composite_behaviours_and_ioctls_bit_vector. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Tristance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Trigram_with_composite_behaviours_and_ioctls_frequency_vector. Please see the screenshot (7.26) which displays a screenshot of the webpage.

12. Tri-gram with composite behaviours and binder semantics

There were 25 system call dimensions in the feature vector. The feature vector dimensions (syscalls) are attached in this text file: Trigram_with_composite_behaviours_and_binder_semantics.

(a) Bit vector

The feature vector (vector.txt), Euclidean Tristance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Trigram_with_composite_behaviours_and_binder_semantics_bit_vector. Please see the screenshot (7.25) which displays a screenshot of the webpage.

(b) Frequency vector

The feature vector (vector.txt), standardised feature vector (standardisation.txt), Euclidean Tristance matrix (matrix.txt), cluster assignments (cut.txt), FMS list (FMS.txt), F1 list (F1.txt), ARI list (ARI.txt) and SC list (SC.txt) can be found by clicking on the relevant text files on the website link

Trigram_with_composite_behaviours_and_binder_semantics_frequency_vector. Please see the screenshot (7.26) which displays a screenshot of the webpage.

7.4.1 System Call Dimensions

Trends can be seen from the results. Within a specific n-gram, the more semantics being represented (e.g. ioctl calls as binder or intent transactions), the higher the number of dimensions in a vector and the larger the feature space. For example in the Uni-gram, the number of dimensions for system calls and ioctls with no semantics is 6 but for system calls and ioctl calls as binder or intent transactions (increased semantics) the number of dimensions is 27. Secondly, for each n-gram the higher the number of system calls per dimension (e.g. 1-gram = 1 system call per dimension), the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector. For example, in a Uni-gram bit vector which had the system call representation of system calls and ioctls, the number of system call dimensions is 6. For the Di-gram with the same system call representation and type of vector (bit vector which had the system call representation of system calls and ioctls) the number of system call dimensions is 25. For the Tri-gram with the same system call representation and type of vector (bit vector which had the system call representation of system calls and ioctls) the number of system call dimensions is 89. This is because the feature space increases and the number of dimensions of the feature vectors are larger.

Below is a graph (Figure 7.27) to illustrate these trends:

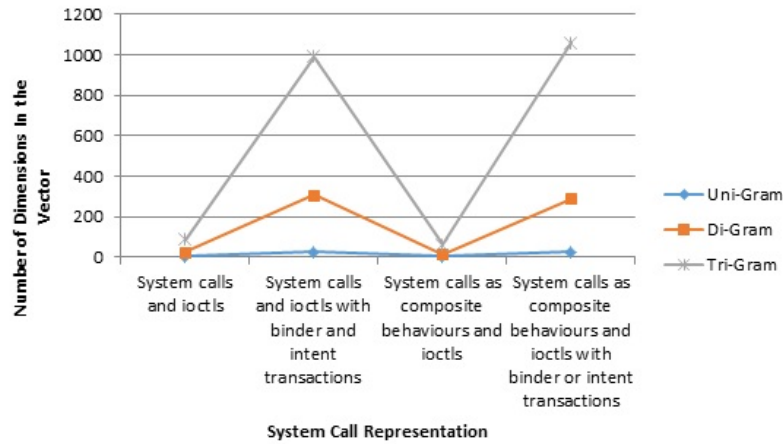


Figure 7.27: This graph illustrates that the more semantics that were being represented (higher number of system calls per dimension) within an n-gram, the higher the number of dimensions in a vector.

This graph (Figure 7.27) illustrates that the more semantics that were being represented within a n-gram, the higher the number of dimensions in a vector. It shows that the higher the number of system calls per dimension, the higher the number of dimensions in a vector. This is in comparison to a smaller number of system calls per dimension with the same system call representation and type of vector.

