

Appraising Machine Learning Models For the Classification of Malicious Code via Static Analysis



Ben Eriksson

Dr Adam Gorine

Supervisor

Cyber and Computing Security (Bsc. Hons)

School of Computing and Technology

April 2020

Declaration

The following research project and the associated proposal is the sole production of myself.

Any documentation associated that is not of my origin has been correctly cited and credited to the original author(s).

Despite the investigation of potentially malicious software, this research project does not infringe upon the ethical research principals outlined in the relevant University of Gloucestershire's handbook.

Any potentially malicious code has been legally obtained from the gratuity of Industry experts, who produce such content for an equal goal of investigative learning within the community in an open-source agreement.

Extensive efforts have been made to ensure the collected data remain isolated from any other Information Technology devices that are not within scope.

Acknowledgements

I would like to use the opportunity to express my eternal gratitude towards my family and friends for their continued encouragement throughout the course of my studies at the hardest of times. Moreover, the enthusiasm expressed by Dr Adam Gorine - my supervisor for this project, has been a great source of corroboration; His expertise has proven to be invaluable for me.

The following research project has required me to take responsibility upon learning an enormous topic that is abroad of my module choices throughout my studies. The opportunity of this project has introduced me to a new perspective of Cyber Security that I wish to pursue as my career.

Abstract

This project investigates Windows PE files, their structures and how these characteristics can be used to identify malicious intent using machine learning, implementing and comparing various machine learning / AI algorithms and frameworks such as Random Forest, K-Fold and Supervised Learning to try and combat the restrictions of static analysis of code and to hopefully prove as an accurate technique that can be implemented into an Anti-Virus engine alongside traditional methods such as signature-based classification.

Table of Contents

Declaration.....	2
Acknowledgements.....	3
Abstract.....	3
Table of Figures.....	6
Table of Tables.....	8
1. Introduction.....	9
2. Problem Statement.....	9
2.1. Research Questions.....	10
2.2. Research Objectives.....	11
2.2.1. Primary.....	11
2.2.2. Secondary.....	11
3. Scope.....	11
4. Addressing Ethical Concerns.....	12
5. Literature Review.....	13
5.1. Malware Campaigns.....	13
6.2. Comparing Classification Algorithms.....	18
6.2.1. Decision Tree-based.....	18
6.2.2. Random Forest.....	19
6.3. Reviewing Proposed Artificial Intelligence Models for Malware Classification.....	22
6.4. Discussing Proposed Academic Solutions.....	24
7. Methodology.....	26
7.1. Overview.....	26

7.2. The Windows Portable Executable Structure	26
7.3. Malware Analysis	27
7.3.1. Feature Extraction.....	27
7.3.2. Visualising Malware Similarity.....	31
7.3.3. Current Automated Solutions for Malware Classification	35
7.4. Dataset Distribution	36
7.4.1. Creating a “clean” Sampleset.....	36
7.4.2. Creating a “malicious” Sampleset	38
8. Appraising the Applied Machine Learning Techniques	40
9. Results and Analysis	44
9.1. Discussing the Measurements of Results	44
9.2. Actual Results.....	46
10. Conclusions and Discussion of Future Prospects	49
11. References	50
12. Appendix A: Dataset Statistics	54

Table of Figures

Figure 1: An Illustration Of Why Signature-Based Detection Is Insufficient (Idika and Mathur., 2007)	16
Figure 2: Input, output and properties of hash functions (SANS Institute., 2003)	17
Figure 3: Flowchart illustrating the proposed framework for the identification of polymorphic Malware (Selamat et al., 2016)	18
Figure 4: Representation of a decision-tree based classification model in the context of Malware.....	19
Figure 5: Line Graph of the accuracy achieved by a random-forest-based framework (K. Raman., 2012)	20
Figure 6: A flowchart presenting as a Random Forest algorithm, a complex decision tree algorithm.....	22
Figure 7: An illustration of how the model combines both new samples and previous performance from a training set for continuous learning.....	23
Figure 8: The dramatic decrease in accuracy Vs. precision in the increase of clustering methods (Bayer et al., 2009)	24
Figure 9: Content within the sections of a Windows PE file visualised as 8-bit greyscale (Nataraj et al., 2011)	25
Figure 10: Using the 255-bit RGB channels as features for malware classification (Fu et al., 2018)	26
Figure 11: (Nataraj., et al., 2011) precision scoring of their framework.	27
Figure 12: Three Generated Hyper-Planes To Classify Two Shapes. (Ray., 2017)	27
Figure 13: The Portable Executable (PE) structure with optional parameters (Belaoued. M, et al., 2016)	29
Figure 14: Quantifying the "imported functions" of the "malware" dataset	30
Figure 15: Quantifying the "imported functions" of the "clean" dataset	30
Figure 16: Bitcoin addresses used for payment within Wannacry, stored as strings within the sample.....	31
Figure 17: Import Functions stored within a Wannacry sample. These would be extracted by the classifier.....	32
Figure 18: Comparison of the data values in respective "overfitting" and "underfitting" scenarios (Amazon Web Services., 2016).....	32

Figure 19: Calculating the cryptographic checksums of the eighteen samples in this dataset.	33
Figure 20: Domain TLD's recognised by the implemented application	34
Figure 21: Generated output displaying similarities between two samples	34
Figure 22: Verifying networking communication performed by the sample	34
Figure 23: A false-positive identification of similarity between two samples	35
Figure 24: Example of how one presented file can be self-containing much more malicious content. 35	
Figure 25: Visualisation of a sample contacting four domains	36
Figure 26: Large scale abstraction of eleven samples contacting a domain	36
Figure 27: Scaled-up abstraction of three samples out of the generated output	36
Figure 28: A suspicious domain contacted by the sample	37
Figure 29: Suspicious IT devices contacted by the sample.....	37
Figure 30: Details of the environment used for the abstraction of samples for the "clean" dataset	39
Figure 31: Operating System "Protection Rings" Diagram (Montana State University., 2005)	40
Figure 32: Visualisation of a low-level interaction Honeypot	41
Figure 33: Visualisation of a high-level interaction Honeypot	42
Figure 34: First 10 files of the clean dataset extracted and described in Jupyter using Matplotlib	43
Figure 35: A correlation matrix of the features between the two datasets used in the classifier.....	44
Figure 36: First five files of the clean dataset	44
Figure 37: First five files of the malicious dataset	45
Figure 38: The "malware" dataset being appended with a "Malware" classification for the training set.....	45
Figure 39: The "clean" dataset being appended with a "Malware" classification for the training set .	46
Figure 40: Two samples from both datasets being removed from classification to be set aside for a testing parameter	46
Figure 41: The split-up of the dataset into "Testing" and "Training" respectively	47
Figure 42: Recall label calculated from $TP / (TP + FP)$	48
Figure 43: Precision label calculated from $TP / (TP + FN)$	48
Figure 44: Equation for how Micro-Average Precision value is calculated	48

Figure 45: Equation for how Macro-Average Precision value is calculated	48
Figure 46: Report of the accuracy results of the P and R labels for a classifier using Logistic Regression.....	49
Figure 47: Mathematical equation of how the number of neighbours is calculated in K-Nearest Neighbours.....	50
Figure 48: Report of the accuracy results of the P and R labels for a classifier using K-Nearest Neighbours.....	50
Figure 49: Report of the accuracy results of the P and R labels for a classifier using Random Forest	51
Figure 50: Calculation of "leaf" importance	51
Figure 51: Equation formulating how leaves are created and split across decision trees (Ronaghan., 2018)	52
Figure 52: First 10 files of the "malware" dataset in Jupyter Notebook	57
Figure 53: First 10 files of the "clean" dataset in Jupyter Notebook	58

Table of Tables

Table 1: Outlining the Research Objectives	11
Table 2: Primary Research Objectives	12
Table 3: Secondary Research Objectives	12
Table 4: Results of Anti-Virus Engine detection after obfuscating known samples with an opaque constants technique (Moser et al., 2007)	17
Table 5: Dataset Distribution	57

1. Introduction

Whilst the idea of combining both computing resources and technical knowledge to solve human social and scientific issues isn't a new concept, the introduction of Machine Learning and the use of Artificial Intelligence (A.I) to solve problems of a grander scale is at an unprecedented level.

The applicability of A.I can be seen as implemented at all scales of suitability. Companies such as Facebook and Twitter use Artificial Intelligence to learn more about their Users at an individual level, tailoring the content that is delivered that is relevant to their interests. Or from a business perspective, deliver customized advertising that will bring a much higher rate of ad-revenue because of the user's interests.

Considering, this use of A.I is arguably trivial when comparing other implementations of the groundbreaking technology; ranging from self-driving cars to progressing the very forefront of Science with Quantum Computing and bio-medical prediction. (Chui et al., 2019).

2. Problem Statement

The following research project will demonstrate the current threats that both consumers and organizations face against the ever-evolving “cat and mouse game” between malware authors and analysts. Current mitigations to this dynamic and prolific problem are simply not sufficient in protecting Information Technology (IT) devices as demonstrated through an enormous sample of study cases such as the Wannacry ransomware campaign – crippling the devices that power the very pillars of society as we know it, such as the National Health Service.

Akin to a real-life comparison of Police forces combatting crime in communities, malware analysts are arguably “reactionary” based rather than “preventative” – only able to respond once an incident, such as an infection of a device or network has occurred. “Signature-based” identification is the first line of defence employed by malware analysts on the war against malware. These “signatures” are unique bit-for-bit fingerprints abstractions of a sample; Used amongst the malware analyst community for identification. From this attribution, preventative mechanisms can be implemented where the sample can be detected in future appearances, where the infection-rate can be measured, but more so to prevent infection of other IT devices.

However, this critical component in combatting the infection of malware relies upon an IT device already being infected, additionally relies on the time consuming, detailed analysis of the exact strain – where the sample can infect numerous devices in the meanwhile.

Because of this bit-for-bit fingerprint, a single change within the sample, such as a character change like an Internet Protocol (IP) address will generate an entirely new “signature”, where the maliciousness of this new sample will be unknown to the preventative mechanisms, despite its behaviours not changing in comparison. Rather comparable to the biological flu or cold, a very simple change in its substrate will render a vaccination ineffective. This is especially problematic for malware analysts, as authors of this malicious code can adapt their program with very little resources including time, finances, and motivations in comparison to those available to a malware analyst.

2.1. Research Questions

To provide a solution to the current limitations induced from prevention mechanisms such as “signature-based” detection a couple of questions such as the following need to be asked, to ensure the highest possible true-positive identification of malware:

Table 1: Outlining the Research Objectives

1	What attributes can be found within malicious code that can be used to identify its intent?
2	Do variants of malicious code have similar attributes and can these common characteristics be applied to the same machine-learning model for classification?
3	Can this identification be accurate enough for true-positive classification, in comparison to alternative methods such as signature-based detection that relies upon previous analysis of the same sample?
4	What signatures and characteristics of malicious samples do current Anti Virus (A.V) engines use to identify and classify variants of malicious code?
5	Can any machine-learning model be used to combat new substrates of malicious code from procreating and propagating before infection of any IT devices without the skillset of a malware analyst?

2.2. Research Objectives

2.2.1. Primary

Table 2: Primary Research Objectives

1	To investigate the characteristics of malicious code and how these attributes can be used by a Machine Learning model for identification.
2	To assess pre-existing Machine Learning models and malware identification techniques and understand their effectiveness in identifying malicious code.
3	To evaluate the performance metrics in terms of reliability and accuracy of a variety of developed Machine Learning models and their performance in detecting malicious heuristics of code, identifying and preventing further infection.

2.2.2. Secondary

Table 3: Secondary Research Objectives

1	Can this solution be combined with pre-existing techniques such as “signature-based” analysis to prevent an IT device from being infected?
2	Perhaps the proposed solution can be combined with the method of malware analysis via sandboxing
3	Appraising standard industry techniques of collating real-world current malware samples for malware analysis

3. Scope

With Microsoft's Windows Operating System holding a majority **86%** market share (Netmarketshare., 2019) of the consumer-level market, this research project only investigates Malware samples that are compatible with the Microsoft Windows NT Operating System such as Windows 10 and Windows 7 pose a potentially disastrous risk to consumers data and privacy especially – who are not expected to have the expertise or resources to implement an effective disaster recovery plan such as that of an organisation, where there will be personnel solely responsible for such recovery plan.

Additionally, due to the very nature of malware, it is sensible to take a precautionary view when classifying malware. It is safer to over-classify than to under-classify. This consideration is a prevalent thought throughout the basis of this research project.

It is dangerous to assume and renders the project unsuccessful that the classification is always correct – despite the amount of training and accuracy scores, malicious files can be very easily miscategorized by the model due to the limitations faced by using static analysis (where the sample’s presented behaviour may change after execution) and lead to infection, from the off-set this possibility has been factored into the tolerance of the classification results by using the median of the statistics that we will come to discuss further.

With this in mind, the “success” of the implementation from this research project will be calculated from the comparison of its true-positive and false-positive outcomes from classification. Alternative methods have been proposed to help alleviate the issues faced by static malware analysis such as the use of sandboxing (running the samples in a protected environment to monitor the actual behaviours). However, research into the use of this is out-of-scope for this project.

In essence, submitted samples will be point-scored based upon the presence of malicious features—rather than categorising them definitively as either malicious or not. Moreover, the proposed solution is not a mechanism towards preventing infection of an IT device such as the function provided by an A.V engine, but rather a proposition of a technique that vendors of these products could integrate into their applications to supplement more traditional methods like “signature-based” detection.

Finally, the state-changes exhibited from malware when the program has executed will not be investigated and is out-of-scope for this project such as methods of persistence or network activity. However, some of these features exhibited are discussed later as potential aspirations for the research project.

4. Addressing Ethical Concerns

The destructive capabilities of malware are of grave concern to the integrity of this research project. Equalising malware to a biological virus which has a variety of means to replicate itself and infect other hosts, malware to has various means and efforts to infect other devices.

Although a means of infection is not by a characteristic of being airborne like a traditional virus, malware, depending upon its variant, can infect other IT devices over a network - especially in a Local Area Network (LAN) environment. With this in mind, the environment upon which the malware is stored, processed and modelled upon is completely isolated from any other IT device in a “sandbox” environment, where the assumption is that the environment will eventually be rendered unusable/infected from a sample, but has been designed so that no progress is lost – nor will such infection result in interfering with other IT devices outside of the specified scope.

Although published datasets for malware analysis with machine learning is derived from genuine malware samples, they are often un-weaponized, meaning the malicious characteristics to the sample have been removed (such as the encryption stage of ransomware).

Because of this, these datasets are only applicable for the development of machine learning models or the learning of analysis techniques, however, still contains the characteristics of the variant it is produced from - making it a perfect example for entry-level malware analysts to investigate without a high level of detrimental risk to IT devices.

5. Literature Review

5.1. Malware Campaigns

Malware analysis is a complex and ever-changing topic, containing many features to train a machine learning model upon. As (Schultz et al., 2000) proposes, “Eight to ten malicious programs are created every day, and most cannot be accurately detected until signatures have been generated for them” indicating a huge necessity for real-time analysis to be made to prevent infection on host IT devices.

