FeSAD: Ransomware Detection with Machine learning using Adaption to Concept Drift

School of Science and Technology, City, Department of Computer Science ,University of London.

Damien Warren Adrian Fernando

06-04-2023

Acknowledgements

Firstly, I would like to thank my supervisor Dr Nikos Komninos, for giving me the opportunity to do a PhD and guiding me through this process. The principles I learned while working with Dr Komninos helped me immensely in research and life, and I will forever be grateful. I want to thank Dr Thomas Chen, who was my second supervisor and helped direct aspects of my research and publications. I would also like to thank the computer science department at City, University of London, where I did my undergraduate studies, masters and PhD. The university has invested in me, and I will never forget the opportunities the organisation has given me. I give immense thanks to my wife, Samantha, who has supported me through this process and witnessed the highs and lows I experienced while doing my PhD. I thank my mother, father and siblings, who helped me on this journey and Robin, who kept me company during the long days of research and running experiments. Finally, I thank God for guiding me on this journey and bringing me this far.

Contents

List of Figures

List of Tables

Abstract

Ransomware classification is crucial, and the main issue with ransomware is that misclassification can have devastating effects compromising valuable data and causing significant monetary loss to organisations. In addition to damaging businesses and individuals, ransomware is a malware type that is evolving rapidly, with new families and variants constantly appearing; this can lead to misclassifications by detection systems. Modern detection systems have moved away from heuristic detection methods and use more flexible approaches such as machine learning. Machine learning is an effective way to detect malware; therefore works well when detecting ransomware. Concept drift occurs in machine learning systems; this implies that the statistical properties of the target variable have changed either suddenly or over time. Concept drift is a notable weakness of a machine-learning malware detection system. Concept drift suggests the machine learning algorithm's rules and principles have become outdated; this phenomenon can represent ransomware evolution. The concept drift phenomenon represented by ransomware evolution presents a significant challenge for Machine Learning intrusion detection systems because of the inevitable degradation of classification models. This thesis proposes FeSAD, a ransomware detection framework designed to counteract the concept drift in ransomware; this is achieved by combining statistical properties of ransomware and benign data with feedback from the classifier to make a reliable classification under concept drift. The FeSAD framework has a feature selection algorithm for systems expected to have concept drift. In addition, there are drift detection and adaptation components to deal with concept drift. The feature selection layer is a proactive solution that generates feature sets that will remain robust over time, and the drift layer is a reactive measure that allows a detection system to reliably and accurately classify samples that show concept drift. The FeSAD framework is designed to work with most machine learning algorithms and is tested under various concept drift scenarios with ransomware and benign files from different distributions; each distribution is defined by the year of release. The FeSAD framework was tested with random forests, multi-layer perceptrons and a Bayesian network and achieved strong results by maintaining a detection rate close to 90% in all concept drift scenarios. The FeSAD framework's strong detection results under concept drift also show that it prolongs the lifespan of a machine learning classifier by maintaining a high detection rate across different ransomware distributions.

Chapter 1

Introduction

Ransomware is a significant threat in the cyber security space, and its attack methodology ensures its impact can be damaging and lasting. Ransomware encryption techniques ensure its victims cannot decrypt their data unless the attacker has made an error; encryption scheme complexity allows attackers to leverage victims, especially if they do not have sufficient backups. In most cases, even if an organisation or individual has backups, the disruption caused by a ransomware attack will cause delays and financial damage for businesses. Ransomware is a malware designed to prevent or reduce user access to their system or files. Ransomware will come in the forms of Locker Ransomware and Crypto Ransomware. Locker ransomware will display a lock screen that prevents the victim from accessing their computers, often acting as law enforcement demanding monetary payment in return for access to the computer. Crypto Ransomware will encrypt critical files on a user's system using complex encryption schemes and demand fees, usually on cryptocurrency, to decrypt the victim's files. In its history, ransomware has become more prominent, advanced and destructive. The rise of ransomware has been attributed to many different factors since it first appeared in 1989. The emergence of ransomware as a service has also increased the availability of ransomware to potential criminals who are less technically gifted. CryptoLocker, CryptoWall and Locky, offer this type of service with the variant CryptoWall alone, generating more than 320 million dollars in revenue during its lifespan (De Groot, 2017). The heavy expansion of Crypto ransomware from 2014 onwards would have many variants going on to make hundreds of millions of dollars in ransoms. Reports from early 2017 indicate total damages and profits from ransomware reaching the 1 billion dollar mark (De Groot, 2017). The use of machine learning has become more prominent in the field of ransomware research; the application of it in general malware detection has also risen to prominence due to the unreliability of the traditional signature-based approaches and the rapid evolution of malware. With malware often evolving and new versions of malware families behaving differently from their predecessors, traditional approaches will find it more and more difficult to detect them. Baig et al. outline malware creators' methods to evade traditional static detection methods. Writers of malware can use simple methods like modification of packed executables to evade detection (Hwang et al., 2020).

1.1 Research Motivation

Inspecting the current landscape of ransomware and malware detection, as well as concept drift research for malware detection, it appears there is a gap in the research for ransomware detection and concept drift adaption. Ransomware is a rapidly evolving malware type, one of the most dangerous digital threats to individuals and businesses. There are currently many ransomware studies and research papers available. The research into ransomware focuses on detecting and identifying zero-day ransomware threats; however, the research seems to operate on the assumption that zero-day ransomware will not behave in a way that could differ from current ransomware. Current ransomware research also attempts to classify ransomware into their respective families and explore how ransomware evades detection. Ransomware evading detection is explored by Olaimat et al. and shows the different layers of difficulty in ransomware detection and how ransomware attempts to mask itself.

We note through our review of existing research that no ransomware detection systems focus on evolution and drift at every level. The types of concept drift discussed in the research background section include feature change; therefore, features need to be robust to avoid becoming outdated with a changing concept. The feature selection processes we have reviewed do not focus on creating robust features that can last beyond the concept it is trained on. Due to the rapid evolution of ransomware, the features a classifier uses can quickly become outdated. As concepts shift, the features of the role play in identifying ransomware can also shift; features that were once important can become unimportant; therefore, it is essential to choose features that have the best chance of staying effective over time.

A large proportion of ransomware research does not consider concept drift; however, zero-day detection is emphasised. FeSA highlights the difference between zero-day ransomware and ransomware that has evolved and shows concept drift behaviour; ransomware being zero-day can be detected by a machine-learning detection system if its behaviour is in line with what the machine-learning classifier expects. Ransomware becomes evolutionary and shows concept drift when it behaves differently from what a classifier expects (Fernando & Komninos, 2022). Ransomware detection that considers concept drift, such as the research proposed by Urooj et al., uses concept drift detection but relies on retraining to adapt to concept drift. The Transcend and Transcendant system proposed by Jordaney et al. and Barbero et al. uses multi-layered statistical approaches to define class boundaries and concept drift detection methods. The Transcend and Transcendant system is built to be feature-set and algorithm-agnostic to adapt to different domains and feature types. The Transcendent system is built from the Transcend system and is a real-world implementation of the proof of concept Transcend system and is proven to be adaptable and robust to different feature types and algorithms when detecting and rejecting samples showing adverse concept drift. The Transcend and Transcendant solutions offer strong and reliable results when detecting concept drift; however, there is no adaption to concept drift, and it relies on human interaction and potential retraining for a classifier to adapt to concept drift.

The studies for general concept drift propose various detection metrics for concept drift that could be applied to malware or ransomware detection; however, there is a lack of research that trains machine learning algorithms to be robust when faced with concept drift. There is also a lack of research into what lengthens the lifespan of machine learning algorithms while expecting concept drift. Expanding the lifespan of a classifier helps avoid the need to retrain constantly. While retraining will always be needed eventually, it is a costly process that involves the creation of new datasets and the time and computational cost associated with retraining a model. In an environment that presents evolving malware often, the computational and time cost of retraining often is high, especially if that is the only defence against concept drift. Current ransomware and malware solutions to concept drift involve continuous retraining once drift is detected; however, the retraining process could occur too late or once drifting samples have been misclassified. There is too much at stake in the ransomware space to allow misclassification due to concept drift. Taking the shortcomings of the current ransomware detection research into account, it is clear that there needs to be a system designed with concept drift in mind at every level; this includes feature selection, detection, adaption and retraining.

1.2 Research Background

Ransomware is a malware type that restricts a user's access to their files by locking a system or encrypting user files. Crypto-ransomware restricts a user's access by encrypting files, and locker ransomware will lock a system to restrict a user's access to files. Our research focuses on Crypto-Ransomware, the most common and potent ransomware (Al-rimy et al., 2018). Ransomware leverages users to provide attackers with monetary payment for the decryption keys to their files (Richardson & North ., 2017). Payments for ransomware infections are usually in cryptocurrencies to maximise the anonymity of attacks. Ransomware attacks create a power imbalance between the attacker and the victim due to the attacker not having an obligation to pay (Urooj et al., 2021). When attackers do not pay, a user is left with the consequences of paying a ransom and not getting access to their files. Attackers may often be motivated to stick to their word of decrypting for payment to encourage other victims to pay, as ransomware attacks can often target multiple users and organisations (Olaimat et al., 2021).