7.4.2 Euclidean Distance

The tables below (Tables 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, 7.12 and 7.13) display the Euclidean Distance matrices for each system call representation and type of vector. The columns are the distances being represented, the rows are the type of n-gram and the cells represent the number of json files that had the distance. A small distance between two files means that the two json files are similar to one another (dissimilar for large distances). A distance of 0 between 2 json files means that the two json file feature vectors are the same:

1. Syscalls and Ioctl Calls

(a) Bit Vector

Vector	0	1	1-2	2	2-3	3	3-4	Largest Distance
Uni-gram	1212	16	1	0	0	0	0	1.414
Di-gram	948	188	90	0	3	0	0	2.236
Tri-gram	779	107	148	52	116	8	18	3.873

Table 7.6: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Syscalls and Ioctl Calls and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0-1	1-10	10-15	15-20	20-30	30-50	50-55	Largest Distance
Uni-gram	672	534	23	1	0	0	0	0	11.657
Di-gram	463	473	284	7	1	1	0	0	21.054
Tri-gram	550	172	442	32	13	13	5	2	56.813

Table 7.7: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Syscalls and Ioctl Calls and Vector Representaion of a Frequency Vector.

2. Syscalls and Binder Semantics

(a) Bit Vector

Vector	0	1	1-2	2	2-3	3	3-4	4	4-5	5	5-6	6	6-7	Largest Distance
Uni-gram	1032	132	57	3	5	0	0	0	0	0	0	0	0	2.236
Di-gram	463	75	218	76	220	38	106	5	23	2	3	0	0	5.477
Tri-gram	379	31	99	49	247	49	207	21	97	4	31	2	13	6.928

Table 7.8: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Syscalls and Binder Semantics and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0-10	10-20	20-30	30-40	40-50	50-60	60-70	70-80	80-90	90-100	100-110	Largest Distance
Uni-gram	1222	5	0	2	0	0	0	0	0	0	0	35.285
Di-gram	842	251	79	33	15	6	0	2	1	0	0	84.746
Tri-gram	540	242	170	111	76	39	18	16	6	9	2	103.078

Table 7.9: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Syscalls and Binder Semantics and Vector Representaion of a Frequency Vector.

3. Composite Behaviours and Ioctl Calls

(a) Bit Vector

Vector	0	1	1-2	2	2-3	3	3-4	4	4-5	Largest Distance
Uni-gram	1216	13	0	0	0	0	0	0	0	1
Di-gram	981	194	53	1	0	0	0	0	0	2
Tri-gram	560	147	241	89	157	5	29	0	1	4.123

Table 7.10: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Composite Behaviours and Ioctl Calls and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0	0-1	1-2	2-3	3-4	4-5	5-10	10-15	15-30	30-35	Largest Distance
Uni-gram	776	441	6	3	2	0	0	1	0	0	11.649
Di-gram	553	440	141	54	17	6	15	3	0	0	13.542
Tri-gram	427	106	126	194	134	58	132	34	16	1	35.761

Table 7.11: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Composite Behaviours and Ioctl Calls and Vector Representaion of a Frequency Vector.

4. Composite Behaviours and Binder Semantics

(a) Bit Vector

Vector	0	1	1-2	2	2-3	3	3-4	4	4-5	5	5-6	6	6-7	Largest Distance
Uni-gram	1037	142	46	0	4	0	0	0	0	0	0	0	0	2.236
Di-gram	388	76	196	129	241	40	118	5	32	1	3	0	0	5.292
Tri-gram	339	38	109	78	285	45	168	13	97	10	39	3	5	6.245

Table 7.12: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Composite Behaviours and Binder Semantics and Vector Representaion of a Bit Vector.

(b) Frequency Vector

Vector	0-10	10-20	20-30	30-40	40-50	50-60	60-70	70-80	80-90	90-100	100-110	110-120	Largest Distance
Uni-gram	1222	5	2	1	0	0	0	0	0	0	0	0	35.285
Di-gram	817	268	92	30	15	3	3	1	0	0	0	0	71.140
Tri-gram	435	293	178	130	87	49	18	16	10	9	3	1	110.816

Table 7.13: This table displays the number of evidences which had each Euclidean Distance (column) and the largest distance for each N-gram (row) for the System Call Representation of Composite Behaviours and Binder Semantics and Vector Representaion of a Frequency Vector.