This is supported by Symantec, an industry revolutionary in commercial A.V engines, who report of “[an increase] in the first six months of 2017, Symantec blocked just over 319,000 ransomware infections” (Symantec., 2017) evidently, the infection of malware across IT devices -especially ransomware - is on the dramatic increase. Case studies such as WannaCry “with infections recorded across 150 countries globally” (Nominet., 2017) lightly demonstrate the global and non-target-specific gripe the threat that malware poses to both consumers and organisations.

Alas, there are very noticeable constraints of statistical analysis of malware. One simple but fatal example of this is code obfuscation. As (Moser., Kruegel and Kirda., 2007) suggests,

“The values of the arrays [...] are crafted such that after [a] for loop, all bits of the first group have the correct, final value, while those of the second group depend on the random input”.

Values such as integers for important functions can be populated based upon random number generation after execution. Simply, the state in which the malware initially presents itself April end up acting differently upon execution. This is a very frequent technique in which sophisticated malware Authors use to bypass A.V engines. Due to the limitations of static analysis – namely the fact that the file is inspected without any execution, certain patterns and behaviours are missed such as a sample piece of ransomware “calling home” to a botnet and/or command and control (C2C) server for instruction.

As Sophos, an industry leader in Cybersecurity and IT infrastructure protection informs, malware variants such as ransomware uses external IT devices – these C2C servers to retrieve encryption keys which are fundamental to the software’s purpose. Sophos also states that ransomware “normally uses the standard port 80 and HTTP or port 443 and HTTPS protocols” (Ransomware: How an attack works - Sophos Community., 2020).

These common protocols are fundamental in day-to-day interaction between user and IT device. For example, the HTTP protocol on the standardised port of 80 is seen as normal web traffic, so any communication through this port is often un-checked by protection mechanisms such as firewalls within an organisation. This ransomware is a great example of how the analysis of these behaviours is very important in classification.

The sorts of behaviours expected from ransomware such as encryption are missed because no cryptographic function takes place until there is an execution of the file, where this is not performed during static analysis and is only detectable in either dynamic analysis (real-time execution and inspection of the file) or using static analysis techniques and tools to detect the possible feature of encryption – such as the presence of a cryptographic library. The presence of libraries such as this is not entirely indicative of the intent of a sample as many legitimate applications employ libraries utilising cryptography.

To highlight the impact of this, a file that uses code obfuscation may be classed as non-malignant. Software authors use code obfuscation to protect their work from pirating and so forth. However, upon execution, if the file were sent through the classification model again in its executed state, would henceforth be classed as malignant due to the features and behaviours now presented. This is problematic because this could create false positives within the training set, or at minimum lead to suspicious files being completely missed.

Another common - albeit sophisticated - approach to avoiding AV signature-based detection is the use of mutation, known as polymorphic code. As (Selamat., Mohd Ali and Abu Othman., 2016) proposes, malware can procreate with similar features and intentions, however, contain enough differences within their code to appear as a completely new file – hence their similarities to the common biological virus, a very common technique in evading A.V engines who employ “signature-based” detection mechanisms.

As Figure 1 illustrates, classification based upon a files “signature” is wholly insufficient for an accurate result.

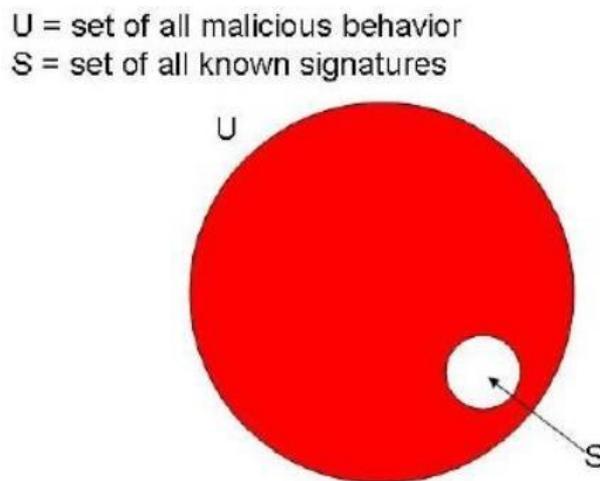


Figure 1: An Illustration Of Why Signature-Based Detection Is Insufficient (Idika and Mathur., 2007)

As we can see, the signatures known to an A.V engine is never a true representation of all malicious behaviours a file could perform.

Whilst the variation between these two factors will be different across various A.V. platforms and arguably not as drastic for some engines, the disparity is still present, nonetheless.

Malware commonly achieves a mutation from a mutagen. This mutation engine utilizes cryptography once a system is infected – a new code structure is generated ready for further infection to another host and therefore bypassing any known signature rules from an Anti-Virus engine.

A prime example of polymorphic code is ransomware. Encrypting victims’ files, ransomware is akin to the traditional highwayman. Utilising random cryptographic keys, these are individually generated mathematical operations, ultimately resulting in the withholding of information by the attacker. As this new code is randomly generated, this new mutagen will be distant enough to that of the original code base that caused the initial infection, so will not be detected from A.V engines. (Sharma and K. Sahay, 2014)

Sophisticated A.V. engines often use a combination of techniques to identify malicious code such as the previously mentioned “signature-based” technique. Utilising the hashing suite of cryptography, hashing at its fundamental is the process of calculating a fixed-length abstraction from data. Signature-based analysis at a premise, looks for specific behaviours or features of a sample and compares their relativity against already confirmed malicious samples.

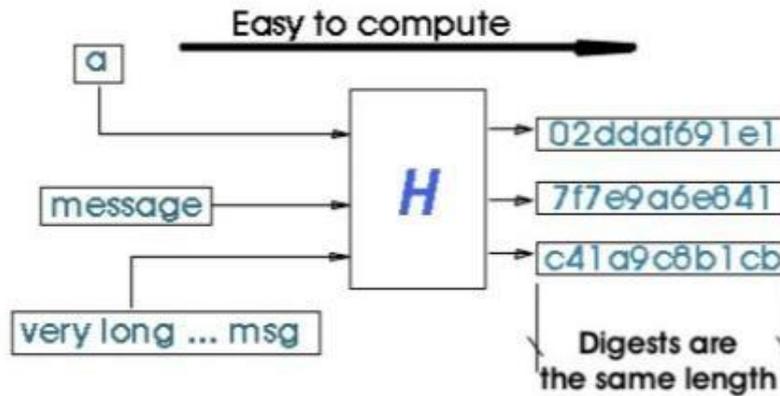


Figure 2: Input, output and properties of hash functions (SANS Institute., 2003)

However, (Bazrafshan et al., 2013) introduces the fact that although malicious code may employ techniques such as polymorphic code, modern-day A.V engines employ further techniques for classification than just signature-based. This does, however, vary between A.V. products and as a result, the same sample may remain undetected by an A.V engine. They propose that a common feature of malicious code is a random “variable name” generation.

They state that “[although] Changing the variable names may confuse human [this] has almost no effect for automated detection techniques”. On the other hand, the possibility of A.V engines and machine learning models being able to recognise patterns in obfuscated code is contested by (Moser et al., 2007) who applied a total of four custom code obfuscation techniques to malicious samples, where these samples were then run through four commercial A.V. engines, the results of this within Table 4 below.

Table 4: Results of Anti-Virus Engine detection after obfuscating known samples with an opaque constants technique (Moser et al., 2007)

	Klez.A	MyDoom.A	MyDoom.AF
McAfee		X	
Kaspersky	X	X	X
AntiVir			X
Ikarus	X	X	X

Applying their best performing algorithm, there was a 66.66% success rate in detecting malicious code, even after obfuscation, where it is theorised that this would be impossible by the works of (Bazrafshan et al., 2013).

Comparing these two-academic works, (Islam et al., 2013), proposes that this disparity in the argument is due to the necessary adaptation of focus into the “determination of features that can identify malicious behaviour as a process instead of employing a unique signature”; Involving the process of extracting other features such as external API calls and libraries – rather than analysing and decompiling code (which is the path this research project takes).

The research published by (Selamat et al., 2016) investigated the identification of polymorphic malware. Using a framework such as that of Figure 3 below.

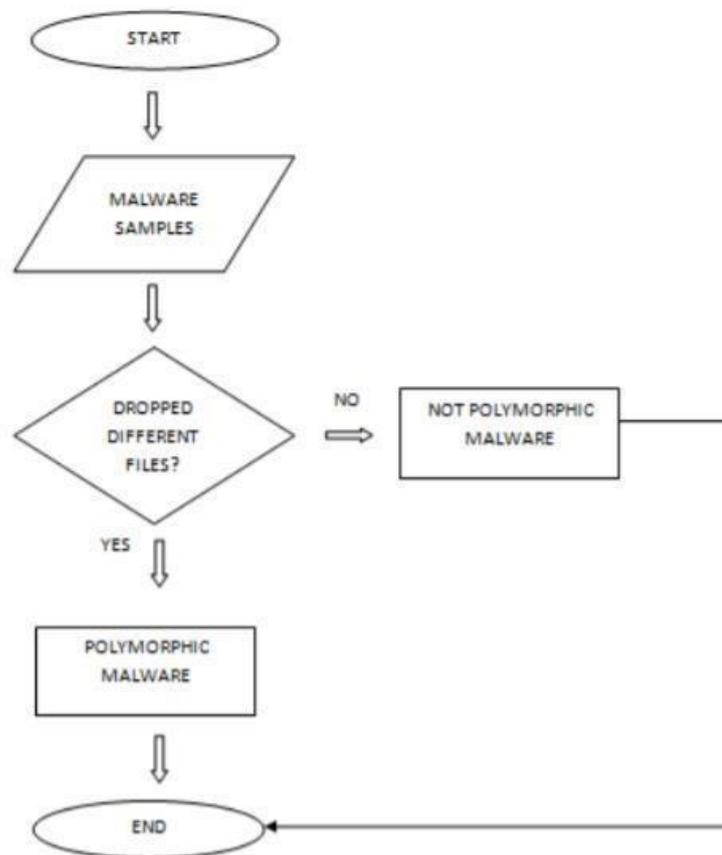


Figure 3: Flowchart illustrating the proposed framework for the identification of polymorphic Malware (Selamat et al., 2016)

6.2. Comparing Classification Algorithms

6.2.1. Decision Tree-based

Much akin to that of how humans interact with the world on a day-to-day basis, for example, evaluating factors such as the weather and temperature to decide if they should wear a jacket before leaving the house, a decision tree-based algorithm, such as logistic regression, is a supervised learning model that predicts a result based upon the presence of features, as detected by nodes placed throughout this model.

Due to the components of this type of model, it is very granular in nature, where each node can be finely tuned to detect and assign a score on the presence of programmed values. However, because of this ability to be so clearly defined, these decision trees can inadvertently become biased by the author, resulting in invalid results.

Contextualising this algorithm, nodes can be programmed to “point score” based upon the presence of a malicious attribute presented within a sample. Illustrated in Figure 4 below, each attribute will hold a certain weight - or score. The final decision on whether not a sample is classed as malicious or not will be as a result of the accumulation of the characteristics (and in turn, the number of points scored in this context) by the nodes present. The diagram below demonstrates the various stages that points are scored – similarly to decision trees.

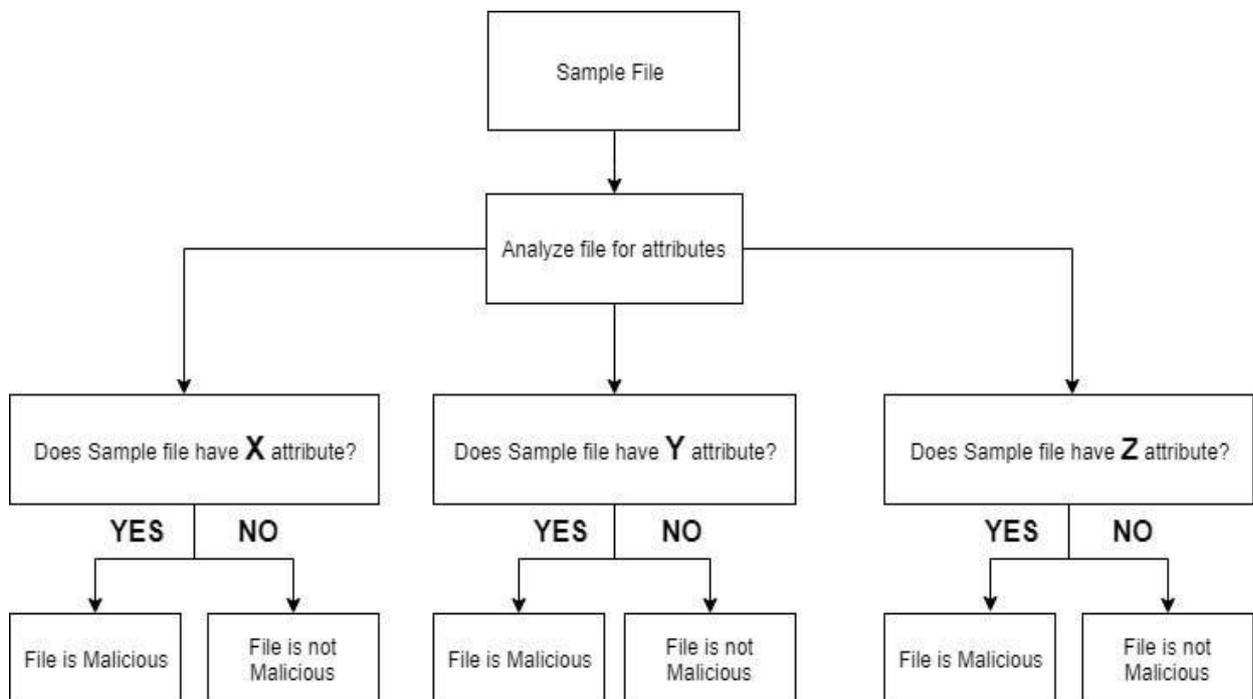


Figure 4: Representation of a decision-tree based classification model in the context of Malware

Whilst this classification algorithm has numerous benefits, such as its ability to have nodes adjusted, nodes are treated individually, so any adjustments made do not influence another node's performance. Unfortunately, it is extremely system resource-intensive, especially with large datasets according to (Anyanwu and Shiva., 2009), who also reinforce the popularity of this model due to its approachability in understanding to implement. In the instance of this research project, it is unsuitable due to the complexity of the datasets provided, but moreover due to the necessity of substantial datasets. In result, the larger a dataset - the more nodes - requiring more human intervention & calibration of how these points are scored and ultimately leading to an increased risk of overfitting.

6.2.2. Random Forest

A random-forest implementation has seen to have a high accuracy rate across a few academics such as those discussed throughout the review of literature, namely the framework created by (K. Raman., 2012). For example, as seen in Figure 5.