Ransomware has seen a rapid evolution in the last decade; however, its roots can be traced back to 1989 with the AIDS trojan; Joseph Popp distributed floppy disks that would scramble and encrypt files names and would ask victims for a licensing fee (Waddell, 2021). The AIDS Trojan was primitive but created the concept that would cause hundreds of millions of dollars of damage across the world in the future. The first large-scale ransomware infection was CryptoWall (De Groot, 2017) which infected 250,000 PC systems worldwide; this ransomware variant generated $3 million over its lifespan. CryptoWall used the ZeuS Botnet to propagate itself. The CryptoWall operation was shut down in 2014 by multiple government agencies across continents. Since its initial outbreak, multiple variants of CryptoWall have generated millions in revenue. The CryptoWall outbreak was a sign that ransomware could be used to launch large-scale malware attacks to generate large amounts of money in the form of ransoms. The WannaCry attack in 2017 is another example of a large-scale ransomware attack that spread worldwide. WannaCry attacked 400,000 machines worldwide and caused severe disruption to the NHS in the UK (Brandon, 2017). The attack was halted when a security researcher could purchase a domain hidden in the malware's source code that terminated it as soon as the domain was active. The attack on the NHS caused damage to hospitals and GPs, with systems being down and patient records being inaccessible. The WannaCry attack targeted out-of-date Windows XP installations and used a flaw in the Microsoft Server Message Block to propagate. Despite the attack, the NHS was using thousands of machines running Windows XP as recently as 2020 (Ten, 2021). Using out-of-date operating systems will continue to create opportunities for attackers to exploit. The NHS cites financial and technical constraints when justifying the continued use of Windows XP (Ten, 2021). The cost of upgrading to Windows across the board is significant for an organisation, and the cost of replacing machines that would not be able to support the upgrade to Windows 10 also needs to be taken into account.

Overall, the damages caused by ransomware passed the $1 billion mark in 2017 and have significantly accelerated. Cyber Reason reports the ten most significant ransomware attacks of 2021, and the damages caused by these attacks surpass the global WannaCry attacks; this shows how potent a threat ransomware is. The 2021 Darkside ransomware attack on the colonial pipeline caused significant disruption when the Darkside ransomware gang attacked the IT systems of the Colonial Pipeline Company; this attack caused significant disruption to a company responsible for transporting over 100 million gallons of oil between New York and Texas (Winder, 2020). The Kaseya ransomware attack leveraged a vulnerability in the Kaseya VSA software to launch an attack on users of this software and their downstream clients (Winder, 2020). Kaseya did not directly pay the ransom; however, this attack caused significant damage to the company and their clients. The Kaseya VSA software had multiple zero-day vulnerabilities that were not addressed despite Kaseya being alerted to this by the Dutch Institute for Vulnerability Disclosure (DIVD). The Sodinokibi ransomware attack on the electronics corporation Acer caused not only disruption but also caused the leak of hundreds of sensitive documents related to the company's financials. The Sodinokibi attack showed that ransomware attacks could also lead to data breaches, and using confidential data can also leverage victims into paying the ransom demanded.

Having discussed the impact of ransomware attacks, it is clear that ransomware can cause severe damage to individuals and businesses. Ransomware can propagate in different ways, and how ransomware propagates determines how scalable the attack is. The popularisation of ransomware as a service (RaaS) has expanded the number of attackers with access to ransomware (Gomez-Hernandez, 2018). Raas operates as a business; ransomware creators will sell the code for their ransomware and allow their clients to use their infrastructure, such as their Command and Control servers, to launch ransomware attacks (Gomez-Hernandez, 2018). The RaaS as a model will allow clients to use code but will take a cut of the profits; therefore, the creators of the RaaS malware will be able to make income through the infections propagated by their clients passively. The fees for RaaS code can vary from as little as \$39 for the Stampado ransomware to over \$1000 for access to ransomware like Locky. RaaS has expanded the ransomware space because it allows users with little technical knowledge to use ransomware and make money with it.

Ransomware can be propagated through many vectors; this makes the detection challenge even more significant as there are multiple patterns of behaviour to look for to isolate ransomware attacks. The most common infection vectors include spam or phishing emails that trick victims into opening malicious files (Zimba, 2017). Drive-by downloads from compromised web pages allow attackers to execute malicious code on the victim's machine without the victim's consent. Ransomware infection vectors can be more specific than common attacks such as phishing, drive-by downloads or exploit kits; in the case of ransomware like WannaCry or Kaseya, the attack vectors were specialised to the victim and targeted the victim, knowing exactly where the weaknesses were. Kaseya's VSA vulnerabilities or the SMB protocol vulnerability in Windows machines were zero-day exploits, and attackers could leverage these exploits to attack high-value targets. Ransomware infections have also gone beyond leveraging users' files to get ransom payments.

Attackers have started using a technique known as double extortion (Kerner, 2021), which involves encrypting data and exfiltrating stolen data to a separate location. After the exfiltration of stolen data, the attacker will leverage the victim with the prospect of releasing sensitive data. The Maze and REvil ransomware strains have been known to deploy double extortion tactics aggressively (Kerner, 2021). A double extorsion attack crippled foreign exchange firm Travelex in 2020 with the encryption of over 5GB of data and the publication of sensitive customer information; this attack cost the firm nearly 2000 jobs, and Travelex was forced into administration seven months after the attack (Fearn, 2020). Looking at the patterns of attack we have seen thus far; it is clear that ransomware has many vectors of attack, and the use of zero-day threats is prevalent in ransomware attacks. Ransomware using zero-day attack vectors creates a daunting challenge for businesses, as simply having systems patched and up to date may not be enough to avoid being the victim of a ransomware attack.

Ransomware has also shown rapid evolution since its initial appearance in 1989 and shows no signs of slowing down. New ransomware strains are detected daily, with SonicWall reporting up to

2500 unique ones detected yearly and up to 38 new ransomware strains in 2018 (SonicWall, 2018). 2021 saw ransomware attack attempts almost double, with 623 million attacks, compared to 304 million in 2020 (SonicWall, 2020). In 2021 SonicWall recorded over 1000 new ransomware variants, with the most dominant variant being Ryuk; the massive volume of attacks shows the spread and popularity of ransomware increasing rapidly, with a variation in targets ranging between businesses and individuals. The ransomware attack methodology has evolved, with triple extorsion attacks being carried out (SonicWall, 2020). A Finnish psychotherapy company, Vestaamo, observed the first known triple extortion attack with an attacker using the name ransomman using a double extortion attack to leverage the company into paying a 40 bitcoin ransom to avoid the publishing of sensitive data and decryption of their files. The attacker also used the privileged information stolen to attempt to leverage clients of Vestaamo into paying fees of $240 to avoid their individual patient information being leaked; this made this a triple extortion attack.

1.2.1 Ransomware Detection with Machine Learning Algorithms

Judging by the state of the ransomware horizon, it is clear that significant challenges are presented to organisations to avoid ransomware infections. The constant evolution of ransomware and the use of zero-day vulnerabilities means the detection of ransomware must be accurate and proactive. There is a wealth of research in ransomware detection, and the most forward-thinking approaches appear to be machine learning or deep learning-based approaches because of their ability to detect ransomware that the classifier has not seen before in training. The machine learning approach eliminates the reliance on heuristics and signatures and attempts to learn the behaviour of ransomware to detect ransomware that has not been seen before (Fernando et al., 2020). Sgandurra et al. present an early detection system for ransomware that uses API calls and Logistic Regression to detect a ransomware infection. The EldeRan system (Sgandurra et al., 2016) uses API calls to monitor ransomware behaviour; this system can monitor ransomware during the earliest stages of execution and analyse how the ransomware sample interacts with the Operating System. Urooj & Maarof present an early detection system that attempts to detect ransomware in its pre-encryption phase; the detection algorithm used for this system is Stochastic Gradient Descent. Urooj et al. attempt to ransomware population drift as ransomware is a fast-evolving malware type; using cryptographic APIs serves as a flag for ransomware about to start encryption (Urooj & Maarof, 2021) and constantly retrains to adapt to new ransomware.

The Ransomwall system (Shaukat & Ribeiro, 2018) uses a layered approach with a machine learning detection engine using Gradient Tree Boosting; Ransomwall's layered approach includes feeding data from the different defence layers into the machine learning classifier. The RansHunt system uses SVM to detect ransomware with static and dynamic data taken from honeypots (Hasan & Rahman, 2017). RansHunt uses the data gathered for training to predict futuristic ransomware behaviour and has a kernel tuned for ransomware. Daku et al. propose a ransomware detection system that uses j48 decision trees (Daku et al., 2018) with dynamic behavioural features but with no focus on ransomware evolution or population drift. Takeuchi et al. propose an SVM-based approach for detecting ransomware; the classifier uses API calls for features involving the ransomware's execution (Takeuchi et al., 2018). The resilient machine-learning approach proposed by Chen et al. attempts to evaluate the resilience of machine-learning detection systems using adversarial learning (Chen et al., 2019). Seong et al. propose using API call sequences combined with multiple machine learning algorithms (Seong et al., 2019); this system uses an approach which uses the frequency of feature appearance to distinguish between malware and benign files. The work proposed by Zuhair et al. focuses on detecting zero-day ransomware strains and tracing them back to their ancestral strains (Zuhair et al., 2020). The approach proposed by Zuhair et al. stands out because they focus on detecting zero-day ransomware and identifying the ancestors of the zero-day ransomware.

Manavi & Hamzeh propose an LSTM (Long Short Term Memory) with static features to detect ransomware (Manavi & Hamzeh, 2021). LSTMs are a form of neural network; however, using static features alone could pose an issue because of obfuscation techniques attackers can deploy. Baek et al. propose an SSD-based ransomware detection system that monitors I/O patterns in SSDs to detect ransomware. Baek et al. use I/O patterns and update after read I/O flags to detect ransomware. Dual Generative Adversarial Networks (Zhang et al., 2021) use GAN adversarial networks to generate pseudo ransomware samples to strengthen a classifier. The adversarial networks help train a ransomware classifier and allow it to distinguish between actual and pseudo ransomware. Chakkaravarthy et al. propose an IoT ransomware detection system with a Social Leopard algorithm; this research focuses on IoT devices which is a necessary avenue of research because of the rapid expansion of IoT (Sibi Chakkravarthy et al., 2020).