Three trends can be seen. Within a specific n-gram and system call representation, the bit vector has a higher number of json files at lower distances (especially at 0). In comparison, frequency vectors that have the same n-gram and system call representation have larger range of distances and the largest distances were bigger e.g. in the Uni-gram which had the system call representation of syscalls and ioctls and the vector representation of a bit vector there are 1212 json files that had a distance of 0. The distances represented were 0, 1, and between 1 and 2 with the largest distance at approximately 1.4. In comparison to the Uni-gram which had the system call representation of syscalls and ioctls and the vector representation of a frequency vector. 672 json files had a distance of 0 and the distances represented were 0, between 0 and 1, between 1 and 5, between 5 and 10 and between 10 and 15. The largest distance was approximately 11.7 (this also applies to Di-grams and Tri-grams).

The same trend was seen within an n-gram (Uni-gram, Di-gram, Tri-gram). Within an n-gram with the same vector representation (bit or frequency), the less semantics that is represented in the feature vector, the higher the number of json files at lower distances (especially at 0). In comparison to the same n-gram with the same feature vector representation but higher semantics are represented. In this case, more distances are represented and the largest distances were bigger. For example, in the Uni-gram which had the system call representation of syscalls and ioctls and the vector representation of a bit vector there are 1212 json files that had a distance of 0 and the

distances represented were 0, 1, and between 1 and 2 with the largest distance at approximately 1.4. This can be compared to the Uni-gram which had the system call representation of syscalls and binder semantics (ie more semantics are represented) and the vector representation of a bit vector. There were 1032 json files that had a distance of 0 and the distances represented were 0, 1, between 1 and 2, 2 and between 2 and 3 where 2.236 was the largest distance (this also applies to Di-grams and Tri-grams).

Another trend could be seen with respect to the type of n-gram being represented. With the same system call representation and type of vector representation (bit or frequency), the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the higher the number of json files at lower distances (especially at 0). When there were higher numbers of system calls per dimension in an n-gram (e.g. 2-gram or 3-gram) with the same system call representation and type of vector, more distances were represented and the largest distances were bigger. For example in the Uni-gram which had the system call representation of syscalls and ioctls and the vector representation of a bit vector there were 1212 json files that had a distance of 0 and the distances represented were 0, 1, and between 1 and 2. The largest distance was approximately 1.4. The Di-gram with the same system call representation and vector representation had 948 json files that had a distance of 0, and the distances represented were 0, 1, between 1 and 2, 2 and between 2 and 3. 2.236 was the largest distance. The Tri-gram with the same system call representation and vector representation, had 779 json files that had a distance of 0 and the distances represented were 0, 1, between 1 and 2, 2, between 2 and 3, 3, between 3 and 4. 3.873 was the largest distance.

This graph below (Figure 7.28) shows that within a Uni-gram, eventhough the system call representations were the same, the number of json files that had a distance 0 was much higher with the bit vectors than the frequency vectors.

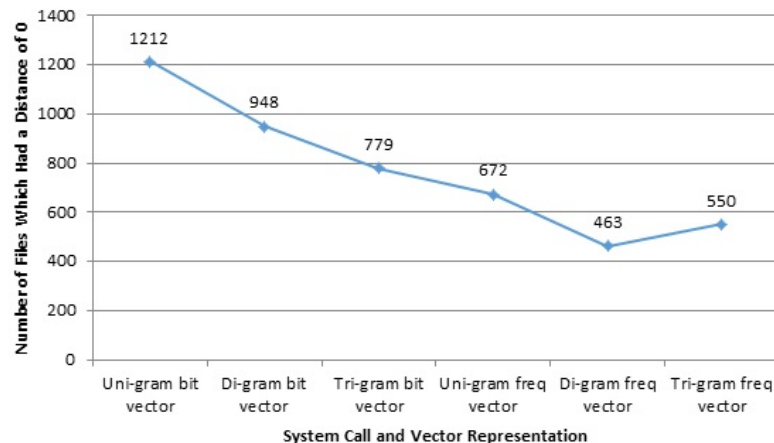


Figure 7.28: This graph illustrates that the number of json files that had a distance 0 was much higher with the bit vectors than the frequency vector.