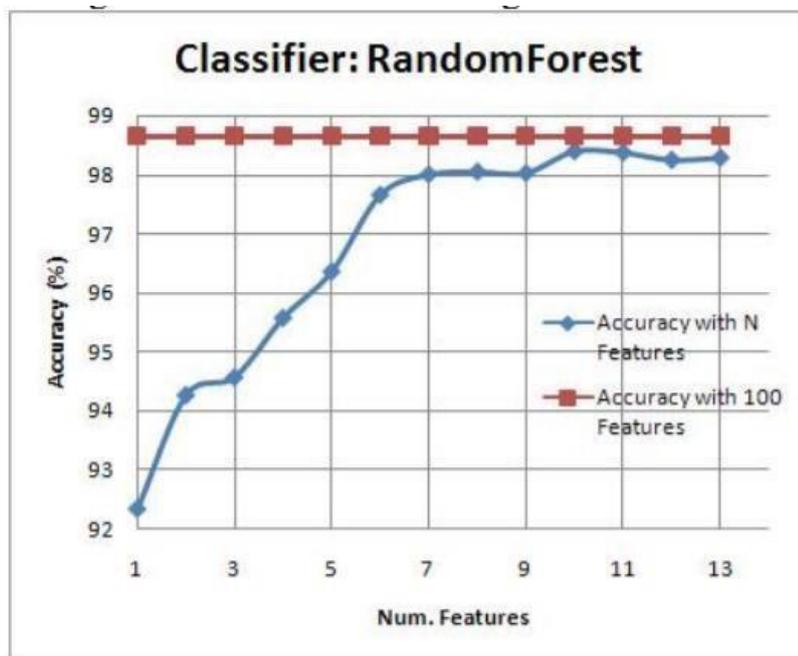


Figure 5: Line Graph of the accuracy achieved by a random-forest-based framework (K. Raman., 2012)

Less prone to overfitting, Random Forest algorithms, unlike the decision tree algorithm does not process the entire dataset in one iteration. Instead, this algorithm reduces its accuracy disparity by splitting datasets based upon provided values, where it will only test for a small subsection of features. At its essence, Random Forests are cohorts of multiple decision trees and perform in a nonsupervised manner (Breiman and Cutler., 2004). Because of this, these models are hard to program and require a quantifiable amount of epochs (iterations), more so due to the fact that the nodes contained are self-sufficient, where each iteration will select a random set of predefined features to use for classification. For example, each epoch may only use eight out of thirty possible features in a dataset.

There may only be two important features present as a parameter in this epoch, where in fact another epoch could have used a total of twelve important features that are indicators of maliciousness. The sequential epoch could be classifying a sample with a different “point scoring” system in comparison to the first epoch, and result in both higher false-negative and false-positive ratios.

Whilst this model has knowledge of previous epochs as illustrated in Figure 6 on the page below, it is argued by (Denil, Matheson and Freitas, 2014) that the causes of success in Random Forest are not well understood, nor traceable due to the mathematical randomness of how nodes within its decision trees are split for each epoch.

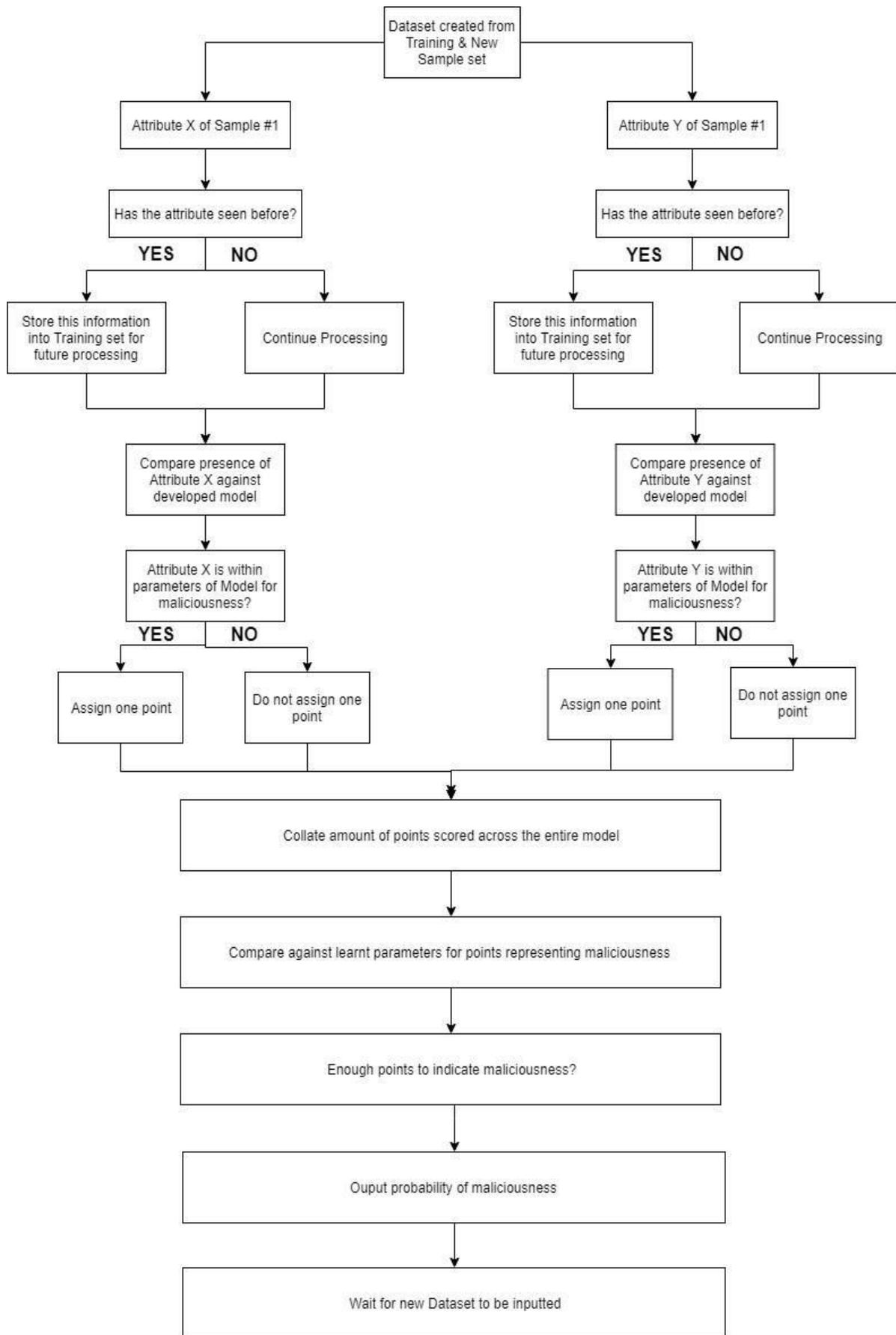


Figure 6: A flowchart presenting as a Random Forest algorithm, a complex decision tree algorithm.

The self-expansive nature of Random Forests is reflected in Figure 7, adding context to this algorithm using the research project, the creation process of the training set is outlined, including how it is devised from the collated samples and iterated through the model multiple times in these epochs.

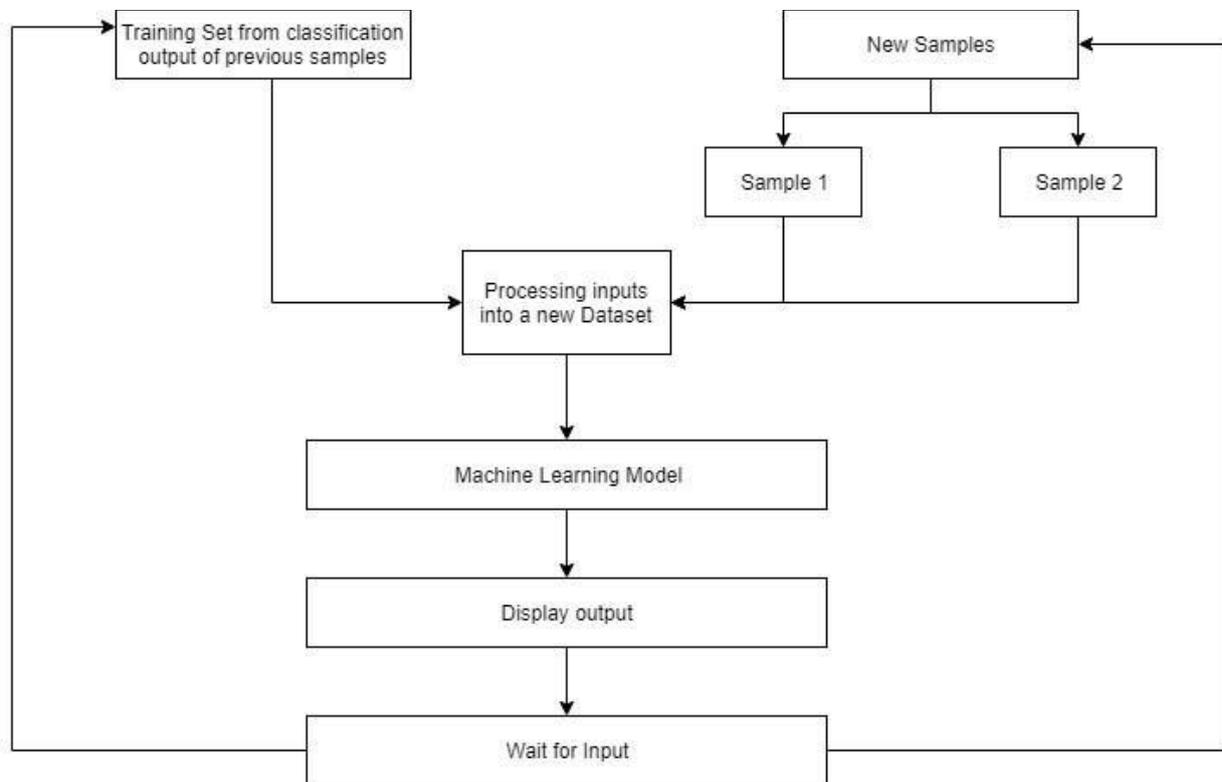


Figure 7: An illustration of how the model combines both new samples and previous performance from the training set for continuous learning

6.3. Reviewing Proposed Artificial Intelligence Models for Malware Classification

Thankfully, there has been some continued investigation into circumventing the limitations that signature-based detection retains. For example, (Schultz et al., 2000) created a framework whom “automatically extracted a binary profile from each example in [their] dataset” using “properties ... such as byte sequences”. This is an effective alternative to other frameworks proposed by researchers such as (Tesauro., Kephart and Sorkin., 1996) who only achieved a small detection rate due to only investigating PC Boot-sector viruses, of which “PC boot sectors are 512 bytes long”.

With this limited amount of code, there is a very minimal expectation of the features that can be extracted. When larger datasets containing a lot of features are created, such as that used by (A. Mohaisen., M. Mohaisen., 2015) consists of a much larger scale of “115,157 malware samples”. Whilst this study “used only a total of 65 features for classification and clustering”, their best performing algorithm achieved an 85% accuracy rate. However, it should be noted that their performance scale was calculated on both the highest yet most time-efficient algorithm.

Other explored frameworks may have achieved a higher accuracy score, however at a much higher system resource cost or timeframe, which is impractical in a real-world scenario such as implementation within a large organisation. For example, (A. Mohaisen., M. Mohaisen., 2015) used an unsupervised learning technique, whilst at a much larger cost of computing resources, achieved “more than 98%” accuracy in their datasets.

This is a noticeable improvement over alternatively suggested frameworks and algorithms such as those of (Bayer et al., 2009), who suggest that “aggressive approximate clustering techniques may need to be employed [with much larger datasets]” resulting in a loss of accuracy due to generalization. The dramatic decrease in accuracy as a cluster – hence dataset sizes increases is shown in Figure 8.

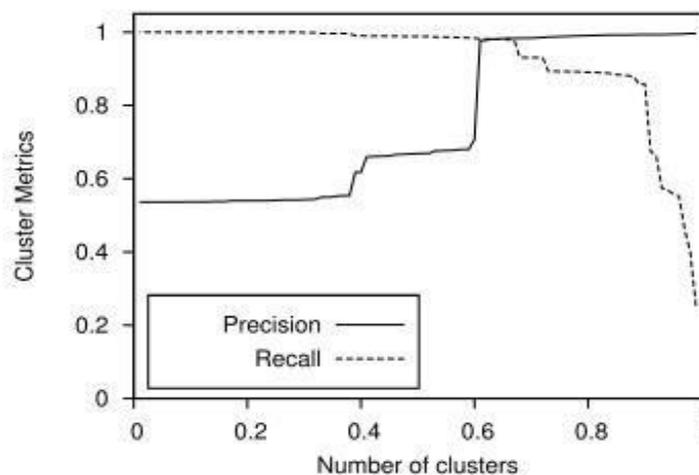


Figure 8: The dramatic decrease in accuracy Vs. precision in the increase of clustering methods (Bayer et al., 2009)

What is an apparent necessity for machine learning models is a large dataset – especially in the context of malware due to the vast range of variants for clustering classification techniques. Creating a simple classifier such as one that uses a select handful of features for classification i.e. determining if a provided picture is a cat or a dog alone still requires a sizeable dataset – often within the hundreds if not thousands - for accurate classification at the expense of computing resources to process such quantities of data in large epochs.

Binary classification appears to be a popular and successful approach to solving the issue of malware classification. Binary classification, within the context of extracting information of malware, involves the “process of classifying given document/account on the basis of predefined classes” (Kumari and Kr., 2017). To extend the context of this, examples of binary classification is a fundamental basis in malware analysis. Malware Analysts do this manually when investigating individual cases, and algorithms on a much larger scale when classifying.

Extracting behavioural patterns from object files such as .dll’s, or in the case of (Dhanyasree., Krishnan and Ambikadevi Amma., 2019) both non-verbal and verbal communication classes can be collated together to classify the legitimacy of a social-media user profile.

Combining different sets of classes that are extracted from the characteristics and information from the object files such as **.dll**'s and **.exe**'s will be a fundamental methodology for classification throughout this research project.

6.4. Discussing Proposed Academic Solutions

Data scientists have been employing traditional machine learning techniques in the classification of malicious code. Using image classification techniques where patterns are extrapolated and visualised into the human RGB formatting, the variation of data values range from three values of 255, where each variation results in an individual pixel – creating a unique, graphical abstraction of machine code.

For example, the academic publications from (Nataraj et al., 2011) proposes a framework actioning the extraction of features present within samples are represented and contextualised into an 8-bit vector graphic such as that of Figure 9, where the four essential hallmarks of a windows PE file sections are devised and the distribution of code within these sections are clearly identifiable.