1.2.2 Feature Selection

Feature selection algorithms are designed to identify valuable features in a feature set and disregard useless features in a feature set; this can be done in various ways (Peker et al. 2015). The majority of ransomware detection research uses a form of feature selection to optimise the feature set used for detection. Concept drift can alter the significance of particular features making once-important features and vice-versa; therefore, it is key to experiment with feature selection and identify how to construct feature sets that can be resilient to concept drift. A genetic algorithm is a search heuristic that takes Charles Darwin's natural evolution theory (Fatima et al., 2019). This algorithm mimics the process of natural selection, which will select the strongest to survive and produce offspring. A genetic algorithm will apply this logic to a dataset and can be used to produce an optimal feature set. The system proposed by Vivekanan et al. uses a genetic algorithm to produce an optimal feature set for malware detection. A typical genetic algorithm will repeat its evaluation and crossover phase, creating numerous features to obtain optimal features. This research uses a genetic engineering approach to reduce the number of generations and features needed to produce the optimal feature set, otherwise known as a feature set. The feature selection process is critical as it increases a classifier's efficiency while reducing complexity and dimensionality. The EACD system proposes a genetic algorithm approach to combatting concept drift (Ghomeshi et al.). This evolutionary algorithm is multi-layered with a base and a genetic layer; both layers act as a natural selection mechanism to find the most robust feature set. Feature sets produced by Genetic algorithms are proven to be resistant to drift and able to adapt quickly to new concepts; however, they are not immune to being affected by concept drift therefore, they would make a useful component of a detection system for drift but would need a supplemental component to ensure adaptability to evolution and concept drift.

1.2.3 Concept Drift

Concept drift is a phenomenon in machine learning classifiers where the relationship between a sample's attributes and classification changes over time (Gao et al., 2007). Concept drift is formally defined as a change in $P(x, y)$. The components of concept drift are $P(x)$, the probability component is the probability of x, and class label conditional probability $P(y|x)$, is the probability of y given x; the change of the joint probability can be better understood via the changes in either of these two components (Gao et al., 2007).

The system proposed by Urooj et al. factors in population drift and addresses it by constant retraining when a drifting sample is detected. The Transcend System proposed is a framework that can work with any machine learning algorithm to output confidence values for predictions (Jordaney et al., 2017). The transcend system attempts to identify drifting samples and will retrain a classifier with drifting samples to increase the detection rate. The transcend system relies on human intervention when analysing drifting samples. Kantchelian et al. combine human intervention with machine algorithms to address concept drift in an adversarial machine-learning scenario; this system tests adversarial learning as an evolutionary family of the training dataset. Web application concept drift is addressed by Maggi et al. and uses anomaly detection to distinguish between genuine changes in a web application and malicious changes; this system relies on retraining to adapt to concept drift and reduce false-positive rates with retraining being carried out once concept drift is detected (Maggi et al., 2009). Singhal et al. use the Heterogeneous Euclidean Overlap Metric (HEOM) to detect concept drift in detecting malicious web URLs. The system combines Gradient Boosted Trees with the HEOM measurement. The concept drifts detection component of the system proposed by Singhal et al. measures the distances between the data distribution between the old training data and the new incoming data. The distance between the training set and the newer data is calculated using the HEOM. The malicious URL detection system attempts to detect concept drift in malicious URL detection systems and uses the Wilcoxon Rank-Sum test. The Wilcoxon Rank-Sum test can identify samples from different distributions (Tan et al., 2013). Barbero et al. propose Transcendent, which builds on the Transcend system to improve detection rates of concept drift samples; however, this approach still relies on retraining to adapt to concept drift. Distances between anomalous samples can be used to detect drift (Baena-Garcia et al., 2006). EDDM (Early Drift Detection Method) and DDM (Drift Detection Method) use the distance between anomalous samples to measure drift; the closer anomalous samples are in a time frame, the worse drift is. RDDM attempts to reduce performance loss due to EDDM retaining old data that represents out-of-date concepts (Barros et al., 2017). ECDD, proposed by Ross et al., uses exponential moving averages combined with mean and standard deviations to detect drift (Ross et al., 2012).

Inspecting the current landscape of ransomware and malware detection, as well as concept drift research for malware detection, it appears there is a gap in the research for ransomware detection and concept drift adaption. A large proportion of ransomware research does not consider concept drift despite looking at zero-day detection; this point is crucial as not all zero-day samples will show behaviour that reflects concept drift, and it is important to distinguish between the two. The FeSA system highlights the difference between zero-day ransomware and ransomware that has evolved and shows concept drift behaviour; ransomware being zero-day can be detected by a machine-learning detection system if its behaviour is in line with the machine learning classifier expects (Fernando & Komninos, 2022). Ransomware becomes evolutionary and shows concept drift when it behaves differently from what a classifier expects. Ransomware detection that considers concept drift, such as the research proposed by Urooj et al., uses concept drift detection but relies on retraining to adapt to concept drift. The Transcend and Transcendant system proposed by Jordaney et al. and Barbero et al. uses multi-layered statistical approaches to define class boundaries and concept drift detection methods; however, their solutions to concept drift adaption are human interaction and potential retraining. The studies for general concept drift propose various detection metrics for concept drift that could be applied to malware or ransomware detection; however, there is a lack of research that trains machine learning algorithms to be robust when faced with concept drift. There is also a lack of research that lengthens the lifespan of machine learning algorithms in concept drift scenarios.

We define a machine learning algorithm as fit for purpose as long as most samples classified benign or ransomware does not show abnormal levels of drift. We define the lifespan of a machine learning algorithm as the period between training and the threshold of abnormal drifting samples being exceeded. The drift calibration layer defines the abnormal drift threshold, and the drift of every incoming test sample is recorded. Once the number of samples showing an abnormal drift exceeds the set threshold, the algorithm is considered at the end of its lifespan, and the machine-learning algorithm must be retrained. A high number of samples showing abnormal drift would indicate that the machine learning algorithm is becoming less confident of its classifications and that a high proportion of samples' statistical properties are different from expected. The FeSAD framework aims to increase the lifespan of a classifier by preparing it to classify samples that show drift and calibrating its drift thresholds to account for the drift commonly seen from ransomware families from different distributions. The key difference between the current research and the FeSAD framework is that the FeSAD framework aims to automate the process of dealing with drifting samples instead of requiring intervention from operators despite eventual retraining. Additionally, the FeSAD framework attempts to prepare the system for incoming drift by anticipating drift based on past drift values. The current work, which provides ways to detect drift, still requires retraining and will not prepare the system for future drift; rather, the current work provides a way for the system to adapt to drift occurring presently instead of preparing for future drift.

1.3 Overview of Approach

This research considers the limitations of ransomware detection and proposes a system that considers concept drift. Ransomwall addresses the issues with ransomware detection on multiple fronts (Shaukat & Ribeiro, 2018). Firstly, the system will build a feature set that can stay robust over time and presents the best chance for a machine learning classifier to remain relevant for as long as possible. In addition to the robust feature set, this research proposes a system that will factor concept drift into its training process and prepare a machine learning detection system for future concept drift. The proposed system will be able to use knowledge of concept drift to reliably identify the ransomware that shows concept drift and expand the life of a machine learning classifier. The types of concept drift are broken down below. The effect of concept drift on a classifier is defined in section 1.2.3.

· **Gradual Concept Drift:** A gradual change over time.

· **Cyclical Concept Drift:** A recurring or cyclical change.

· **Abrupt Concept Drift:** A sudden or abrupt change.

1.3.1 Feature Selection Layer

The genetic algorithm has been proven effective against systems that experience concept drift by Ghomeshi et al.; however, the EACD system uses a method of retraining. The EACD system proves that the genetic algorithm provides high adaptability to concept drift. The feature selection layer will use a genetic algorithm to isolate the optimal feature set for robust ransomware detection. The genetic algorithm will be modified to enforce features that provide the most discrimination between ransomware and benign files. The idea behind the feature selection layer is to enforce a group of essential features and allow a genetic algorithm to build around these core features. The features are chosen using a combination of information gain and effectiveness in differentiating between ransomware and benign files. The feature selection layer assumes that the chosen features will be less likely to be affected by abrupt concept drift due to the nature of Windows API calls. Windows API calls are how a program communicates with an operating system and essentially provide a pattern of execution data for a program. The assumption that feature sets produced by the genetic algorithm won't be affected by abrupt concept drift is due to this, implying a complete shift in the baseline ransomware behaviour and how it interacts with the operating system. The feature selection layer is designed to be a proactive measure to lengthen the lifespan of a ransomware machine learning classifier; however, it would not be effective in an abrupt concept drift scenario or a scenario where there is a feature change, and the features chosen by the algorithm become unimportant.

1.3.2 Drift Layer

The drift layer is the adaptive aspect of this approach because concept drift can affect systems in different ways. The feature selection layer would struggle to deal with abrupt concept drift and feature changes because it operates solely within the machine learning algorithm. The drift layer attempts to adapt and react to changes in the concept. The drift layer takes information from the classifier and combines it with statistical information from the data to judge whether a sample is displaying concept drift. The drift layer's purpose is to allow the classifier to make reliable and accurate predictions for as long as possible before retraining; this includes classification under concept drift. The drift layer's secondary purpose is to extend the lifespan of a classifier. The system will be trained to expect concept drift, and the drift layer will have parameters that allow it to judge what level of concept drift is normal and abnormal. The drift layer will also have parameters that allow it to judge whether a sample displays unacceptable levels of concept drift and is likely to be misclassified by the machine learning algorithm. The system can be used with almost any underlying machine learning algorithm; however, its performance depends on its features and suitability for detecting ransomware. The drift layer will alert a user once the classifier is experiencing unacceptable levels of drift and retraining is required. Ultimately, the drift layer is designed to extend the lifespan of a classifier, but it is not designed to make it immortal as this is impossible. The drift layer addresses the flaws identified in the current research and should help minimise the risk of ransomware misclassification.

1.4 Research Question, Hypothesis and Objectives

1.4.1 Research Question

How can we improve the efficiency and accuracy of a machine learning-based ransomware detection system when ransomware evolves over time?

1.5 Research Aim and Individual Objectives

· To review the most recent and cited studies for ransomware detection.

· To review relevant studies on concept drift in data, in malware detection.

· To propose a new framework for ransomware detection which will be able to detect drift and adapt the feature set if possible (If an algorithm used allows local replacement). This framework should reduce training and only resort to completely retraining the system as a last resort.