This graph below (Figure 7.29) shows that the fewer the semantics represented in the feature vector by the system call representations, the higher the number of json files at lower distances.

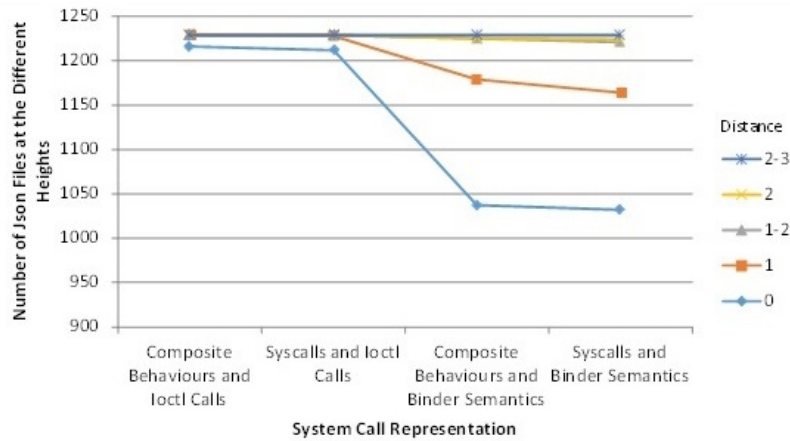


Figure 7.29: This graph illustrates that the fewer the semantics represented in the feature vector by the system call representations, the higher the number of json files at lower distances.

This graph below (Figure 7.30) shows the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the higher number of json files at lower distances (especially at 0).

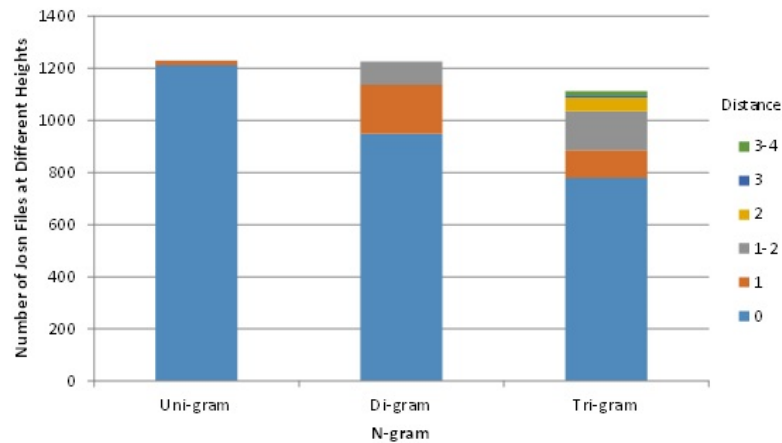


Figure 7.30: This graph illustrates that the lower the number of system calls per dimension in an n-gram (e.g. 1-gram = 1 system call per dimension), the higher number of json files at lower distances (especially at 0).

These trends can be seen because when more features are being represented, the feature space increases and the number of dimensions of the feature vectors are larger and the distance between the feature vectors are larger and so there are many more distances and less vectors that are similar.