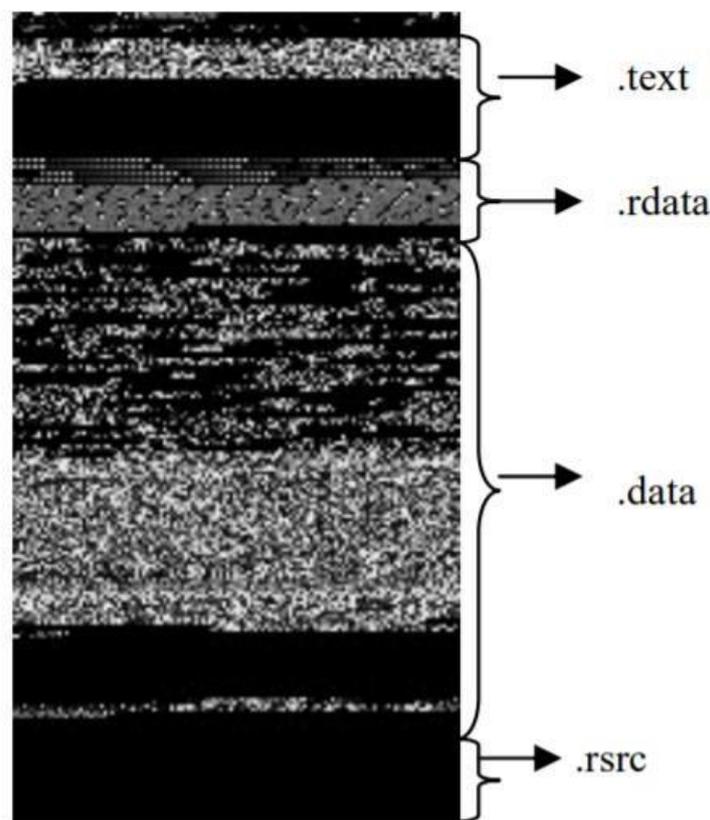


Figure 9: Content within the sections of a Windows PE file visualised as 8-bit greyscale (Nataraj et al., .2011)

A classifier will be capable of recognising the presence of numerical values, in this case, along a greyscale, and classify based the difference of these visual characteristics.

Rather than extracting strings from samples within the dataset and storing within dictionaries, frameworks portray elements including these strings for comparative analysis.

Whilst they achieved a classification rate of 98%, their framework was only implemented to classify samples of malware into respective families. Due to the characteristics of malware - namely the expectation of similarly placed bit-for-bit machine code - a classifier of this nature is expected to have a significant true-positive ratio. However, academic work such as that by (Fu et al., 2018) uses the full range of these possible RGB values (rather than greyscale) to classify samples, akin to that of Figure 10 below.

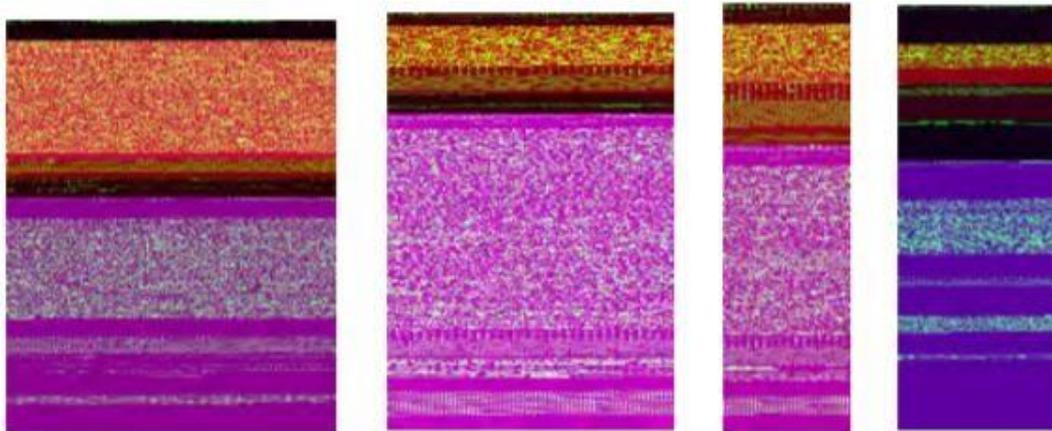


Figure 10: Using the 255-bit RGB channels as features for malware classification (Fu et al., 2018)

More RGB values permit specific characteristics of a sample to be identified from another, where these may have been lost in an 8-bit vector due to having fewer colours to portray as - allowing for machine learning models to process more characteristics of a sample. As a result, they are more accurate than classifiers using 8-bit vectors but at a much more intense computing resource usage.

Whereas (Nataraj et al., 2011) uses a comparably limited feature vector range of 320, (Fu et al., 2018) uses an extracted vector range of twenty per pixel across three 255 colour channels – unlike the 8-bit greyscale alternative; Making their classifier a lot more complicated, and higher computational cost, they achieved a precision accuracy of 99% of being able to both classify malicious intent & categorise variant shown in Figure 11 below.

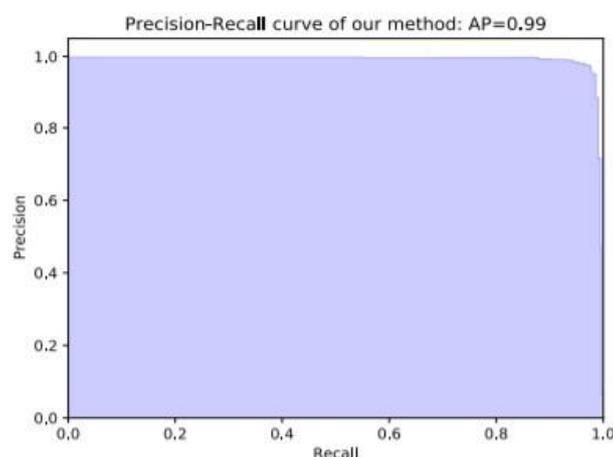


Figure 11: (Nataraj., et al., 2011) precision scoring of their framework.

Their framework reinforced the use of a K-nearest neighbours algorithm in a large-scale dataset due to the reduced growth in classifier training time, whereas a support vector algorithm uses hyper-planes to categorise samples, where the author is to select the hyper-planes of most value to the right categorisation such as in the scenario of Figure 12.

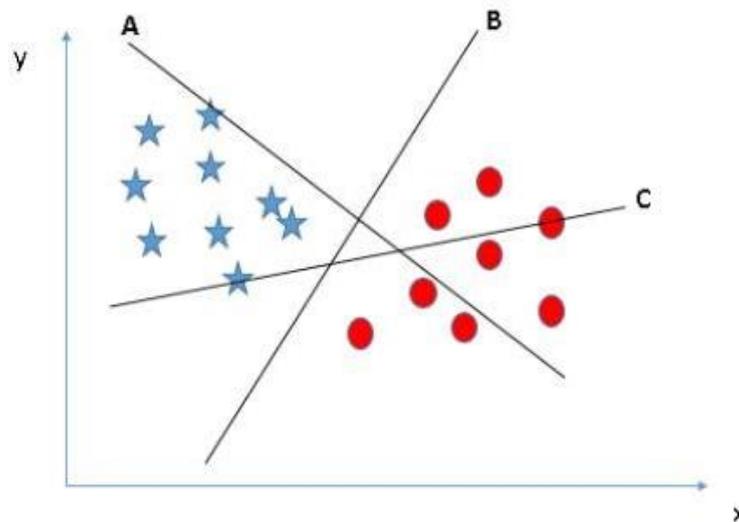


Figure 12: Three Generated Hyper-Planes To Classify Two Shapes. (Ray., 2017)

7. Methodology

7.1. Overview

When trying to solve a problem using machine learning, there are two significant algorithmic approaches that are considered, each with beneficiaries and adversaries that need to be pitted against each other on a per-scenario basis.

7.2. The Windows Portable Executable Structure

Microsoft Windows' PE file is a standard of file-formatting, insisting the structure of various objects across the family of Windows Operating System releases.

This PE structure is a prevalent ruleset that can be found across many file-types; ranging from executable files for applications to text-font files and object files - all essential for the functionality of an Operating System.

The Windows PE File comprises of eleven sections. Whilst application Authors can populate these sections as desired, each section will contain metadata, informing the Operating System of how the file should be processed within the application (e.g. defining it as an executable file rather than an object file) (Windows Developer Center., 2020) whilst the Windows Operating System family has evolved, the PE file has continued with little variation.

Microsoft's Windows Operating System has done a remarkable job at maintaining compatibility with previous generations of the Windows Operating System. The **MS-DOS** header within a PE file is an excellent example of these efforts of backwards compatibility. This header enables the Operating System to process the file and check for compatibility across the entire hierarchy of the file itself. Where there is no compatibility, rather than just crashing such as the behaviours of the first editions of MS-DOS used too, the Operating System will now just inform the user that there is no current compatibility.

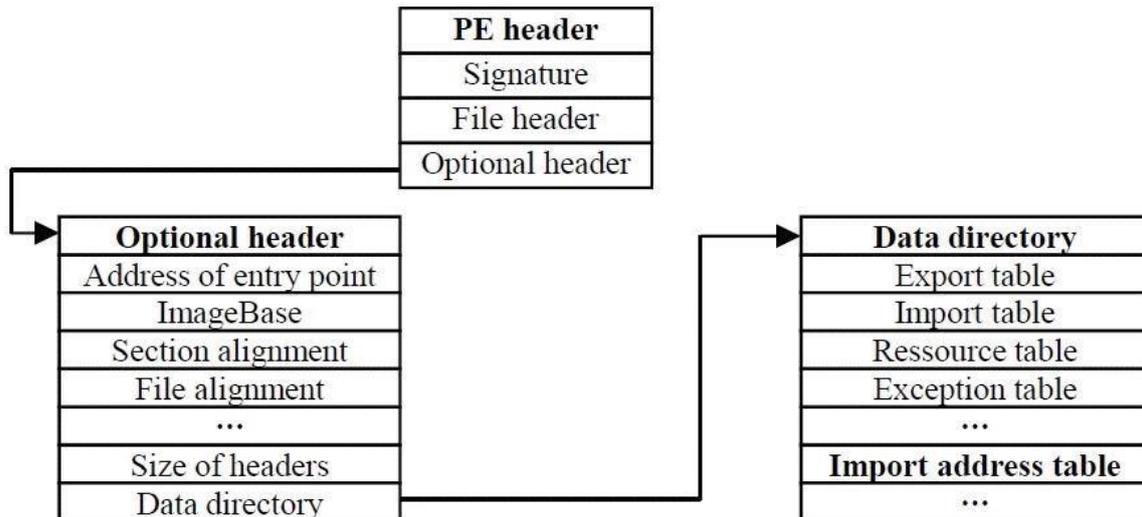


Figure 13: The Portable Executable (PE) structure with optional parameters (Belaoued. M, et al., 2016)

There can be optional parameters, which are often populated during the development process of the software attributed to the PE file. These are not essential for a PE file, for example, an icon that is a graphical way of identifying the application to the user. This is illustrated in Figure 13.

Knowledge of this structure, including the objects (or features) an executable can contain, is paramount to the feature extraction process of this research project.

7.3. Malware Analysis

7.3.1. Feature Extraction

An unequivocal necessity of using artificial intelligence to solve real-world problems is the use of feature extraction. From an observers perspective, the sample set is a mere abstraction of files – whilst their contents are of the Windows PE structure, these characteristics are plentiful and largely unnecessary as reviewed throughout other academic work and implementations. Feature extraction is the processing of data to generate values that are useful to the machine learning model whilst ignoring

those that are not. In the context of this research project, characteristics of the Windows PE structure are copious and mostly unnecessary for classification. For example, the compatibility header of the Windows PE file. The value stored within this is unnecessary in this implementation because the file is never executed – as it is only processed at its face value, there is no Operating System-level operability is needed, just that the file is indeed a Windows PE file.

For example, the number of Import functions an executable has a notable correlation between the maliciousness of a file. This is illustrated in comparing both Figure 14 and Figure 15.

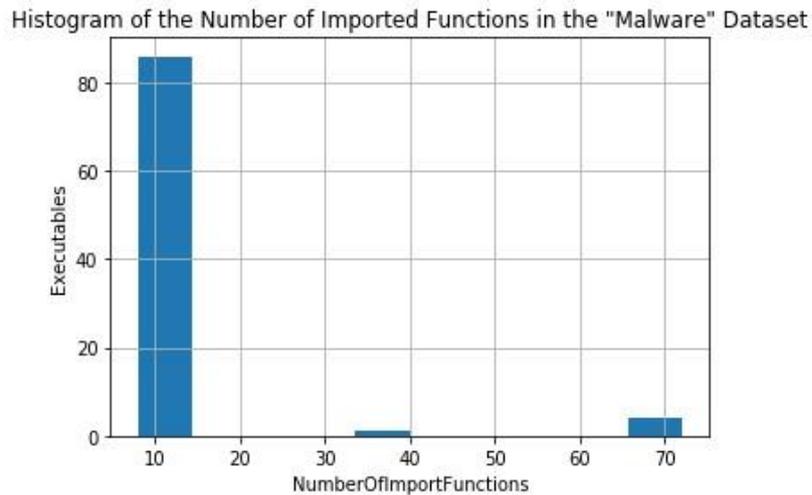


Figure 14: Quantifying the "imported functions" of the "malware" dataset

Where a total of ninety-two malicious executables have the import function count of between fifteen and seventy-three import functions. This is comparable to the numerical presence of import functions in the clean dataset, amounting to a much wider range of import functions in ninety-two executables.

The sample-set in this dataset has between zero and 470 import functions – a significant increase in comparison.

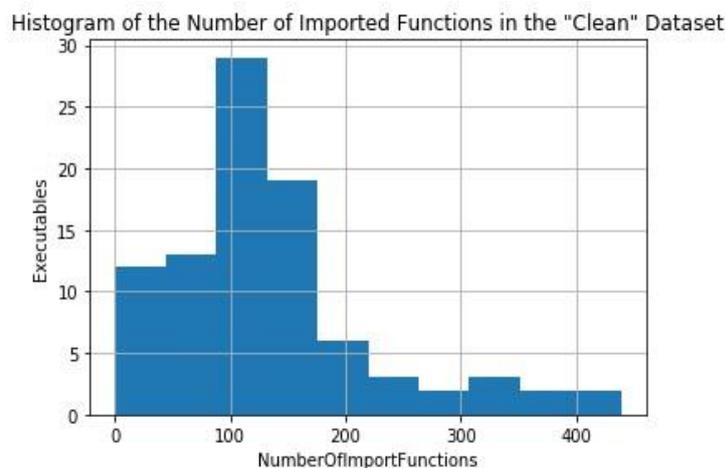


Figure 15: Quantifying the "imported functions" of the "clean" dataset

The disparity in figures like this may serve as influential roles in the maliciousness of executables. Namely “clean” samples such as software installers are often much more complicated than samples such as with an agenda of ransomware or contain features that these clean, non-malignant samples won’t, such as IP addresses and bitcoin wallets to name a few. Arguably there is no reason for standard executables to contain hardcoded networking values such as IP addresses. This idea is reinforced in the analysis of a Wannacry sample where there are only forty-two imports. Specifically directory traversal and cryptographic functions, but also contains a bitcoin wallet for payment as a hardcoded value stored within a string as illustrated in Figure 16, again, using Wannacry as an example.

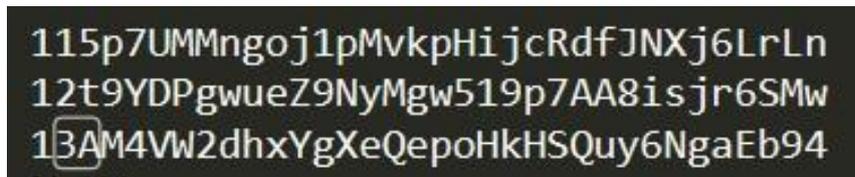
A screenshot of three Bitcoin addresses displayed on a dark background. The addresses are: 115p7UMMngo j1pMvkpHi jcRdfJNXj6LrLn, 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw, and 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94. The first character of the third address, '1', is highlighted with a white square.