· To find commonalities between different machine learning algorithms which will be used in order to build an aspect of the framework which will allow the algorithm to be adapted instead of retraining.

1.5.1 Research Hypotheses

To achieve the research objectives, the following hypotheses have been defined.

· The shortcomings and challenges of current research in ransomware detection, concept drift and feature selection algorithms can be identified, and thus a solution to remediate these shortcomings can be created.

· A genetic algorithm can produce feature sets that improve machine learning algorithm performance under concept drift.

· Machine learning algorithm metrics combined with statistical and distance metrics can reduce performance degradation in machine learning-based ransomware detection systems when exposed to concept drift.

· It is possible to use concept drift when training machine learning detection systems to anticipate future ransomware evolution.

1.5.2 Research Assumptions

To achieve the research objectives, the following hypotheses have been defined.

· Machine Learning algorithms can effectively distinguish between ransomware and benign files using API calls as features.

· A genetic algorithm effectively identifies strong feature sets for machine-learning solutions.

· Ransomware behaviour evolves over time, and this evolution presents itself in machine learning detection systems as concept drift.

· Concept drift in machine learning-based ransomware detection systems will cause performance degradation.

1.6 Contribution of the Research

· Investigating the current ransomware detection space and understanding how concept drift research can be integrated into ransomware detection.

· Proposal of a feature selection algorithm that creates robust feature sets for ransomware detection under concept drift.

· Proposal of the FeSAD system, which considers concept drift during its training phase and can classify ransomware accurately and reliably under concept drift scenarios.

1.7 Thesis Outline

- **Chapter 1 Introduction:** This chapter provides a background to the research undertaken, along with the scope, aim and contribution of the research undertaken.
- **Chapter 2 Literature Review:** This chapter explores the existing research in ransomware detection, the importance of feature selection and the use of concept drift in the malware detection space.
- **Chapter 3 FeSAD System:** This chapter explores the FeSAD system and how it's feature selection works and also how it is designed to adapt to concept drift scenarios.
- **Chapter 4 Experimental Evaluation & Results:** This chapter explains the experiments conducted to test the proposed system and the results obtained.
- **Chapter 5 Conclusion & Future Work:** This chapter explores what this thesis has achieved and what these achievements will contribute to future work.

1.8 Publications

1.8.1 Published

· Fernando, D.W.; Komninos, N.; Chen, T. A Study on the Evolution of Ransomware Detection Using Machine Learning and Deep Learning Techniques. IoT 2020, 1, 551-604. https://doi.org/10.3390/iot1020030

· Damien Warren Fernando, Nikos Komninos, FeSA: Feature selection architecture for ransomware detection under concept drift, Computers & Security, Volume 116, 2022, 102659, ISSN 0167-4048. https://doi.org/10.1016/j.cose.2022.102659

1.8.2 Submitted

· D.W.Fernando, N.Komninos, FeSAD Ransomware Detection with Machine learning using Adaption to Concept Drift, IEEE Transactions of Secure and Dependable Computing.

Chapter 2

Literature Review

In this chapter, we look into ransomware, malware detection, ransomware detection systems and the challenges they face. In addition, we will review some approaches that attempt to solve the problem of concept drift in general and regarding malware. The literature review will give an idea of what currently exists in the field of ransomware detection while also outlining what is missing and how it can be built upon.

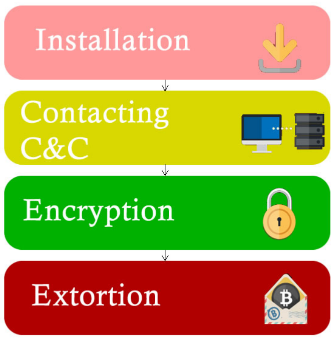

Figure 2.1: Ransomware Methodology

2.1 Ransomware Operation

Initially, it was essential to understand how ransomware works before reviewing approaches which worked to mitigate it. Ransomware is complex in its operation; understanding it is the first step to detecting and stopping it. Figure 2.1 shows the process ransomware follows during the infection of a host; this includes initial installation, downloading instructions from a C&C server, data encryption and then the victim's extortion. This process is further elaborated on in the following sub-sections.

2.1.1 Infection

Ransomware infections typically tend to be carried out by infection vectors. Firstly the most prominent vector is ***malicious emails***, the payload is delivered as an email attachment from emails sent through spam using botnets and other compromised hosts (Zimba, 2017). ***Exploit kits*** are another prominent method of infection. Exploit kits are software packages which scan a system for vulnerabilities with the intent to infect it with malicious software (CyberPedia). Another prominent method of infection is ***drive by downloads*** in which victims are lured to malicious websites which execute malicious code (Zakaria et al., 2017). The structure of a ransomware attack follows the methodology presented in Figure 2.1. the following subsections give a detailed view of what these stages are like.

Infection occurs after the payload has been dropped into the system. One prominent infection method is the download dropper methodology (Liska & Gallo, 2017). This approach uses an initial file involving a small piece of code to evade detection and reach out to the command and control centre. Ransomware authors will attempt to split execution into different scripts and processes to avoid AV signature-based detection (Liska & Gallo, 2017). When an organisation is targeted in an attack, ransomware will spread through the network, determining file share locations and infecting them to maximise disruption and increase the possible ransom. The executables will not execute until multiple machines have been infected.

2.1.2 Command and Control (C&C)

Once ransomware is installed within a system, it will reach out to a C&C centre looking for instructions (Liska & Gallo, 2017). C & C centres will respond with various requests, giving the ransomware instructions on how to proceed with the execution. Some variants of ransomware will report significant amounts of system information which can give attackers an idea of what type of system they have attacked and if it is worth going beyond just a ransomware attack depending on what kind of files are present on the system. The ransomware will reach out to the C&C centre for the encryption keys after installation to ensure the keys are kept secret (Sophos, 2016). It is almost impossible to decrypt files without the decryption keys. Command and control channels differ from ransomware family to family; some will use standard HTTP to complex Tor-based services to connect (Liska & Gallo, 2017). Tor is a browser designed to allow anonymous communication. The more complex the means of communication to reach the C&C centre, the harder it is to trace the ransomware's origins and the extortionists' base location.

2.1.3 Encryption and Extortion

Modern ransomware will utilise asymmetric encryption, so the actual ransomware will come with an RSA public key used by the ransomware to establish a secure channel to its command and control server (Taile & Patel, 2017). Public key encryption will mean the plaintext messages between the server and the client (system infected with ransomware) will be encrypted so that third parties will find it very difficult to decrypt. The critical factor in this process is that the public key can only decrypt messages encrypted by the corresponding private key. This private key is held on the server only the attackers can access, making it impossible for the victim to retrieve it. Different variants of ransomware will encrypt files on the system in different ways. Some will use symmetric encryption methods; some will use asymmetric encryption methods. Symmetric methods will generate a symmetric key locally and encrypt files using this key; the advantage is the lack of performance overhead, reducing the chances of being detected. Asymmetric keys will use a public key which can encrypt, but the decryption process requires the corresponding private key, which is only stored on the C&C server. Once a system has been locked or files have been encrypted, the ransomware will kill tasks and processes that may interfere with its execution; in some cases, install Tor or an alternative specific payment method. Crypto ransomware will seek and delete backups, i.e. shadow volume copies and system restore points. After an infected system, the ransomware will request payment of a specified version of cryptocurrency or a prepaid voucher to either unlock the victim's system or provide a decryption key.