7.4.3 Validation Metrics

The table below (Table 7.14) displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at. (Red indicates the highest FMS, F1, ARI or SC score for each n-gram):

Vector and Syscall Representation	Best FMS	Best F1	Best ARI	Best SC

Uni-gram, Syscalls and Ioctl Calls, Bit Vector	0.260 at height 1.414	0.045 at height 1.414	0.0 at height 0.0	0.998 at height 1.0
Uni-gram, Syscalls and Ioctl Calls, Frequency Vector	0.333 at height 11.657	0.111 at height 0.464	0.0004 at height 1.254	0.836 at height 5.219
Uni-gram, Syscalls and Binder Semantics, Bit Vector	0.333 at height 2.236	0.037 at height 1.414	0.003 at height 1.732	0.915 at height 1.0
Uni-gram, Syscalls and Binder Semantics, Frequency Vector	0.334 at height 35.2851	0.004 at height 2.190	0.038 at height 3.245	0.838 at height 35.2851
Uni-gram, Composite Behaviours and Ioctl Calls, Bit Vector	0.150 at height 1.0	0.035 at height 1.0	0.0004 at height 1.0	1.0 at height 1.0
Uni-gram, Composite Behaviours and Ioctl Calls, Frequency Vector	0.333 at height 11.6491	0.132 at height 0.412	0.002 at height 0.353	0.872 at height 11.6491
Uni-gram, Composite Behaviours and Binder Semantics, Bit Vector	0.333 at height 2.236	0.045 at height 1.0	0.003 at height 1.732	0.918 at height 1.0
Uni-gram, Composite Behaviours and Binder Semantics, Frequency Vector	0.334 at height 35.284	0.037 at height 3.001	0.003 at height 2.138	0.845 at height 35.284
Di-gram, Syscalls and Ioctl Calls, Bit Vector	0.333 at height 2.236	0.070 at height 1.414	0.001 at height 2.0	0.868 at height 1.0
Di-gram, Syscalls and Ioctl Calls, Frequency Vector	0.335 at height 21.054	0.074 at height 1.260	0.002 at height 0.501	0.831 at height 21.054
Di-gram, Syscalls and Binder Semantics, Bit Vector	0.334 at height 5.4772	0.039 at height 1.732	0.006 at height 3.6061	0.475 at height 1.0
Di-gram, Syscalls and Binder Semantics, Frequency Vector	0.334 at height 84.746	0.021 at height 37.227	0.001 at height 32.030	0.761 at height 84.746
Di-gram, Composite Behaviours and Ioctl Calls, Bit Vector	0.335 at height 2.0	0.036 at height 1.414	0.001 at height 2.0	0.913 at height 1.0
Di-gram, Composite Behaviours and Ioctl Calls, Frequency Vector	0.335 at height 13.542	0.103 at height 1.077	0.002 at height 0.689	0.791 at height 13.542
Di-gram, Composite Behaviours and Binder Semantics, Bit Vector	0.334 at height 5.2924	0.026 at height 3.162	0.004 at height 2.646	0.395 at height 1.0
Di-gram, Composite Behaviours and Binder Semantics, Frequency Vector	0.334 at height 71.1402	0.020 at height 37.439	0.007 at height 25.237	0.727 at height 63.699
Tri-gram, Syscalls and Ioctl Calls, Bit Vector	0.335 at height 3.873	0.082 at height 2.0	0.001 at height 2.449	0.742 at height 1.0
Tri-gram, Syscalls and Ioctl Calls, Frequency Vector	0.335 at height 56.813	0.074 at height 2.842	0.002 at height 0.555	0.838 at height 56.813
Tri-gram, Syscalls and Binder Semantics, Bit Vector	0.334 at height 6.928	0.049 at height 2.449	0.003 at height 3.162	0.389 at height 1.0

Tri-gram, Syscalls and Binder Semantics, Frequency Vector	0.334 at height 103.078	0.022 at height 63.337	0.005 at height 38.194	0.657 at height 103.078
Tri-gram, Composite Behaviours and Iocnl Calls, Bit Vector	0.335 at height 4.123	0.065 at height 1.732	0.002 at height 2.449	0.570 at height 1.0
Tri-gram, Composite Behaviours and Iocnl Calls, Frequency Vector	0.335 at height 33.953	0.065 at height 2.538	0.004 at height 5.599	0.749 at height 35.761
Tri-gram, Composite Behaviours and Binder Semantics, Bit Vector	0.333 at height 6.2453	0.058 at height 2.236	0.004 at height 4.796	0.389 at height 6.2453
Tri-gram, Composite Behaviours and Binder Semantics, Frequency Vector	0.334 at height 110.816	0.022 at height 65.038	0.004 at height 65.841	0.669 at height 110.816