Figure 16: Bitcoin addresses used for payment within Wannacry, stored as strings within the sample.

pFile	Data	Description	Value
0000808C	0000DAB2	Hint/Name RVA	017D GetModuleFileNameA
00008090	0000DAC8	Hint/Name RVA	0381 VirtualAlloc
00008094	0000DAD8	Hint/Name RVA	0383 VirtualFree
00008098	0000DAE6	Hint/Name RVA	00F8 FreeLibrary
0000809C	0000DAF4	Hint/Name RVA	0210 HeapAlloc
000080A0	0000DB00	Hint/Name RVA	01A3 GetProcessHeap
000080A4	0000DB12	Hint/Name RVA	017F GetModuleHandleA
000080A8	0000DB26	Hint/Name RVA	0328 SetLastError
000080AC	0000DB36	Hint/Name RVA	0386 VirtualProtect
000080B0	0000DB48	Hint/Name RVA	0233 IsBadReadPtr
000080B4	0000DB58	Hint/Name RVA	0216 HeapFree
000080B8	0000DB64	Hint/Name RVA	035B SystemTimeToFileTime
000080BC	0000DB7C	Hint/Name RVA	025A LocalFileTimeToFileTime
000080C0	0000DB96	Hint/Name RVA	004B CreateDirectoryA
000080C4	0000DF5E	Hint/Name RVA	01B7 GetStartupInfoA
000080C8	0000D8D4	Hint/Name RVA	031B SetFilePointer
000080CC	0000D8C6	Hint/Name RVA	031F SetFileTime
000080D0	0000D8B2	Hint/Name RVA	0117 GetComputerNameW
000080D4	0000D89A	Hint/Name RVA	0140 GetCurrentDirectoryA
000080D8	0000D882	Hint/Name RVA	030A SetCurrentDirectoryA
000080DC	0000D874	Hint/Name RVA	01F8 GlobalAlloc
000080E0	0000D864	Hint/Name RVA	0252 LoadLibraryA
000080E4	0000D852	Hint/Name RVA	01A0 GetProcAddress
000080E8	0000D844	Hint/Name RVA	01FF GlobalFree
000080EC	0000D832	Hint/Name RVA	0066 CreateProcessA
000080F0	0000D7E4	Hint/Name RVA	0034 CloseHandle
000080F4	0000D81C	Hint/Name RVA	0390 WaitForSingleObject
000080F8	0000D808	Hint/Name RVA	035E TerminateProcess
000080FC	0000D7F2	Hint/Name RVA	015A GetExitCodeProcess
00008100	0000DA6C	Hint/Name RVA	00E3 FindResourceA

Figure 17: Import Functions stored within a Wannacry sample. These would be extracted by the classifier

Careful consideration into the features extracted for use within the machine learning model is vital, this is due to not only the value that certain features contain in attributing the maliciousness intent of a sample such as those discussed, but also to prevent overfitting and a reasonable training and classification time throughout the implementation. Furthermore, it is imperative to balance the risk of overfitting and underfitting.

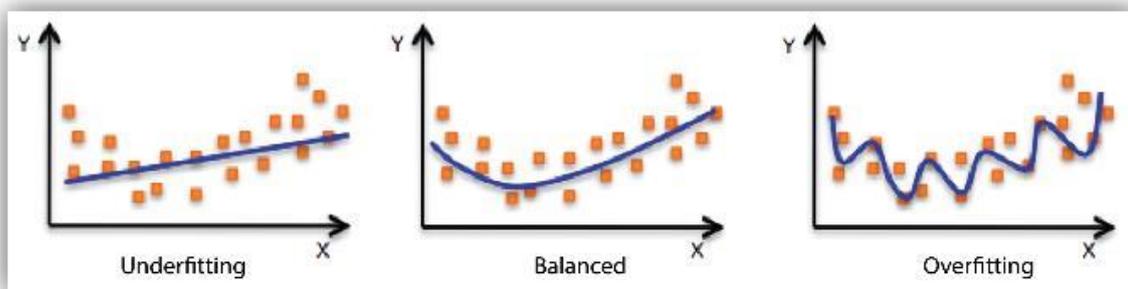
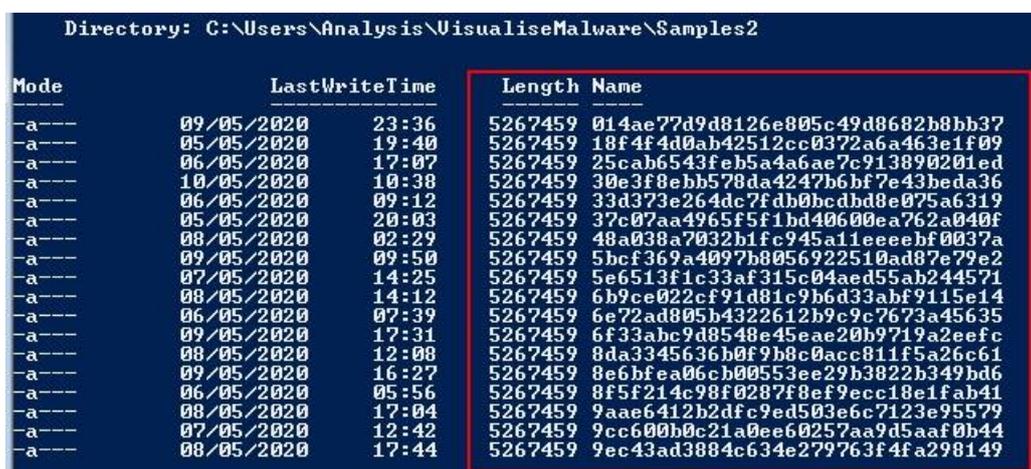


Figure 18: Comparison of the data values in respective "overfitting" and "underfitting" scenarios (Amazon Web Services., 2016)

7.3.2. Visualising Malware Similarity

An arguably less sophisticated application - albeit equally informative, of data science in the context of Malware is the case of similarity graphing and comparison. Again applying feature extraction techniques, features such as strings or networking information can be extricated from Windows PE files. Once collated, the values such as domain names can be used to build a malware sample identification framework, much rather than a classification framework that classifies samples based upon the presence of malicious intent. To develop on this, as samples are commonly referenced to within the information security community by their cryptographic MD5 or SHA-1 fingerprint, the name of the APT threat is often unknown, especially for new samples. Being able to run a sample through a framework such as this will allow APT campaigns to be identified and tracked a lot with more efficiency - assuming that a former sample of this campaign has been identified. This problem is identified in Figure 19 below, where eighteen samples with unique names as of result their MD5 hashes have the exact bit-for-bit file size as each other.



Mode	LastWriteTime	Length	Name
-a----	09/05/2020 23:36	5267459	014ae77d9d8126e805c49d8682b8bb37
-a----	05/05/2020 19:40	5267459	18f4f4d0ab42512cc0372a6a463e1f09
-a----	06/05/2020 17:07	5267459	25cab6543feb5a4a6ae7c913890201ed
-a----	10/05/2020 10:38	5267459	30e3f8ebb578da4247b6bf7e43beda36
-a----	06/05/2020 09:12	5267459	33d373e264dc7fdb0bcdbd8e075a6319
-a----	05/05/2020 20:03	5267459	37c07aa4965f5f1bd40600ea762a040f
-a----	08/05/2020 02:29	5267459	48a038a7032b1fc945a11eeebf0037a
-a----	09/05/2020 09:50	5267459	5bcf369a4097b8056922510ad87e79e2
-a----	07/05/2020 14:25	5267459	5e6513f1c33af315e04aed55ab244571
-a----	08/05/2020 14:12	5267459	6b9ce022cf91d81c9b6d33abf9115e14
-a----	06/05/2020 07:39	5267459	6e72ad805b4322612b9c9c7673a45635
-a----	09/05/2020 17:31	5267459	6f33abc9d8548e45eae20b97719a2eefc
-a----	08/05/2020 12:08	5267459	8da3345636b0f9b8c0acc811f5a26c61
-a----	09/05/2020 16:27	5267459	8e6bfea06cb00553ee29b3822b349bd6
-a----	06/05/2020 05:56	5267459	8f5f214c98f0287f8ef9ecc18e1fab41
-a----	08/05/2020 17:04	5267459	9aae6412b2dfc9ed503e6c7123e95579
-a----	07/05/2020 12:42	5267459	9cc600b0c21a0ee60257aa9d5aaf0b44
-a----	08/05/2020 17:44	5267459	9ec43ad3884c634e279763f4fa298149

Figure 19: Calculating the cryptographic checksums of the eighteen samples in this dataset.

It is logical to assume that these samples are of the same APT campaign due to their exact file sizes, despite their different file names. However, to confirm this, every file will have to be analysed – a lengthy process for a malware analyst; this is a minute example of large datasets that April have hundreds of these similarly presented files.

In this implementation, the samples are analysed by extracting their strings – who often contain hardcoded networking information such as domain names and/or IP addresses. The framework looks for the domain name TLDs or suffixes in Figure 20 below, where any presence is stored within the framework in case any other sample has the same domain name suffix.

```
domain_tlds = ['com', 'tor', 'co', 'co.uk', 'org', 'eu', 'net', 'us']
```

Figure 20: Domain TLD's recognised by the implemented application

If another sample has the same domain name suffix, it is rendered into a graph using GraphViz, creating a visualisation of that in Figure 21.

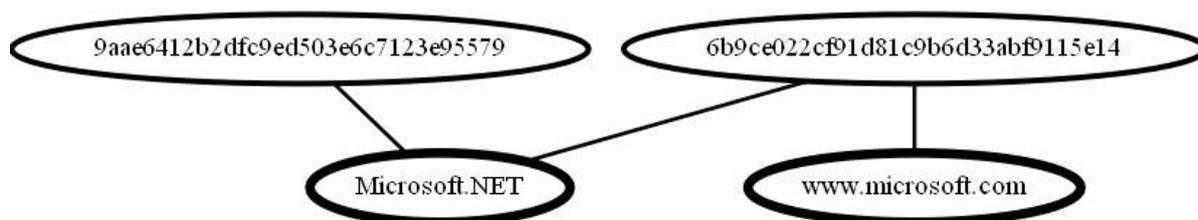


Figure 21: Generated output displaying similarities between two samples

Out of the eighteen analysed samples, only two were identified as being similar. When these files are run through online sandboxing, they are identified as being apart of the Wannacry APT campaign. Illustrated in Figure 22 , sample “6b9ce022cf91d81c9b6d33abf9115e14” has strings contacting www.microsoft.com in which the graph has successfully displayed. Analysing this sample in an online sandbox environment confirms the presence of this networking information.

```
.http://www.microsoft.com/pki/certs/CSPCA.crtO  
.http://www.microsoft.com/pki/certs/tsPCA.crtO
```

Figure 22: Verifying networking communication performed by the sample

Sample “6b9ce022cf91d81c9b6d33abf9115e14” contacts Microsoft.com after analysis in the HybridAnalysis sandbox environment

However, one flaw of this framework in this example is this false-positive detection of what seems to be a domain name; this is due to the presence of “.net” where there is a simple reference to the “Microsoft.NET” programming library, and not domain name such as that in Figure 23 on the following page.

```
M%WINDIR%\Microsoft.NET\Framework\v2.0.50727\cvtres.exe
O%WINDIR%\Microsoft.NET\Framework\v2.0.50727\mscorsvw.exe
olume2\Windows\Microsoft.NET\Framework\v3.0\Windows Communication Foundation\infocard.exe
```

Figure 23: A false-positive identification of similarity between two samples

Samples “9aae6412b2dfc9ed503e6c7123e95579” and “6b9ce022cf91d81c9b6d33abf9115e14” have references to the Microsoft.NET programming library, resulting in false-similarity identification in this context.

Moreover, another flaw is that often malicious samples such as the Wannacry ransomware are self-extracting; what appears to be one file is, in fact, multiple more, where different functionalities are contained within, akin to that of Figure 24. The framework will not analyse these files as that will require execution – past the remit of static analysis; potentially resulting in fundamental identifiable attributes and behaviours being missed by the classification model.

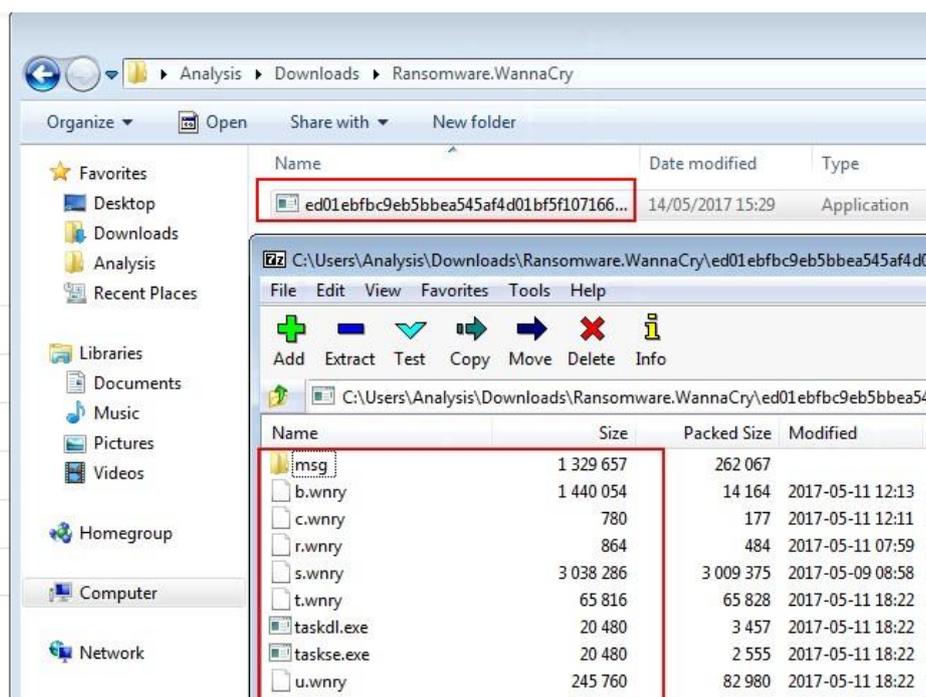


Figure 24: Example of how one presented file can be self-containing much more malicious content.

It could be argued that a larger dataset of malicious samples is needed to reduce this false-positive ration. This theory was tested with 96 known to be malicious samples, a sample increase of 433.33%, where their presented strings were extracted and processed by the same framework. What is interesting is that only one sample, “b9bef10e5b2b11a40f9b330256fcb38e”, had any domain name strings extracted and visualised, as illustrated in Figure 25 below.