2.1.4 Ransomware Threat Model

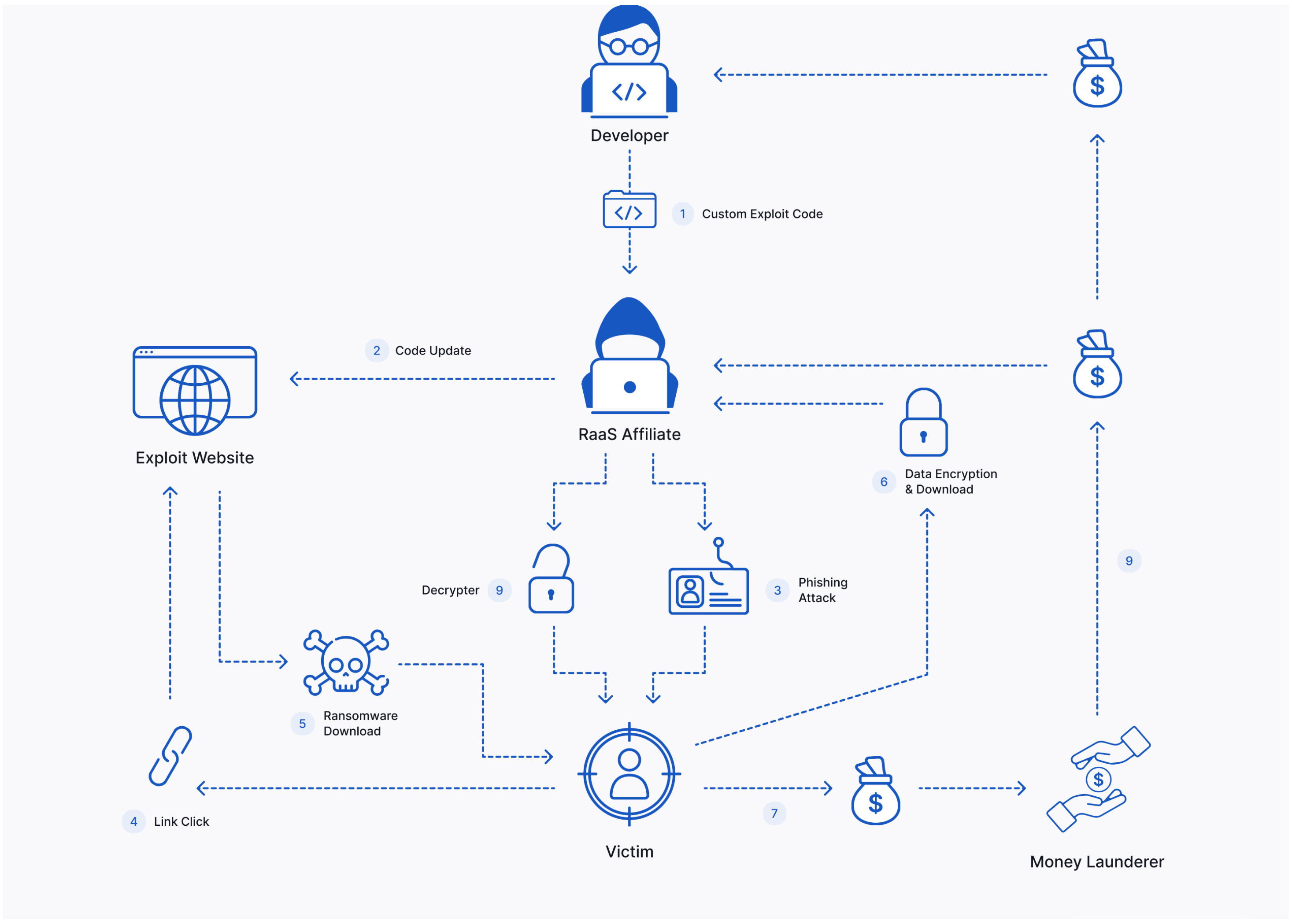

Figure 2.2: Ransomware Threat Model

Figure 2.2 shows the ransomware threat Model for RaaS (Ransomware as a Service); this model demonstrates the ransomware threat and its operating procedure in a very effective and common model for ransomware operations (Kost., 2021). RaaS is attributed to ransomware's popularity and rapid spread over the last as it allows anyone to sign up as an affiliate to use ransomware with a healthy profit split between the affiliate and the developers.

2.1.5 Ransomware Detection challenges

Regarding ransomware, various security challenges will present themselves. Firstly, a ransomware infection must be detected because once files are encrypted, it will be almost impossible to decrypt them without either paying for a decryption key or relying on the fact that the ransomware developers will have made an error or decided to make the keys available. The fact that ransomware is propagated through various methods means that early detection of it will have to consider the different propagation methods that ransomware is likely to use. A more significant challenge in the grand scheme of ransomware evolution is the noticeable concept drift in ransomware, which will be elaborated on in this paper. With a noticeable concept drift between ransomware from 2 to 3 years ago and modern ransomware, creating models that do not have to be constantly retrained will prove to be a significant challenge. In addition, the risk that new unknown ransomware that does not match the trained model can slip through the model poses a significant threat, considering how destructive ransomware infections can be. The prediction of ransomware evolution to integrate into models will also be a challenge which will prove critical in terms of the detection of ransomware going forward. Incorporating AI into ransomware attacks (Olenick, 2018)provides the most significant challenge; with attackers using similar AI-based techniques to those employed by AI-based defence systems, the configuration of defensive measures will have to consider highly adaptable AI-based attack vectors.

2.1.6 Genetic Algorithms for Feature Selection

A genetic algorithm is a search heuristic that takes Charles Darwin's natural evolution theory (Fatima et al., 2019). This algorithm mimics the process of natural selection, which will select the strongest to survive and produce offspring. A genetic algorithm will apply this logic to a dataset and can be used to produce an optimal feature set. The system proposed by Vivekanandan & Nedunchezhian uses a genetic algorithm to produce an optimal feature set for malware detection. A typical genetic algorithm will repeat its evaluation and crossover phase, creating numerous features to obtain optimal features. This research uses a genetic engineering approach to reduce the number of generations and features needed to produce the optimal feature set, otherwise known as a feature set. The general structure of a genetic algorithm is shown below.

· **Fitness Function:** The fitness function determines the ability each individual has to compete, in the context of a detection system, this would be determined by how accurate a feature set is.

· **Population Generation:** The initial population of individuals is generated randomly from the pool of available chromosomes; in most cases, chromosomes represent features that will create a feature set.

· **Selection** The selection phase is designed to take the fittest individuals and allow them to pass their genes onto the next generation. In the context of a detection system, these would be feature sets that achieve the highest accuracy.

· **Crossover:** Crossover is the process of two selected individuals being mated to produce a child, which will be a combination of both parents. This phase can be repeated with the offspring and so forth but can be limited to a select number of generations.

· **Mutation:** Genes of the offspring can be subject to mutation with a low random probability, in the context of a feature selection algorithm, this can mean inheriting a random feature that does not exist in either parent.

When a genetic algorithm operates, the population generation will come first; an initial pool of individual feature sets is produced. The crossover phase will be activated, and the fitness function determines the most robust individuals to result from each crossover phase. The crossover phase is the same as breeding in the natural world, and each crossover phase produces a new ”generation” of feature sets. Once a feature set defined as optimal by the fitness function is created, the crossover phase can halt. During each crossover phase, there is a defined probability for mutation; this mutation would imply an offspring feature set has a chance of getting a feature not derived from either of its parent feature sets.

2.2 Ransomware Detection Using Machine Learning

2.2.1 EldeRan

The EldeRan system is based on the observation that ransomware performs specific actions that are unique or significant concerning those performed by benign software (Sgandurra et al., 2016). The EldeRan system monitors a sandboxed environment (Cuckoo Sandbox) and extracts features in the following classes; Windows API calls, Registry Key Operations and File System operations, Directory operations, the set of operations done per file extension, dropped files and the strings of the actual executable. Besides the strings, the features are gathered and analysed dynamically. Once the collection of the features is complete, this is fed into a feature selection algorithm to extract the most relevant features. Once the final set of features is extracted, the data is put through the Regularized Logistic Regression classifier, which will return either ”ransomware” or ”goodware”. The system is trained offline but is run online, and new samples are classified at run time; this can be done on user PCs.

2.2.1.1 Feature Mapping

The feature selection component of this system uses the mutual information criterion, which allows the most discriminating features to be obtained (Sgandurra et al., 2016). The features used are binary; therefore, it is either the presence or absence of a feature that is the value. The mutual information criterion allows the user to quantify the amount of discrimination each feature adds to the classifier. The mutual information criterion will give the system an idea of how dependent or independent features are on whether a file is ransomware or benign. The mutual information formula reduces the feature list from an initial feature list of 30,967. According to the mutual information criterion, the most significant features in the final 400 features were related to Registry Key operations, with 48.25% of the features in the final 100 being Registry Key operations. The next most relevant category is the API stats features which make up 24% of the final features. The remaining 24% of features are all less than 10% individually. The additional features consist of traversed directories, files opened, deleted and modified, amongst other directories, and file-related activity, which is not specified in the research paper; however, manual inspection of their raw dataset indicates this.

2.2.1.2 Algorithm

The features are fed to the regularised logistic regression classifier to classify the executables used as either benign or malicious. Logistic regression is known to be strong at classifying when multiple variables are considered. Overfitting in the algorithm was reduced by using a regularisation function that attaches a cost penalty function to each feature, preventing overfitting. The justification for the use of regularised logistic regression is the fact that logistic regression is easier to train and add new samples to as opposed to a supervised method like SVM. Methods like Naïve Bayes will assume independence between features, but the assumption made when attempting to detect ransomware is that there is a strong dependence between the features (Sgandurra et al., 2016).

Due to the dataset's high volume of features, the algorithm chosen for classification was the Logistic Regression algorithm. The method aims to model the log-posterior probability of the different classes given the data via linear functions depending on the features. Then, the posterior probability of a sample being classified as ransomware ($R = 1$) given its feature vector x can be written as in eq.2.1. In the formula presented in eq.2.1, x represents the feature vector, w represents the vector of weights, b being the biased term and the sigmoid function $sgm(t)$ is given in eq.2.2.

$$Pr(R = 1|x, w, b) = sgm(w^T x + b) \tag{2.1}$$

$$sgm(t) = \frac{1}{1 + exp(-t)} \tag{2.2}$$

The main issue with Logistic regression is that it is prone to over-fitting when using the maximum likelihood of the posterior (Sgandurra et al., 2016); this is where regularisation is introduced by adding a penalty term to the cost function. The cost function for the regularised regression, C' becomes eq.2.3 where w represents the vector of weights, b being the biased term. λ represents the regularised parameter.

$$C'(w, b) = C(w, b) + \frac{\lambda}{2} \sum_{i=1}^{D} w_i^2 \tag{2.3}$$

The regularisation parameter is λ; this combination of a regularisation parameter acts as a penalty function, and the mutual information criterion counteracts the effects of over-fitting Logistic Regression.

2.2.1.3 Experiments

The Regularized Logistic Regression classifier is tested against implementations of Linear SVM and Naïve Bayes classifiers. The Mutual Information Criterion is used in all cases to find the most relevant features before applying the classifiers to the data. The experiments also used data from the VirusTotal AV detection engines. The data from VirusTotal contains data from multiple AV engines; this was aggregated and used in a voting system. This system worked around the rule that if the majority of the AV engines indicated that the sample was malware, it would be decided that VirusTotal classified the sample as ransomware. The top 5 vendors with the highest accuracy rates for experimental purposes were also used as a comparison benchmark. If an AV vendor does not provide a label, it is discarded from the results and not considered when calculating false positives; this gives the AV vendors an advantage. The experiments consistently used an 80% and 20% split for training and test data with over 100 different combinations of this 80% and 20% split. The dataset contains 942 benign applications and 582 ransomware samples. The detection rate achieved by the EldeRan system is 96.34% in comparison to 92.19% for SVM and Naïve Bayes achieving 94.53% accuracy rate. The EldeRan system also achieved the lowest false positive rate at 1.16%. Compared to the top 5 AV vendors, EldeRan was only second to AV vendor 1, with a detection rate of 96.89% and a false positive rate of 0.66%. It must be noted that the AV vendors all achieved better false positives than the EldeRan system and the other machine learning algorithms. The final phase of the experiments is on the new unknown ransomware samples. When experiments were carried out of the 11 families included in the dataset, 1 would be left out to test the system's ability to detect unknown ransomware samples that the algorithm had never been trained on. The overall detection rate goes down to 93.3% in this phase of experiments in which the system only attempts to classify unknown ransomware samples.