Table 7.14: This table displays the best Fowlkes Mallows score (FMS), F1-Score (F1), Adjusted Rand Index (ARI) and Silhouette Coefficients (SC) (nearest to 1) for each dendrogram and the cut it was obtained at.

Looking at this table (7.14), the experiment with the best clustering according to FMS is the Di-gram which had the system call representation of Composite Behaviours and Iocnl calls. The feature vector representation was a bit vector at a score of 0.335 at height 2.0. This has the highest FMS score (one that was closest to 1) when the dendrograms were cut for all system call and feature vector representations. 0.335 is not close to 1. This means that despite this being the best clustering out of all the methods of feature selection and model construction, it is not similar to the true assignments. In the Uni-gram, 2 experiments had the same value and this was the highest FMS score. This is not beneficial as 0.334 is not very close to 1 (closer to 0). So if a programmer only had access to a Uni-gram, then using FMS, they can use any of these experiments, despite them being different to each other in terms of the number of dimensions in a vector and the features abstracted from the behavioural profiles.

The experiment with the best clustering according to F1 is the Uni-gram which had the system call representation of Composite Behaviours and Iocnl Calls. The feature vector representation was a frequency vector with a score of 0.132 at height 0.412. This had the highest ARI score (one that was closest to 1) when the dendrograms were cut for all system call and feature vector representations. Again 0.132 is not close to 1, so despite this being the best clustering out of all the methods of feature selection and model construction, it is not similar to the true assignments.

The experiment with the best clustering according to ARI is the Di-gram which had the system call representation of Composite Behaviours and Binder Semantics. The feature vector representation was a frequency vector at a score 0.007 at height 25.237. This had the highest ARI score (one that was closest to 1) when the dendrograms were cut for all system call and feature vector representations. Again 0.007 is not close to 1, so despite this being the best clustering out of all the methods of feature selection and model construction, it is not similar to the true assignments.

The experiment with the best clustering according to SC is Uni-gram which had the system call representation of Composite Behaviours and Iocnl Calls. The feature vector representation was a Bit vector at a score of 1.0 at height 1.0. It had the highest SC score (one that was closest to 1) when the dendrograms were cut for all system call and feature vector representations.

In all evaluation metrics, the best clustering method that was chosen was different. With FMS it was the Di-gram which had the system call representation of Composite Behaviours and Iocnl calls. The feature vector representation was a bit vector. With F1 it was the Uni-gram which had the system call representation of Composite Behaviours and Iocnl Calls. The feature vector representation was a frequency vector. With ARI it was the Di-gram which had the system call representation of Composite Behaviours and Binder Semantics. The feature vector representation was a frequency vector. With SC it was the Uni-gram which had the system call representation

of Composite Behaviours and Ioctl calls. The feature vector representation was a bit vector. The method of feature selection (system call representation) was the same in FMS, F1 and SC. The method of model construction (feature vector representation) was the same in FMS and SC, it was the type of n-gram that was different and the scores produced were different. These experiments are completely different to each other in terms of the number of dimensions in a vector (4 for Uni-gram and 16 for Di-gram) and the features abstracted from the behavioural profiles. The type of n-gram was different and the scores produced were different.

Similarly, within an n-gram the methods of feature selection and model construction which produced the highest score were different. For example for the Uni-gram, the methods that produced the highest score in FMS and ARI it was the feature selection method of Syscalls and Binder Semantics and the model construction method of Bit Vector. FMS had the same score in Composite Behaviours and Binder Semantics frequency vectors. In comparison, both F1 and SC had the feature selection method of Composite Behaviours and Ioctl Calls but the model construction methods were different in F1 it was Frequency vector and in SC it was Bit Vector. No n-gram had the highest score at the same experiment. The highest scores for each n-gram were frequency vectors, not bit vectors with a few exceptions such as the Uni-gram vectors in SC.