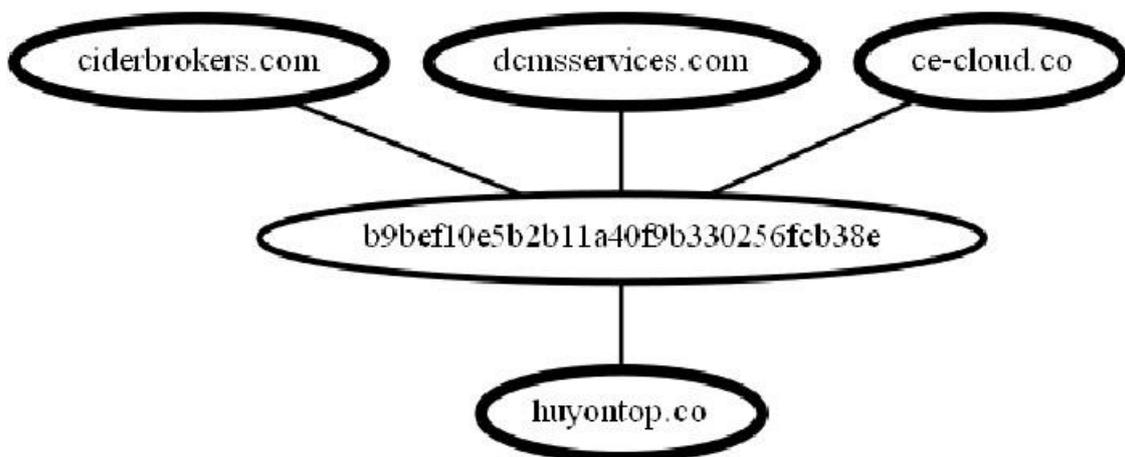


Figure 25: Visualisation of a sample contacting four domains

However, only fourteen samples were visualised as having a similar domain name within their strings as currently presented without execution. Alas, these all referred to the Microsoft.NET programming library, another false-positive.



Figure 26: Large scale abstraction of eleven samples contacting a domain

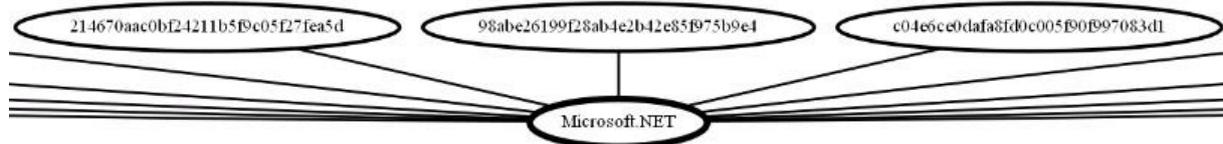


Figure 27: Scaled-up abstraction of three samples out of the generated output

This leaves a total of eighty-one other samples having no presented strings correlating to any domain name suffixes. Sample “ae12bb54af31227017feffd9598a6f5e” reinforces the grave error of these samples not presenting as having any domain name strings within.

When analysed, the sample in question does, in fact, contact an external domain TLD and countless IP addresses in Figures 29 and 30 on the following page.

Domain	Address	Registrar	Country
www.iuqerfsodp9ifjaposdfjhgosurijfaewrgwea.com OSINT	104.17.41.137 TTL: 299	NAMECHEAP INC Name Server: BRUCE.NS.CLOUDFLARE.COM Creation Date: Fri, 12 May 2017 00:00:00 GMT	United States

Figure 28: A suspicious domain contacted by the sample

38.107.105.34	63894 TCP
175.79.238.59	63906 TCP
211.127.182.168	63917 TCP
131.74.121.165	63919 TCP

Figure 29: Suspicious IT devices contacted by the sample

After further analysis, this sample is also apart of the Wannacry APT campaign but was totally ignored. With only a 14.58% chance of visualisation, let alone the chances of 1.04% of correct identification in this dataset, a potential use case of this type of framework being the only barrier in identifying malware is a disastrous implementation.

7.3.3. Current Automated Solutions for Malware Classification

At present, whilst the market is expanding rapidly, there is still limited saturation of commercial services specialising in malware classification.

When they are present, they are locked behind expensive paywalls – suitable for only large enterprise settings. For enterprises, this cost is negligible when they are on the receiving end of a wide spectrum of APT's. Services such as Hybrid Analysis execute samples in real-time on dedicated IT devices, where behaviour is monitored and then ultimately analysed. The process, for a regular consumer, ranges between ten and fifteen minutes, where only one sample can be analysed at a time – up to a maximum of five in twenty-four hours.

7.4. Dataset Distribution

7.4.1. Creating a “clean” Sampleset

Multiple sources for sample collection was considered and tested, accumulating into two main types of “clean” datasets for use throughout the machine learning model. These two different datasets included two divergent uses cases of Windows PE functionalities. For example, “clean” dataset one, henceforth referenced too as the “fresh-host” dataset is composed of ninety-two system executables and software installers. We label this dataset as **clean** in the parameters that the samples are collated from a fresh-install of an Operating System – in this instance, Windows 7 Professional 32 bit.

Within this context, the source of the dataset is a “Virtual-Machine” using the commercial-platform “VMware Workstation Pro 15” on-top of a Windows 10 Operating System henceforth referred to as the Hypervisor.

A fresh-install of the Operating System onto the host entails installing the Operating System using a digital-replica of the traditional installation disk, with no additions to the post-installation state. It is essential that no additional software is installed on this host, as such action may render the dataset as tainted (i.e. security patches).

View basic information about your computer

Windows edition

Windows 7 Professional
Copyright © 2009 Microsoft Corporation. All rights reserved.
Service Pack 1
[Get more features with a new edition of Windows 7](#)



System

Rating: [System rating is not available](#)
Processor: Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz 4.00 GHz
Installed memory (RAM): 3.00 GB
System type: 32-bit Operating System
Pen and Touch: No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name: WIN-R49LR4M2PBI [Change settings](#)
Full computer name: WIN-R49LR4M2PBI
Computer description:
Workgroup: WORKGROUP

Figure 30: Details of the environment used for the abstraction of samples for the “clean” dataset

The efforts to ensure that the host remains **clean** is continued throughout collation of data by ensuring the host remains disconnected from the Internet. An expected procedure of a fresh install of the Windows Operating System is to “call home” and check for security updates. The updates that are downloaded are varied based upon numerous factors, such as:

- Service Pack version
- License type e.g. Windows Home, Windows Professional
- 32bit or 64bit

This is problematic as these updates introduce a large number of variables when collecting the dataset. For this research project, we want to keep the baseline as generic - hence applicable as possible.

The samples collected are from “Windows” as this is the Operating System’s users-abstraction of the Kernel-layer. The Operating System at “Levels 2” and “Level 3” is merely an abstraction of the Level 0 – the Kernel, who is not directly accessed by the user, but through the Operating System itself, as visualised in Figure 31.

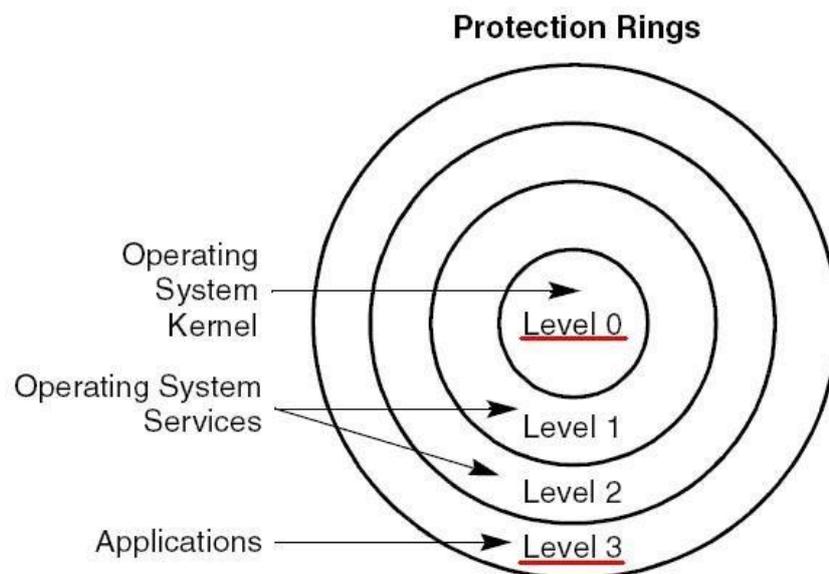


Figure 31: Operating System "Protection Rings" Diagram (Montana State University., 2005)

In the context of a post-install state, the host contains an approximate 11,005 .DLL object files. The actual result April vary, as some object files are not visible or accessible to the user as intentional from the Operating System. The alternative dataset curated as a “clean” sample set for this research project is a collation of files originating from published software installers and executables.

An archival file host, FileHippo, a service to make installers available to everyone who needs them, is used throughout this research project to create a clean real-world executable dataset...

...Executables contained within this dataset are of the fruition of commercial software developers and services for the purposes of application installation. These types of files contain similar behaviour to that expected to be exhibited from the samples in the malicious dataset, making a valuable training resource for the framework proposed, as ultimately malware samples, at an abstraction, are software applications written by software developers albeit with malicious intent. Because of these similarities, the framework can be created to extract comparable features and classify based upon the deviations against the clean dataset.

7.4.2. Creating a “malicious” Sampleset

The use of Honey pots, an extremely valuable tool suite available to malware analysts and APT trackers alike was considered throughout this research project. For example, The HoneyBow toolkit, a high-level interaction Honey pot developed by (Zhuge et al., 2006) "integrates three malware collection tools ... [rather than emulating, uses] truly vulnerable services as victims to lure malware infections" this method allows real-case and potentially never seen before zero-days to be captured for analysis, where these samples will have never of been investigated by an A.V engine before. Whilst this type of data is most ideal in theory because these samples have never been classified before via an A.V engine; due to the nature of the classifier, any sample parsed through that is not apart of the training set is considered a never seen before sample – in effect, there is no known knowledge regarding it. Figures 32 and 33 on the page below illustrate the key conceptual differences between high-level and low-level interaction Honey pots.

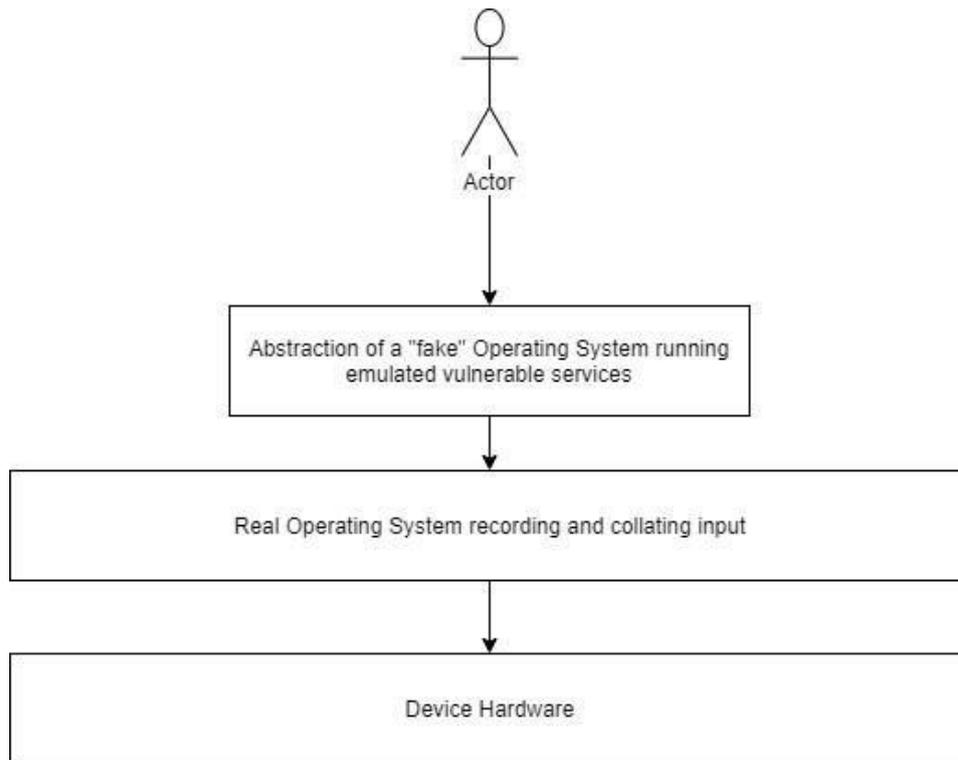


Figure 32: Visualisation of a low-level interaction Honeypot

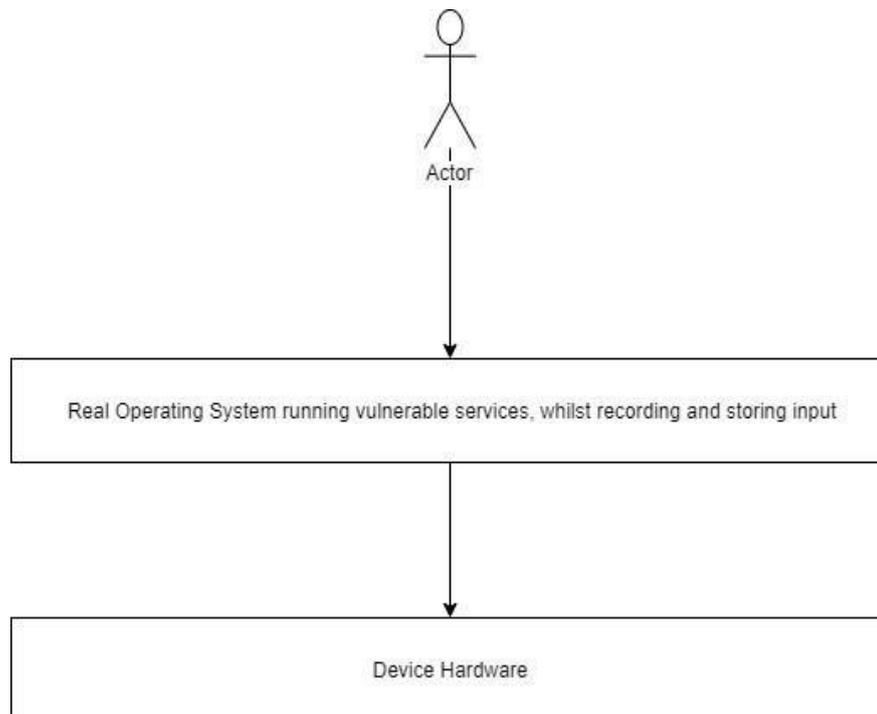


Figure 33: Visualisation of a high-level interaction Honeypot

Dionaea, an open-source and low-interaction honeypot application collates malware samples by exposing vulnerable services to the internet (P. Bo., 2015).

Because of the capabilities offered by this application, this specialised software was used to capture current and present malware samples in circulation across the internet today. Spanning across two cloud-computing resources hosted on Amazon Web Services (AWS), these resources, termed as “instances” are internet-facing on managed infrastructure. Extraordinary steps have been taken to ensure that these instances remain within the AWS terms of service, specifically ensuring responsible handling of data at rest and in transit. Data stored within these instances are encrypted at the block level with individual keys to each instance, meaning no data can be un-encrypted and interacted with outside of this instance. Moreover, although the instances are internet-facing, whitelisting-based firewall rules have been implemented to ensure no networking traffic can egress from each instance unless specified on an IP address basis.