2.2.1.4 Pros & Cons

This approach comes with many positives while having some limitations. Firstly the system achieves a very high detection rate (96.34%) on ransomware families it is trained on, along with a 93.3% detection rate on unknown families it is unaware of, i.e., zero-day threats. With its effective use of static and dynamic features, the system can achieve detection rates that are more than competitive with the current Anti-Virus system. The system can identify ransomware infections in its earliest stages with static and dynamic features. Thanks to Logistic Regression, the relationship between the features and whether a file is a ransomware or benign is understandable. This system has limitations; firstly, it lacks any network features in this model. Considering the Cuckoo Sandbox provides means to extract network features from executed files, leaving them out of the feature set is a big miss. The main problem with this approach, as acknowledged by the authors, is that this method struggles to detect ransomware that lies dormant for an extended period or requires user input to activate the executable due to the reliance on sandbox techniques and lack thereof of scripting to simulate user input. Finally, the fact that regularised logistic regression is not robust with non-linear decision boundaries means that the model may find it hard to find complex relationships between features contributing to whether a file is ransomware or not is a limiting factor.

2.2.2 RansomWall

RansomWall is a layered system built to detect ransomware infections in real time. The layers are organised in order of execution of ransomware. This system is designed to be used for Microsoft Windows. The structure of the system is layered, with the first layer being the Static Analysis Engine, the second layer is a Honey Files & Trap Layer, the third layer is the Dynamic Analysis engine which works in real-time, and the fourth layer is the Backup Layer, and the 5th and final layer is the machine learning layer. The behavioural analysis data for the files are all extracted using Cuckoo Sandbox, with the static data coming from IDA (Interactive Disassembler). The system is set up in five layers, the first being the static analysis layer, which analyses the executable in a static context, i.e. strings. The second layer is a trap layer which uses honey files and directories. These files and directories are placed so ransomware will attack them before another actual user files; analysis of ransomware shows that a large proportion of them use a depth-first search approach when looking for files to encrypt (Shaukat & Ribeiro, 2018). The third layer is the dynamic analysis engine which provides behavioural data for the executable; the final two layers are the backup layer and the machine learning layers, which handle backing up of files if a ransomware infection is detected with the machine learning classifying samples based on the offline training the model has received.

2.2.2.1 Feature Mapping

The machine learning layer comprises Logistic Regression, Support Vector Machines, ANNs, Random Forests and Gradient Tree Boosting. The machine learning layer is based on "Sequential Supervised Learning with Moving Average Sliding Window" (Shaukat & Ribeiro, 2018). The output is either benign or ransomware hence why classifiers are used. Training is done offline; the execution of the system occurs in real-time in which the static, dynamic and trap layers send data to a feature collector, which converts data into the feature set. If a process is tagged as suspicious, the feature values of the data are sent to the machine learning layer, which will then process using the selection of algorithms to determine whether the sample is benign or ransomware. The exact algorithm for deciding what features to use from the three layers is not specified.

2.2.2.2 Algorithm

Gradient Tree boosting works on a gradient descent type system, which builds models progressively. Firstly a simple model will be built, after which the error residual will be calculated for the model, and then a new model will be built to attempt to correct the errors from the first model. The algorithm will continually rebuild the model to reduce the error in the previous model until the model prediction is acceptable. The gradient descent function attempts to reduce the gradient to close the gap between actual and predicted values. The collection of weak learners is represented in eq.2.4. F_i is the strong model at step i and f_i is the weak model at step i This formula is iterative and repeats until the stop criteria is fulfilled.

$$F_{i+1} = F_i - f_i \tag{2.4}$$

2.2.2.3 Experiments

Data is taken at 1-second intervals; however, to avoid glitches, data from 3 intervals will be taken and averaged. The model is trained on 11 out of 12 ransomware families selected and 221 out of

442 benign files. The trained model is tested against the remaining one ransomware family and the remaining 221 benign files. This is due to the belief that most ransomware attacks are zero-day attacks (Shaukat et al., 2018). The results for this system are auspicious, with the Gradient Tree Boosting method yielding a remarkable 98.52% detection rate with a false positive rate 0.0056%. The false-negative rate is due to two samples terminating early during execution making file system activity limited and, therefore, not detected (Shaukat et al., 2018).

2.2.2.4 Pros & Cons

RansomWall is a thoroughly executed approach with many upsides but has its limitations. In terms of strengths, RansomWall being multi-layered is its greatest strength. Having classifiers trained to detect ransomware using multiple behavioural vectors gives a stronger chance of detection than using one feature type like only static or API calls alone. It is important to ensure additional performance overheads introduced by the higher dimensionality are justified by additional performance. It uses static, dynamic, and honeypot layers to detect ransomware, making it a unique and secure approach. Using these layers to feed into a machine-learning engine gives the system a powerful appeal. In addition to detection, the system gives a protection layer in the form of its backup layer, which backs up files on detecting a ransomware infection. The system causes minimal overhead and uses robust ML algorithms to detect ransomware. Its detection rate using GTB is exemplary at 98.25%. Regarding limitations, the RansomWall system uses a dataset of 574 ransomware samples and deficient 442 benign files. Considering how varied benign files can be, it raises the question of whether the system has enough training on expected behaviour. Despite having a comprehensive multi-layered approach, the system does not use any network behaviour, which usually provides one of the earliest indicators of ransomware infections. Finally, the use of GTB can sometimes become convoluted due to it being prone to overfitting; therefore, shrinkage and tree depth must be carefully monitored. According to the authors, expanding RansomWall to function on large-scale networks is the next step for the system; however, the most beneficial addition to this system in the future would be the active monitoring of network features along with the dynamic and static features; this would be a massive step forward for the system to function on large-scale real-world networks effectively.

2.2.3 RansHunt

RansHunt is a framework designed to identify key features that define a ransomware infection and then use these features to detect ransomware using support vector machines (Hasan & Rahman, 2017). This system uses dynamic and static features extracted from 21 ransomware families. This dataset is completed by an additional 1283 samples, benign, ransomware, and other malware. The system is compared to Decision Trees and Naïve Bayesian methods.

2.2.3.1 Feature Mapping

Feature selection aims to find the features which will allow the system to distinguish ransomware from normal, benign files. The mutual information criteria identify the best features for the SVM. Mutual information criteria allow the user to quantify how much discrimination each feature adds to the classifier (Sgandurra et al., 2016). It takes X and Y as two discrete random variables with a joint probability mass function $p(x, y)$ and marginal probability mass functions $p(x)$ and $p(y)$ (Hasan & Rahman, 2017).The mutual information $MI(X, Y)$ is the relative entropy between the joint distribution and the product of the marginal distribution distributions. The features are finalised in training samples.

2.2.3.2 Algorithm

SVM consists of a Hyperplane dividing n-dimensional space, representing data divided into two classes, in this case, ransomware or benign files. The hyperplane is designed to maximise the distance between two separate classes, with the maximal margin being defined as the most significant distance between the examples of the two classes computed from a distance between the closest instances of both classes (Hasan & Rahman, 2017). The hyperplane is represented by a vector w and a scalar m in a way that the inner products of w with vectors $\varphi(Xi)$ from the two classes are divided by an interval between -1 and 1 subject to b (Hasan & Rahman, 2017).

$$(w \cdot \varphi(X_i)) - b \geqslant +1 \tag{2.5}$$

For every X_i that belongs to the first class and or ever X_i that belongs to the second class:

$$(w \cdot \varphi(X_i)) - b \leqslant -1 \tag{2.6}$$

2.2.3.3 Experiments

The experiments were carried out in a dynamic and static context, with static analysis on IDAPro and IDA2SQL and dynamic analysis on Cuckoo Sandbox. The dataset contains 360 ransomware samples, 532 types of malware and 460 benign files. Selecting features for the static analysis initially started with 64,984 features but was reduced to 100 using Mutual Information Gain. Mutual information gain measures the information you gain on one variable by learning the value of another variable. The dynamic features are taken from the cuckoo sandbox, which totalled 67 and focuses on Registry Key operations, API function, and files operation features. These features are combined with the static features obtained into a hybrid dataset. The static and dynamic datasets are kept in their initial states so tests can be run on them. The system is trained on 90% of the dataset using 10-fold cross-validation. The RansHunt system uses SVM with a normalised polynomial kernel, compared to the results obtained by Naïve Bayes and Decision Trees implemented in WEKA. The SVM algorithm is compared to the most prominent malware analysis platforms available: VirusTotal and Malwr. The RansHunt system achieved a 93.5% accuracy rate with the static dataset, a 96.1% accuracy rate with the dynamic dataset and a 97.1% accuracy rate with the hybrid dataset. RansHunt outperforms Decision Trees (95.6%) and Naïve Bayes (96%) along with the VirusTotal Engine (95%) and the Malwr Database (93%). RansHunt also achieves the lowest error rate at 2.1%. RansHunt is also compared to 3 mainstream Anti-virus programs and is only outperformed by NG-AV; this is most likely due to the inclusion of R&D along with fine-tuned signatures (Hasan & Rahman, 2017). It must be noted that for unknown threats, RansHunt would be more effective because it does not rely on signatures or a virus database. The inclusion of other malware, including worms, is to be able to detect a ransomware variant named "Ransomworm", which is one of the forecast attack patterns in the coming year.