In the FMS, F1 and ARI metrics, the scores produced were not high as they were closer to 0. Even though each metric calculated the best method of feature selection and model construction which produced the best clustering, these experiments were not similar to the CopperDroid dataset so these results should be taken with caution. As the scores produced were not near to 1 and the same experiment did not produce the best clustering in both FMS, F1 and ARI, if a clustering system used either one of the best clustering methods and was presented with another dataset that contained both benign and malicious behaviour, there is no guarantee that the system will group malicious behaviour together (in the same clusters) and benign behaviour together, so the user will not be able to distinguish between benign and malicious behaviour. However, in SC the best clustering was 1.0 (the highest score possible) and most scores produced were close to 1. This means that this model has well defined clusters.

The table below (Table 7.15) shows the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up. For example, for the Di-gram Bit vector with Composite Behaviours and Ioctl calls, it is the best FMS score so will yield a score of 1 whereas Uni-gram Composite Behaviours and Ioctls Bit Vector is the worst FMS score so will yield a score of 24. This table is ordered from lowest (best) to highest (worst) total score.

Vector	SC	FMS	F1	ARI	Total
Tri-gram Composite Behaviours and Ioctls Freq Vector	15	9	3	4	31
Di-gram Composite Behaviours and Ioctls Freq Vector	13	3	6	13	35
Tri-gram Syscalls and Ioctls Freq Vector	10	6	4	17	37
Uni-gram Composite Behaviours and Ioctls Freq Vector	6	1	17	14	38
Di-gram Syscalls and Ioctls Freq Vector	12	5	5	16	38
Uni-gram Syscalls and Binder Semantics Freq Vector	9	15	9	8	41
Di-gram Composite Behaviours and Ioctls Bit Vector	5	18	1	18	42
Tri-gram Syscalls and Ioctls Bit Vector	16	4	2	20	42
Uni-gram Composite Behaviours and Binder Semantics Freq Vector	8	16	8	11	43
Uni-gram Composite Behaviours and Binder Semantics Bit Vector	3	12	22	12	49
Tri-gram Composite Behaviours and Ioctls Bit Vector	20	8	7	15	50
Di-gram Syscalls and Binder Semantics Bit Vector	21	14	14	2	51
Uni-gram Syscalls and Binder Semantics Bit Vector	4	17	21	10	52
Di-gram Syscalls and Ioctls Bit Vector	7	7	19	19	52
Di-gram Composite Behaviours and Binder Semantics Freq Vector	17	24	11	1	53
Uni-gram Syscalls and Ioctls Freq Vector	11	2	18	23	54
Tri-gram Syscalls and Binder Semantics Freq Vector	19	22	13	3	57
Tri-gram Composite Behaviours and Binder Semantics Freq Vector	18	21	12	7	58
Tri-gram Composite Behaviours and Binder Semantics Bit Vector	24	10	20	5	59
Tri-gram Syscalls and Binder Semantics Bit Vector	23	11	16	9	59
Uni-gram Syscalls and Ioctls Bit Vector	2	13	23	24	62

Di-gram Composite Behaviours and Binder Semantics Bit Vector	22	20	15	6	63
Uni-gram Composite Behaviours and Ioctl's Bit Vector	1	19	24	22	66
Di-gram Syscalls and Binder Semantics Freq Vector	14	23	10	21	68

Table 7.15: This table displays the scores of each feature vector by their respective positions in the FMS, F1, AR and SC from best to worst and then the total score of these values added up.

This table (Table 7.15) shows that overall the best clustering method is Tri-gram with the system call representation of Composite Behaviours and Ioctl's and vector representation of Frequency Vector.