The only egress these instances can make is to the IP address of the analysis machine used in the research project, where any communication to this is also encrypted over the Secure Socket Shell (SSH) – a modern industry standard for communication between client/server. No potentially malicious code is executable as the collected files are only capable of executing on a Windows environment and not Linux, so there is no risk to infrastructure. With these implementations in place, these honeypots are compliant to terms of service (AWS Service Terms., 2020). Each instance is running the following vulnerable services to collate samples for the dataset:

- DiskStation FTP
- Samba 4.3.11
- MySQL Server

8. Appraising the Applied Machine Learning Techniques

Before the datasets were run through various classification models, the datasets needed to be processed into more suitable formatting for the classification model, namely to prevent overfitting and to increase maximum assurance of the generated testing metrics. Both datasets are loaded into Jupyter and then compared.

	Unnamed: 0	AddressOfEntryPoint	DebugRVA	DebugSize	Dll	ExportRVA	ExportSize	IATRVA
0	00070b0d4cb037c40d5d2464f92841aeb9ad863472bf95...	21704.0	4880.0	28.0	0.0	0.0	0.0	4096.0
1	00696555cbf6db83af785f8acb2270b9411cfc75e7f6d3...	29424.0	4256.0	28.0	49472.0	32576.0	163.0	36864.0
2	007247436f041ca59c5ee0e8636c668c2a43376aeb8cfa...	227872.0	82400.0	84.0	49472.0	0.0	0.0	3522560.0
3	007bdab757d03d94e60c9b1e3eec13b07562705c514992...	10656.0	4320.0	28.0	49472.0	0.0	0.0	20480.0
4	008fa2b9697f9a173e40572face100410e51975e34a5ce...	152696.0	4224.0	28.0	33120.0	0.0	0.0	200704.0
5	00fa0679d4d159772298f2fafc2d349620264a592f900a...	4755.0	12576.0	84.0	33088.0	0.0	0.0	12288.0
6	012426c78cc11ace9c6789be4da0fbbd2cfd7cddc4df3...	188842.0	422864.0	84.0	33088.0	0.0	0.0	307200.0
7	01579b7ac743f18645c1ac3ca8b3dbe0de0e20b7fbb98e...	25776.0	4320.0	56.0	49472.0	0.0	0.0	172032.0
8	01ae2f2e549cda25720972cac5fffb27eda2255fb67d0c...	12707.0	0.0	0.0	34112.0	0.0	0.0	32768.0
9	01b38fd2c1df5dafdaaff096fb5d4f93697428afcb98ef...	21200.0	4368.0	28.0	49472.0	0.0	0.0	28672.0

10 rows × 2699 columns

Figure 34: First 10 files of the clean dataset extracted and described in Jupyter using Matplotlib

In Figure 35, a correlation matrix is plotted after comparing the values contained within columns of the two datasets compared.

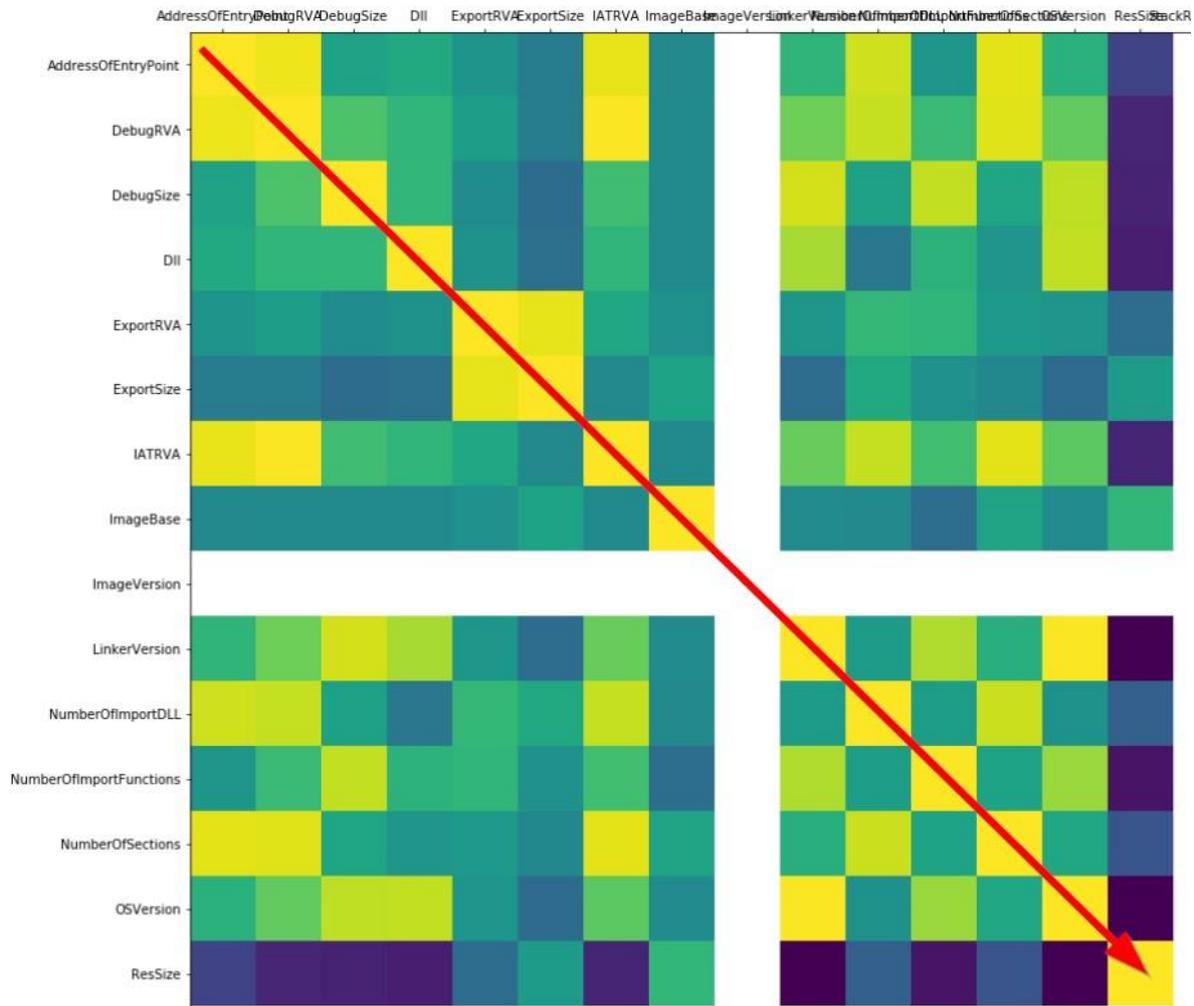


Figure 35: A correlation matrix of the features between the two datasets used in the classifier

Visualised in the colour scale from least correlation of darker colours, blue and purple, towards the most correlation canvased by green and then yellow. There is a noticeable trend in the similarity between the values of the columns within the two datasets, a promising trend of classification because of the features extracted, such as import functions and sizes.

Out[78]:

	Unnamed: 0	AddressOfEntryPoint	DebugRVA	DebugSize	Dll	ExportRVA	ExportSize	IATRVA	ImageBase
0	0129086ae5fa2269d1037ff0ac0fca48	70427	117040	28	320	130688	358	139804	268435456.0
1	01404bb45219113d64965f76803deddb	4585	0	0	0	8592	72	8192	268435456.0
2	014ae77d9d8126e805c49d8682b8bb37	4585	0	0	0	8592	72	8192	268435456.0
3	017f63d0be693e53bc5b8edd426cfbd1	4585	0	0	0	8592	72	8192	268435456.0
4	01bdc6fb077098f4a3b60f4b0e479a7f	4585	0	0	0	8592	72	8192	268435456.0

5 rows x 22 columns

Figure 36: First five files of the clean dataset

Out[77]:

	Unnamed: 0	AddressOfEntryPoint	DebugRVA	DebugSize	Dll	ExportRVA	ExportSize	IATRVA	ImageBase
177	142c996c50811f1be1d343bf57f7f65ec535e94db0381d3...	48640	4304	28	49504	0	0	61440	5.368709e+09
178	14376ddd07cf8df48803a3c0e606a66f6250bbe56e6fd9...	14688	4192	28	49472	0	0	24576	4.194304e+06
179	1445bcc87cac845d42f21517a251871f8c949436c9ee15...	60208	4176	28	49472	0	0	69632	4.194304e+06
180	155bb96bc972f227c563e2c3db12ea00d212af6c1fe882...	47331	78320	28	33088	0	0	77824	4.194304e+06
181	15667792199cc764a62e3e61f5380b08298a107b12c7c3...	17920	4192	28	49504	0	0	28672	5.368709e+09

5 rows × 22 columns

Figure 37: First five files of the malicious dataset

These two separate datasets are then combined together, where a binary value of either “1” or “0” is appended as to whether or not it is from the malicious sample set or not, where this value is used by the classifier in the prediction metrics and not the performance metrics, as detailed below in Figures 38 and 39 on the following pages.

filename	Malware
3ae5fa2269d103...	1
45219113d6496...	1
d9d8126e805c4...	1
l0be693e53bc5b...	1
fb077098f4a3b6...	1

Figure 38: The "malware" dataset being appended with a "Malware" classification for the training set

filename	Malware
1343fbf5f7f65ec5...	0
3a3c0e606a66f6...	0
21517a251871f8...	0
1e2c3db12ea00d...	0
e3e61f5380b082...	0

Figure 39:: The "clean" dataset being appended with a "Malware" classification for the training set

Once these two datasets are combined, we remove one sample from each dataset - both malicious and clean. This is to create an initial testing parameter, where there will always be at least one malicious and clean sample where it is not classified, but more so the values of both form a baseline.

```
In [84]: amount_dataset = len(merged_df)
amount_malicious = len(merged_df.loc[merged_df['Malware'] == 1])
amount_clean = len(merged_df.loc[merged_df['Malware'] == 0])

print("")
print("Amount of Malicious files not used as initial training: {}".format(amount_malicious, (amount_malicious/amount_dataset)))
print("Amount of Clean files not used as initial training: {}".format(num_false, (amount_clean/amount_dataset)))
print("")
```

Amount of Malicious files not used as initial training: 91
Amount of Clean files not used as initial training: 91

Figure 40: Two samples from both datasets being removed from classification to be set aside for a testing parameter

In total, this leaves a total of 182 samples to be analysed. From this, two categories for the classifier can be created:

- Training
- Testing

Original # of Malicious files: 91

Original # of Clean files: 91

Amount split for Training - Malicious: 62

Amount split for Training - Clean: 65

Amount split for Testing the Classifier (un-seen samples) - Malware: 29

Amount split for Testing the Classifier (un-seen samples) - Clean: 26

Figure 41: The split-up of the dataset into "Testing" and "Training" respectively

With a heavy bias on splitting the samples more into training then classifying, due to the arguable disparity in the correlation of the cohort of samples. This method also helps to prevent overfitting after the epochs as data from the sample set are split and tested until there is no unique splitting. When this runs out, the behaviours learnt by the classifier are forgotten and the next cohort of epochs are considered brand new by the classifier.

9. Results and Analysis

9.1. Discussing the Measurements of Results

The performance metric of classification-based artificial intelligence research projects is measured in four ratios:

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)

Whilst the individual influence that these four ratios weigh on the measured success of the project, this use case measures them with data being classified, aptly, Malware in mind. With Malware, you want to achieve the highest True-Positive and False-Positive values. It is safer to assume that files are malicious then discount them as not. Henceforth, the results of TP dictate the accuracy of the classifier, whilst the FN dictates the reliability, where a higher FN over TP ratio indicates overfitting and/or classification, where the classification technique will need to be revisited.

In the case of this project, there are two weighable labels for the performance date:

- Recall (R)
- Precision (P)

$$R = \frac{TP}{(TP + FP)}$$

Figure 42: Recall label calculated from $TP / (TP + FP)$

$$P = \frac{TP}{(TP + FN)}$$

Figure 43: Precision label calculated from $TP / (TP + FN)$

An algorithms classifier R-value is an indication of how it is performing on samples that are not training data, essentially the success of the algorithm in a real-world application. Whereas, the P-value is how well that algorithm is performing on training data – a lab scenario. These results are weighted to an average, where this is used to measure its success.

The weighted average is calculated from two averages, micro and macro formulating expressions of those in Figure 44 and Figure 56.

$$\text{Micro - Average Precision} = \frac{TP1 + TP2}{(TP1 + TP2 + FP1 + FP2)}$$

Figure 44: Equation for how Micro-Average Precision value is calculated

$$\text{Macro - Average Precision} = \frac{TP1 + TP2}{(TP1 + TP2 + FN1 + FN2)}$$

Figure 45: Equation for how Macro-Average Precision value is calculated

The average between these two precision labels formulates the total average “precision”. A similar technique is used for the “recall” label.

9.2. Actual Results

9.2.1. Logistic Regression

This algorithm selects a variety of features for each epoch, where the accuracy of the testing data is compared against the accuracy rate obtained throughout the training epochs, the variation of features tested are obtained through obtaining the best line of fit through the most features, more so than the importance of a feature in classifying malicious behaviour.

Achieving a weighted P average of 0.95, a 95% chance of correctly classifying samples is inferred against the training data, however, with the same weighted average and classification chance on unseen data, this algorithm is the highest result out of those implicated so far.

Classification Report		
	precision	recall
0	1.00	0.88
1	0.91	1.00
micro avg	0.95	0.95
macro avg	0.95	0.94
weighted avg	0.95	0.95

Figure 46: Report of the accuracy results of the P and R labels for a classifier using Logistic Regression

9.2.2. K-Nearest Neighbours

Implementing this algorithm leads to an arguably increased chance of deviation as the epochs increase, or more rather, the values of both micro and macro averages will deviate. K-Nearest Neighbours looks at surrounding features if the classifier finds a comparable value between the training data and the presented file. Where, for example, the value of K is equal to the number of features or neighbours surrounding the selected feature as per the best fit.

The ultimate value of the surrounding neighbours, for example, if they are malicious features will dictate its classification. To contextualise:

$$K = \frac{MF1 + MF2 + MF3}{CF1 + CF2}$$

Figure 47: Mathematical equation of how the number of neighbours is calculated in K-Nearest Neighbours

Where MFx and CFx represent Malicious Feature and Clean Feature. In this case, K=5, however, the presence of three MF in counterpart to two CF equates a probable classification of malicious in this example.

	precision	recall
0	0.47	1.00
1	0.00	0.00
micro avg	0.47	0.47
macro avg	0.24	0.50
weighted avg	0.22	0.47

Figure 48: Report of the accuracy results of the P and R labels for a classifier using K-Nearest Neighbours

In this algorithm, there was a P and R label of 0.47% or 0.22% both against training and unseen sample sets. Due to how this algorithm works, an increase in epochs is expected to return a similar scoring because it takes no influence from previous results. The high-scoring as a result of the training set could be a result of the use of small dataset size in comparison to other implementations, but also, due to how the algorithms assume that a sample must be the same as its neighbours. A small sample set increases the likeliness of this K= integer being of a higher value, where an increase of features are to be construed as being akin together in terms of maliciousness purely based upon their respective distance to where the algorithm deems the best fit.

9.2.3. Random Forest

With the lowest weighted R and P averages, this algorithm was the lowest-performing in comparison to other classification models. Achieving a feeble accuracy scoring of 0.17% in precision and a 0.34% recall rate, this algorithm, using supervised learning, is nor real-world and simulated lab environment appropriate for classification.