2.2.3.4 Pros & Cons

RansHunt is a strong approach and shows very promising results; however, this is not to say that it does not come with limitations. In terms of strengths, RansHunt boasts a 97.1% detection rate with a 2.1% error rate. The training methods used for RansHunt use the concept of being future proof with the model being trained on worms and Trojans. These behavioural patterns are taught to the model to anticipate next-generation ransomware, "Ransomworm" [12]. This new trend in ransomware behaviour is expected to be a ransomware/worm hybrid and expected to come into prominence within the next two years. In terms of ML methods, the system uses a highly robust SVM system which by default is tuned to avoid over-fitting issues posed by other models. SVM's kernel can be fine-tuned to specific problems, meaning the system can be tuned and updated to be purpose-built for ransomware. The feature set uses a hybrid of static and dynamic features, identified by the Mutual Information Criterion. In terms of weaknesses, SVM tends to underperform in comparison to deep learning approaches despite being strong in terms of speed and memory efficiency. Unmodified SVM also does not give probabilistic confidence of the values calculated, meaning the model is less understandable. When it comes to the system, the accuracy levels of detection may be deceptive because this system is not tested on zero-day threats. The dataset is not split in a way in which one family would be excluded from training and left specifically to testing. This approach, while using a hybrid approach of static and dynamic features, decides not to include any network features which could have been pivotal for early detection. The true effectiveness of this system needs to be tested on zero-day threats, while a 97.1% detection rate is very high; this statistic may not reflect real-world performance. It would be useful to test how useful the future-proofing on the system is, in terms of the ransomware & worm hybrid. As this system has a dataset of hybrid features, static and dynamic; it would make sense to include network features in the dataset also to add a layer of security to the system.

2.2.4 Behavioural-Based

Behavioural-Based Classification and Identification of Ransomware Variants Using Machine Learning considers that the use of polymorphic and metamorphic ransomware is starting to increase (Daku et al., 2018). This approach uses machine learning models to identify modified versions of ransomware based on their behaviour. The study uses 150 samples of ransomware from 10 different ransomware families. This research utilises some of the newest ransomware samples to understand how machine learning algorithms work when classifying evolving ransomware. This method uses an iterative approach to identify optimum behavioural attributes which achieve the best classification accuracy. The behavioural data is taken from the Cuckoo sandbox, which produced behavioural logs for executables. Unlike many other studies, this research does not attempt to classify ransomware apart from benign files; the goal is to classify ransomware into their respective families with a dataset comprising only ransomware samples. The experiments are done in WEKA, a platform that allows the user to utilise various machine learning algorithms on a dataset.

2.2.4.1 Feature Mapping

The initial set of behavioural attributes is selected by taking behavioural attributes which appear in 95% of behavioural reports. The next step of the procedure was to add and remove attributes that increased the classification accuracy of the J48 algorithm. An iterative approach is used to select the attributes that give the maximum accuracy. The most important features of the feature set are located on the top levels of the tree. An iterative approach utilises this method to root out irrelevant features until the attributes that appear on the top of the trees in iterations make up the bottom of the trees, as well. The final number of features used in the dataset is 12. The 12 behavioural attributes used are not specified. All the behavioural attributes are common in all the samples; they vary in type, either nominal, binary, or numeric. This approach takes inspiration from Reference, which uses extensive testing on a wide variety of datasets when performing attribute selection.

2.2.4.2 Algorithm

As the name suggests, the J48 Decision Tree structure takes on the structure of a tree. The training dataset is used to construct a tree to make test data predictions. The aim is to achieve the most accurate results with the least number of decisions. The J48 decision tree can be used to solve classification and regression problems. This method of classification relies on the concept of Entropy and Information Gain. Entropy refers to the uncertainty of the data. Example: The entropy of a coin toss is indefinite as there is no way of predicting the outcome; however, if the coin were two-headed, the entropy would then be zero as the outcome can be predicted with 100% outcome. The primary algorithm used in decision trees is the ID3 (Iterative Dichotomister 3). The ID3 algorithm aims to start from the root and partition the data into a homogenous dataset. We want the attribute that would result in the highest information gain (return the most homogenous branches). The decision tree approach is popular because it can handle large datasets very well, deals with noise and operates in a White Box, meaning that we can observe how the outcome is obtained and what decisions lead to the outcome. It is a popular method of tackling medical diagnosis, spam filtering and security screening. The entropy calculation involves splitting the dataset and calculating the entropy of each branch. Calculate the information gained from the split. Information gain is the differences in the initial entropy and the proportional sum of entropies of the branches. The attribute with the highest gain value is selected as the decision node. A branch with an entropy of 0 becomes the leaf node. Other branches will still require further splitting. This process runs recursively until further splitting is impossible.

2.2.4.3 Experiments

The J48 decision tree algorithm is compared with the K-Nearest neighbour algorithm and the Naïve Bayes. N-Fold cross-validation tests the models in the training phases (Daku et al., 2018). The n parameter is set to $n = 10$, which is the default in WEKA, the platform where all the experiments are carried out. Overall results are relatively weak, with the J48 decision tree algorithm achieving the highest accuracy of a 78% detection rate, Naïve Bayes at 61% and K Nearest Neighbour at 77.33%. This inaccuracy may be attributed to the use of the newer variants of polymorphic ransomware, with samples of Cerber having a detection rate as low as 50%. Cerber is suspected to be designed to avoid machine-learning detection techniques, and these results show further evidence that supports this. There is a significant drop-off when correctly classifying newer versions of ransomware in general, with Locky, Petya, Jaff, WannaCry, Sage and Cerber variants all having classification rates well below the 80% mark; this suggests the behaviour of these variants displays high polymorphism with variants behaviour differing significantly enough to be challenging to identify which family the ransomware belongs to.

2.2.4.4 Pros & Cons

This research takes an interesting approach in terms of not detecting ransomware and differentiating them from benign files but classifying which family of ransomware they belong to. This approach has strengths along with some pretty apparent weaknesses. This model's main strength is that the results give us a good idea of which ransomware families display polymorphic behaviour. We can get a good idea of which ransomware variants might be using machine learning evasion techniques. This study's most robust machine learning algorithm, the J48 decision tree, implicitly performs feature selection, thus eliminating the need for a separate feature reduction system. The preparation of data used in decision trees is relatively short, making it easier for users to feed data into this algorithm. In terms of weaknesses, the lack of significant modification or tweaking to the algorithms used means the results are naturally limited. Whereas with modification, the classification of the ransomware families may be more accurate. Furthermore, decision trees in this process can be complex, considering how complex and diverse ransomware can be. Expectations in decision trees play a big part in the classification process; while expectations are realistic, the classifications are strong; however, if expectations are irrational, this can lead to errors. Decision trees can follow a natural course for events but cannot always plan for every contingency. To improve this work in the future, the J48 algorithm could be tweaked to be tuned for the data presented. The classification of more modern ransomware variants needs to be significantly improved. A method considering the high polymorphism in the newer samples will significantly improve the correct classification rate.

2.2.5 SVM

Detecting ransomware using second in this survey that utilises Support Vector Machines, using the API calls used by ransomware. The idea behind this model is to train an SVM to learn the API calls ransomware makes to detect unseen ransomware (Zero-day threats). The SVM system uses a vector representation of the API calls, in which the number of API calls is counted; this is done by looking into the execution logs of the samples (Takeuchi et al., 2018). A standardised vector representation is designed to accommodate the programs' diversity. The experiments carried out in this research are all done in Cuckoo Sandbox.

2.2.5.1 Feature Mapping

The API calls used by the ransomware samples are used as the feature set for the data; however, it is not specified what exact features are used. The central concept is that the API logs of the malicious programs need to be quantified for the SVM to use them as features (Takeuchi et al., 2018). Because the number of API calls differs from the program, a standardisation of the vector representation is used. This standardisation method is described in detail (Takeuchi et al., 2018). Using the logs and quantifying them into vectors, the SVM can learn the sequence of API calls ransomware uses during execution.

2.2.5.2 Algorithm

SVM classifiers consist of a hyperplane dividing n-dimensional space based on the data into two classes between positive and negative samples. The hyperplane aims to maximise the margin between the two classes. The maximal margin is defined by the most significant distance between the examples of the two classes computed from the distance between the closest instances of both classes (Hasan & Rahman, 2017). The mathematical and algorithmic background for SVM has been discussed in the review of RansHunt; therefore, it will not be repeated in this section.

2.2.5.3 Experiments

The experiments are conducted in Cuckoo Sandbox, which produces all behavioural logs. The dataset used contains 276 ransomware files and 312 benign executables. Various files were taken from trusted sources and vendors. The Cuckoo Sandbox environment deployed virtual machines with ten main and 1000 subfolders. Once a behavioural log is produced, it is converted into a vector and fed into the support vector machine. Once the SVM processes the features, it will decide if the file is either ransomware or benign. The true positives and true negatives decide the validity of the results. The results are compared to the approach used by (Rieck et al., 2011). The SVM model in this approach achieves a 97.48% detection rate, which is superior to Rieck et al.'s approach, which produces an accuracy of 94.18%. When using the standardised vectors designed to compensate for software diversity, the detection accuracy of the SVM-based approach decreases to 93.52%; this is possible because only 588 samples are used; therefore, the dataset is not diverse enough.

2.2.5.4 Pros & Cons

This approach is the second SVM base approach reviewed in this study and has strengths and weaknesses, much like the RansHunt system, which also used SVM. Firstly the system achieves a robust detection rate of 97.48% with non-standardised vectors and a detection rate of 93.52% with standardised features. API calls allow the system to analyse ransomware dynamically, which means the system can potentially be used live. The use of SVM allows for the use of the Kernel trick allowing for fine-tuning of the Kernel to purpose fit the problem, in this case, ransomware. In terms of weaknesses, this model suffers from a noticeable lack of diversity in training, demonstrated by its standardised vectors reducing detection accuracy, the use of 276 samples of ransomware may be acceptable, but the fact that the system only uses 312 benign samples is massively limiting. Diversity in benign software is crucial, and using only 312 samples limits the model's ability to identify normal behaviour compared to ransomware behaviour. In terms of future work, this research can be improved simply by diversifying the dataset to use their standardisation vectors better. Adding network and static features would be beneficial, considering the malware trends suggest that ransomware is becoming increasingly polymorphic.