Classification Report		
	precision	recall
0	0.42	1.00
1	0.00	0.00
micro avg	0.42	0.34
macro avg	0.19	0.37
weighted avg	0.17	0.34

Figure 49: Report of the accuracy results of the P and R labels for a classifier using Random Forest

Random forests use ensemble techniques to pool together the values decided across many decisions trees as per the operation of the algorithm. Decision-tree based techniques like these use a formula similar to that below in Figure 50:

$$\text{Leaf importance} = \frac{x_{\text{samples}}}{y_{\text{total samples}}}$$

Figure 50: Calculation of "leaf" importance

The weight of features, referenced as leaves, are dictated by the number of samples in a dataset that matches with the leaf (x) divided by the total number of samples (y) in the dataset. The weight of each feature across all decision trees is then collated where the classification is then made.

Scikit, the python machine learning library, creates leaves or known as nodes based upon the amount of features present within the dataset, annotated in Figure 51.

$$f_i = \frac{\sum_{j:\text{node } j \text{ splits on feature } i} n_{ij}}{\sum_{k \in \text{all nodes}} n_{ik}}$$

Figure 51: Equation formulating how leaves are created and split across decision trees (Ronaghan., 2018)

10. Conclusions and Discussion of Future Prospects

Reassessing the accuracy and reliability scoring of the appraised algorithms, out of the three implementations used throughout the methodology In this project, the non-supervised logistic regression algorithm had the highest accuracy in classification against both training and unseen sample sets.

With a performance ratio of 0.95% in both training and un-seen sample categorisations, there is minimal overfitting, albeit there being twenty features to determine a classification; this is proven by such similar weight average in both the Recall and Precision labels.

Performance of a higher metric was somewhat expected against other models tested, such as the Random Forest algorithm due to the limited dataset size. The scoring is also similar to that of the academic work discussed previously.

Revisiting the objectives and questions desired for discovery in this project, objectives have been met. For example, similarities between malicious code were identified and attributed together, where these attributes were used to successfully apply machine learning models to a standard of possible implementation within the real world.

However, in reflection, some models such as the Random Forest April have performed better without the limitations experienced throughout the project. Namely, the limited dataset size. Whilst this is a manageable size for Logistic Regression, Random Forest is another classification model, however, whilst supervised, requires a larger scale number of samples within the dataset due to the number of decision trees and nodes created by its algorithm.

The limited dataset size was due to time and financial constraints. Running the honeypots used to collate the samples, where duplicate samples were removed, was limited to two devices due to these factors. Even with months, ultimately, the dataset was restricted to the collection from these devices.

In the future, a sizeable dataset would be ideal, where perhaps different types of malware are researched, where the features extracted are not only those of ransomware, but perhaps a different type such as keyloggers or worms. Moreover, another limitation experienced throughout the project is due to static analysis of files, where their presented behaviours April be contrasting to the behaviours presented when executed.

Finally, it would be interesting to see how techniques like malware visualisation as used in this project can be combined with machine learning models and how the accuracy April be influenced.

11. References

Amazon Web Services, 2016. *Underfitting Vs. Overfitting*. [image] Retrieved from:

<https://docs.aws.amazon.com/machine-learning/latest/dg/images/mlconcepts_image5.png>

[Accessed 4 April 2020].

Amazon Web Services. 2020. *AWS Service Terms*. [online] Retrieved from:

<<https://aws.amazon.com/service-terms/>> [Accessed 12 April 2020].

Anyanwu, M. and Shiva, S., 2009. Comparative Analysis of Serial Decision Tree Classification

Algorithms. *International Journal of Computer Science and Security*, [online] 3(3), pp.3-4. Retrieved

from: <<https://pdfs.semanticscholar.org/d5af/379fb7b66ecce77c5e99fda15050e8b90157.pdf>>

[Accessed 14 April 2020].

Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., & Kirda, E. (2009). Scalable, Behavior Based Malware Clustering. P 11-12.

Bazrafshan, Z., Hashemi, H., Fard, S. and Hamzeh, A. (2013). A Survey on Heuristic Malware Detection Techniques. Conference on Information and Knowledge Technology, 5, pp.113-116.

Retrieved from: <https://ieeexplore.ieee.org/abstract/document/6620049>.

Bo, P., 2015. *Welcome To Dionaea'S Documentation! — Dionaea 0.8.0 Documentation*. [online] Dionaea.readthedocs.io. Retrieved from: <<https://dionaea.readthedocs.io/en/latest/>> [Accessed 8 Januaray 2020].

Breiman, L. and Cutler, A., 2004. *Random Forests - Classification Description*. [online]

Stat.berkeley.edu. Retrieved from:

<https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#features> [Accessed 14 April 2020].

Chui, M., Manyika, J., Miremadi, M., Heneke, N., Chung, R., Nel, P. and Malhotra, S. (2019).

NOTES FROM THE AI FRONTIER INSIGHTS FROM HUNDREDS OF USE CASES. McKinsey Global Institute, pp.3-20. Retrieved from:

<https://www.mckinsey.com/~media/mckinsey/featured%20insights/artificial%20intelligence/notes%20from%20the%20ai%20frontier%20applications%20and%20value%20of%20deep%20learning/notes-from-the-ai-frontier-insights-from-hundreds-of-use-cases-discussionpaper.ashx>.

Denil, M., Matheson, D. and Freitas, N., 2014. Narrowing the Gap: Random Forests In Theory and In

Practice. *International Conference on Machine Learning*, [online] 32(31), pp.665-673. Retrieved from:

<<http://proceedings.mlr.press/v32/denil14.html>> [Accessed 14 March 2020].

Detection. *2016 6th International Conference on IT Convergence and Security (ICITCS)*. Retrieved from: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7740362>.

Dhanyasree, P., Krishnan, S. and Ambikadevi Amma, T. (2019). Deception Detection in Social Media through Combined Verbal and Non-Verbal Behavior. *International Journal of Advanced Research in*

Computer Science and Software Engineering, 5(4), p.446. Retrieved from:

<https://pdfs.semanticscholar.org/6488/57a8208b1af246f17ad08894237aa4a00c3c.pdf>.

- FU, J., XUE, J., WANG, Y., LIU, Z. and SHAN, C., 2018. *Malware Visualization For Fine-Grained Classification*. [ebook] Beijing: Beijing Institute of Technology, pp.3-8. Retrieved from: <<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8290767>> [Accessed 17 April 2020].
- Idika, N. and Mathur, A., 2007. *An Illustration Of Why Signature-Based Detection Is Insufficient.* [image] Retrieved from: <<https://pdfs.semanticscholar.org/888c/b0ac77e16ec98c8e2a5af79e567d8a43dcd1.pdf>> [Accessed 4 April 2020].
- Islam, R., Tian, R., Batten, L. and Versteeg, S., 2013. Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, [online] 36(2), pp.646656. Retrieved from: <<https://researchoutput.csu.edu.au/en/publications/classification-of-malwarebased-on-integrated-static-and-dynamic->>.
- Kumari, R. and Kr., S. (2017). Machine Learning: A Review on Binary Classification. *International Journal of Computer Applications*, 160(7), pp.11-15.
- Mohaisen, A., Alrawi, O. and Mohaisen, M. (2015). AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security*, 52, pp.251-266. Retrieved from: <https://doi.org/10.1016/j.cose.2015.04.001>
- Montana State University (2005). Operating System "Protection Rings" Diagram. Retrieved from: <https://www.cs.montana.edu/courses/spring2005/518/Hypertextbook/jim/media/intelPrivilege Levels.gif> [Accessed 10 Dec. 2019].
- Moser, A., Kruegel, C. and Kirda, E., 2007. Limits of Static Analysis for Malware Detection. *TwentyThird Annual Computer Security Applications Conference (ACSAC 2007)*, [online] pp.1-7. Retrieved from: <<https://ieeexplore.ieee.org/document/4413008>> [Accessed 14 April 2020].
- Nataraj, L., Karthikeyan, S., Jacob, G. and Manjunath, B., 2011. Malware images. *Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec '11*, [online] Retrieved from: <<https://dl.acm.org/citation.cfm?id=2016904.2016908>> [Accessed 18 March 2020].

Nominet (2017). TECHNICAL WHITEPAPER - TRACKING THE WANNACRY

RANSOMWARE. Nominet, p.3. Retrieved from:

<https://media.nominet.uk/wpcontent/uploads/2017/11/21123044/WannaCry-Whitepaper1.pdf>.

Raman, K., 2012. *Selecting Features To Classify Malware*. [ebook] San Francisco, pp.1-4. Retrieved from:

<<https://pdfs.semanticscholar.org/3100/cab4391d933bd7cce4047ce9e67cb1960750.pdf>>

[Accessed 26 April 2020].

Ray, S., 2017. *Three Generated Hyper-Planes To Classify Two Shapes*. [image] Retrieved from:

<https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_21.png> [Accessed 17 March 2020].

Ronaghan, S., 2018. *Random Forest Feature Generation Within Scikit-Learn*. [image] Retrieved from:

<https://miro.medium.com/max/1400/1*oar13be_cUsLR35MA_t6WQ.png> [Accessed 23 April

2020].

SANS Institute, 2003. *Input, Output And Properties Of Hash Functions*. [image] Retrieved from:

<<https://www.giac.org/paper/gsec/3294/study-hash-functions-cryptography/105433>> [Accessed 4 April

2020].

Schultz, M., Eskin, E., Zadok, E. and Stolfo, S. (2000). Data mining methods for detection of new malicious executables. Oakland, CA, USA: IEEE, pp.1-4. Retrieved from:

<https://doi.org/10.1109/SECPRI.2001.924286>.

Selamat, N., Mohd Ali, F. and Abu Othman, N. (2016). Polymorphic Malware

Sharma, A. and K. Sahay, S., 2014. Evolution and Detection of Polymorphic and Metamorphic

Malwares: A Survey. *International Journal of Computer Applications*, [online] 90(2), pp.7-11.

Retrieved from: <<https://research.ijcaonline.org/volume90/number2/pxc3894098.pdf>> [Accessed 23 March

2020]. Netmarketshare (2019). Operating System Market Share. Retrieved from

<https://netmarketshare.com/operating-system-market-share.aspx>

Sophos Community. 2020. *Ransomware: How An Attack Works - Sophos Community*. [online] Retrieved

from: <<https://community.sophos.com/kb/en-us/124699>> [Accessed 2 April 2020].

Symantec (2017). ISTR Ransomware 2017. Insider Threat Report. Mountain View, CA, USA: Symantec, pp.6-8. Retrieved from: <https://www.symantec.com/content/dam/symantec/docs/securitycenter/white-papers/istransomware-2017-en.pdf>.

Tesauro, G., Kephart, J. and Sorkin, G. (1996). Neural networks for computer virus recognition. IEEE Expert, 11(4), pp.5-6. Retrieved from: <https://pdfs.semanticscholar.org/1b6a/47119f7fea9229f786684f9e83c1441c2075.pdf>

Windows Developer Center. (2020). PE Format - Win32 apps. Retrieved from: <https://docs.microsoft.com/en-us/windows/win32/debug/pe-format>

Zhuge, J., Holz, T., Han, X., Song, C. and Zou, W. (2006). Collecting Autonomous Spreading Malware Using High-Interaction Honeypots. Retrieved from: <http://www.cs.ucr.edu/~csong/icics07.pdf>

12. Appendix A: Dataset Statistics

Table 5: Dataset Distribution

Dataset	File Count	File Makeup
Clean	92	Windows PE Structured Software Installers
Malicious Files	92	Windows PE Structured Collected Malware Samples

Unnamed: 0	AddressOfEntryPoint	DebugRVA	DebugSize	Dll	ExportRVA	ExportSize	IATRVA	ImageBase	ImageVersion
0	0129086ae5fa2269d1037ff0ac0fca48	70427	117040	28 320	130688	358	139804	268435456	0
1	01404bb45219113d64965f76803deddb	4585	0	0 0	8592	72	8192	268435456	0
2	014ae77d9d8126e805c49d8682b8bb37	4585	0	0 0	8592	72	8192	268435456	0
3	017f63d0be693e53bc5b8edd426cfbd1	4585	0	0 0	8592	72	8192	268435456	0
4	01bdc6fb077098f4a3b60f4b0e479a7f	4585	0	0 0	8592	72	8192	268435456	0
5	02c5f1515bf42798728fac17bfe1e4c1	70407	117040	28 320	0	0	139732	268435456	0
6	044a104a6de4833a707096f1e9cb7588	4585	0	0 0	8592	72	8192	268435456	0
7	07f48bef6b0cd45e85fe7b9258cbf411	4585	0	0 0	8592	72	8192	268435456	0
8	0838aaa5c0510d2cef7178eb21774fc5	4585	0	0 0	8592	72	8192	268435456	0
9	0ab2aeda90221832167e5127332dd702	4585	0	0 0	8592	72	8192	268435456	0

10 rows x 21 columns

Figure 52: First 10 files of the "malware" dataset in Jupyter Notebook

Unnamed: 0	AddressOfEntryPoint	DebugRVA	DebugSize	Dll	ExportRVA	ExportSize	IATRVA	ImageBase
0	00070b0d4cb037c40d5d2464f92841aeb9ad863472bf95...	21704.0	4880.0	28.0 0.0	0.0	0.0	4096.0	4.194304e+06
1	00696555cbf6db83af785f8acb2270b9411cfc75e7f6d3...	29424.0	4256.0	28.0 49472.0	32576.0	163.0	36864.0	4.194304e+06
2	007247436f041ca59c5ee0e8636c668c2a43376aeb8cfa...	227872.0	82400.0	84.0 49472.0	0.0	0.0	3522560.0	4.194304e+06
3	007bdab757d03d94e60c9b1e3eec13b07562705c514992...	10656.0	4320.0	28.0 49472.0	0.0	0.0	20480.0	4.194304e+06
4	008fa2b9697f9a173e40572face100410e51975e34a5ce...	152696.0	4224.0	28.0 33120.0	0.0	0.0	200704.0	5.368709e+09
5	00fa0679d4d159772298f2fafc2d349620264a592f900a...	4755.0	12576.0	84.0 33088.0	0.0	0.0	12288.0	4.865393e+08
6	012426c78cc11ace9c6789be4da0fbbd2cfd7cddc4df3...	188842.0	422864.0	84.0 33088.0	0.0	0.0	307200.0	4.194304e+06
7	01579b7ac743f18645c1ac3ca8b3dbe0de0e20b7fbb98e...	25776.0	4320.0	56.0 49472.0	0.0	0.0	172032.0	4.194304e+06
8	01ae2f2e549cda25720972cac5fffb27eda2255fb67d0c...	12707.0	0.0	0.0 34112.0	0.0	0.0	32768.0	4.194304e+06
9	01b38fd2c1dff5dafdaaff096fb5d4f93697428afcb98ef...	21200.0	4368.0	28.0 49472.0	0.0	0.0	28672.0	4.194304e+06

10 rows x 2699 columns

Figure 53: First 10 files of the "clean" dataset in Jupyter Notebook