2.2.6 SDN (Software Defined Networks)

Machine Learning-Based Detection of Ransomware Using SDN (Software Defined Networks) takes the network monitoring route, using a specific type of hardware, programmable forward engines (PFEs). PFEs allow the collection of per-packet network monitoring data at high rates (Cusack et al., 2018). This hardware monitors the traffic flow between the infected PC and the C&C centre, which gives ransomware execution commands. High-level network flow features are extracted from the traffic and are used for classification. This approach uses random forests which fingerprint malicious traffic. This method utilises network flows to show that a flow-based fingerprinting method is feasible and accurate enough to detect ransomware before the beginning of the encryption phase.

2.2.6.1 Feature Mapping

The features in the model are extracted based on observation of the victim system's communication with C&C. Firstly, communications between the ransomware and its C&C centre go through proxies which will cause a higher latency (Cusack et al., 2018). The measurement of packet intervals will represent this latency. The volume of incoming traffic is expected to be higher than the outgoing traffic volume because this increased inbound traffic represents the initial infection, encryption key retrieval and payment method notifications. The burst lengths of traffic flow are also recorded, which can help distinguish between a clean and a malicious download. The final feature count is 8, consisting of inflow and outflow length with inter-arrival time metrics. This feature count was reduced from 28 based on the random forest classifier's accuracy using different feature combinations.

2.2.6.2 Algorithm

A popular machine learning algorithm is based on the J48 decision trees described in the J48 decision trees. Random forests require almost no data preparation but result in accurate results. Random forests are a collection of decision trees which is why it is referred to as a "random forest" with multiple trees being built on two-thirds of the training data, which is chosen at random. Multiple predictor variables are randomly selected; the best split on these selected variables is used to split the node. The misclassification rate is calculated using the rest of the data. The total error rate is calculated as the overall error rate. This model tunes the random forest using three parameters; the number of trees in the forest, the depth of each tree and the number of features used in the trees. The tree depth is 15, and the number of trees used is 40; this combination was chosen to minimise computational overhead and learning time. Random forests use a tree-based approach using the Gini index to split the nodes. The Gini Index is shown in eq.2.7 where Y is the total number of classes and p_y is the probability of picking a data point with class y.

$$Gini(n) = \sum_{y}^{|Y|} p_y(1 - p_y) \tag{2.7}$$

2.2.6.3 Experiments

The experiments were conducted on 265 unidirectional unique ransomware network traffic flows. The average baseline consists of 100MB of non-malicious traffic, which includes traffic flows from web browsing, data downloading and file streaming. Features were decided upon using the analysis of malign and benign traffic flows. The data is split 70% training and 30% testing. 10-fold cross-validation is used performed to ensure the splitting is unbiased. Using 28 features yields a detection rate of 89%, and the eight-feature model yields an accuracy of 87%. The set of 8 features is used because it is less computationally heavy. Further experiments are carried out on only the Cerber ransomware family using the eight most prominent network features of Cerber, yielding an AUC of 0.987.

2.2.6.4 Pros & Cons

This approach to ransomware detection is unique in that it takes only network behaviour without the need to look into the payload to detect ransomware. The main strength of the approach is the ability to pinpoint network behaviour on a high level, which can help detect ransomware. This model can detect while minimising computational overhead while maintaining a reasonable detection rate. Random forests can be useful in a dataset that is not too large like this one. In terms of weaknesses, the obvious weak point is the relatively low detection rate at 87%; this is interesting considering that a whole decision tree-based approach tends to perform exceptionally well in behavioural analysis; this may suggest that network behaviour on its own may not be sufficient when detecting ransomware. Random forests tend to struggle with highly diverse behaviour, i.e. next generational ransomware, which may break the model's average trends already known. This model is not tested on zero-day threats, which is a big miss, as its potential to detect the unknown is somewhat untested. While achieving very high detection rates, this study has its own set of limitations. While achieving a 97.1% detection rate is impressive, the dataset used is vastly limited in terms of only having 210 ransomware samples and 264 benign files. The minimal training the models receive on benign behaviour will likely lead to the model becoming confused when it comes to being deployed into the real world unless actual expansion and tuning is carried out. The lack of tuning also reinforces the authors' statement that this research acts as a baseline that other researchers can build on, considering the algorithms have received no tuning to allow them to be purpose-built for ransomware. The future of this work is envisioned to be expanded by other researchers into areas such as accurate time detection of ransomware using cloud-based machine learning classifiers (Alhawi et al., 2018). The expansion of the dataset and increase in relevant attributes will improve detection rates and the longevity of the model overall.

2.2.7 Bayesian Networks

This multi-classifier detection system is a network-based detection system based on the Locky Ransomware. This approach is adapted by using close observations of the malware's behaviour on the network to develop a set of observable features that can identify and differentiate between network traffic generated by Locky Ransomware from regular network traffic. The choice of isolating Locky was made due to its prominence as a Crypto-Ransomware. The multi-classifier works in multiple layers with one classifier monitoring packet and one classifier monitoring network flow.

2.2.7.1 Feature Mapping

The features used in this system were not built or decided upon using an algorithm but decided using observations on the packets and the network flow. The first distinguishable feature identified is the IP-wise reset connections ratio (Almashhadani et al., 2019). The authors noticed that in the 15-minute time frame, the malicious traffic would repeat the same set of IP addresses with a high reset connection ratio nearly every time. This feature does not have significant longevity because future Locky variants could terminate their connections early using TCP and FIN without RST to make their traffic seem more benign. The second distinguishable feature is the increase in the number of HTTP-POSTs within the traffic stream due to Locky. Most Locky variants use the HTTP-POST without specifying the User-Agent, with no such instances being found in regular traffic. The third distinguishable feature identified is the large volume of DNS name errors in the malicious traffic. This feature is identified as a behavioural feature because Locky is based on the DGA algorithm, designed to generate many pseudo-random domain names. The underlying technology behind the malware would have to change for this feature to be labelled non-behavioural. The next feature found was the DNS labels used by the malware. It was noted by the authors that Locky only used DNS names with two labels, whereas benign traffic used multiple labels for DNS names. The final feature used was the presence of NBNS (NETBIOS NAME SERVICE) packets in 6.05% of malicious traffic.

2.2.7.2 Bayesian Networks

A Bayesian Network (BN), in general, is a relationships network that uses statistical methods to represent probability relationships between different nodes. It is a compact representation of the joint probability distribution for reasoning under uncertainty. The mathematical function, Bayesian theory used for a Bayesian network is shown below.

$$P(H|E,c) = \frac{P(H|c)\cdot P(E|H,c)}{P(E|c)} \tag{2.8}$$

Here, H is the hypothesis given additional evidence E and background information c. $P(H|E,c)$, is the posterior probability, which is the probability of H considering the effect of E given c. $P(H|c)$ is the prior probability of H given c alone. The term $P(E|H,c)$ is called the likelihood and gives the probability of the evidence assuming the hypothesis H and the background information c is true. Finally, the last term $P(E|c)$ is called the expectedness, or how expected the evidence is given only c. It is independent of H and can be regarded as a marginalizing or scaling factor.

2.2.7.3 Experiments

The experimental setup in this system consists of 5 PCs, the first PC, PC1, is the victim machine where the ransomware is infected. The second PC2 hosts two virtual machines, VPC1 and VPC2, which represent three clean machines on the network and will show how the ransomware attempts to spread through the network. PC3, the third PC, will use Wireshark to capture traffic for analysis. This PC will run Ubuntu, and its NIC is set to promiscuous mode to prevent infection, along with the fact that Locky cannot target Linux systems. The data extracted is fed into the packet and flow-based classifiers, which will then make a decision based on the choices of the two classifiers; these classifiers are built using Random Forest, LibSVM, Bayesian Networks and Random Tree; however, the technical aspects of how these classifiers contribute are not specified. In terms of the packet-based classifier, the Random Trees prove most effective with an accuracy of 98.72%. The flow-based classifier displays the most success with Bayesian networks with an accuracy of 99.83%. On average, the flow-based classifiers were all more accurate than the packet-based classifiers.

2.2.7.4 Discussion

This approach comes with a good number of positives while also having drawbacks; the system does achieve a high accuracy rate at 98.72% for packet-based detection and 99.83% for flow-based detection for Locky ransomware. This approach's novelty and most significant strength are that it uses two vectors to detect the ransomware: a packet-based detection algorithm and a network flow-based detection algorithm. The system provides two angles to monitor ransomware activity, giving the system a better chance of catching the malware before it propagates through the network or infects the machine. The main issue with this research is that it only uses one family of Ransomware, Locky, to perform all tests and training. Considering the research was published in 2019, it would be constrained when it came to detecting any other type of ransomware; considering how vast the amount of ransomware in the wild is, using Locky alone seems almost redundant. The second most significant issue with the system was that the features were all collected based on the authors' observations and not based on any mathematical feature engineering/extraction approach, which would have been chosen from many extracted network features. For the system to move forward, it would need to be exposed to more ransomware families; considering the vast number of them in the modern environment; it is redundant to train a detection algorithm to only detect one family out of dozens of existing ransomware families. Using a mathematical feature extraction method instead of manual observations would enhance the system significantly.

2.2.8 NetConverse

NetConverse analyses machine learning algorithm performance on a Windows ransomware network traffic dataset (Alhawi et al., 2018). The research considers the development of Ransomware variants which are now being engineered to evade machine learning detection. The dataset comprises conversation-based network traffic. This approach considers dynamic analysis techniques' limitations in that new ransomware variants can be redesigned to decrease the detection rate by machine learning algorithms.

2.2.8.1 Feature Mapping

Feature selection is made using TShark, which outputs statistical and calculated data along with static feature extraction. TShark is an extension of the network analyser, Wireshark. Each network PCAP file is merged into a dataset based on the features extracted from within the PCAP file, and ten features are used. The features range from the protocol used to the origin and destination address to packets and duration.

2.2.8.2 J48 Decision Trees

As the name suggests, the data structure takes on the structure of a tree. The training dataset is used to construct a tree to make test data predictions. The aim is to achieve the most accurate results with the least number of decisions made. The J48 decision tree can be used to solve classification and regression problems. This method of classification relies on the concept of Entropy and Information Gain. Entropy refers to the uncertainty of the data. Example: The entropy of a coin toss lies between 0 and $log(2)$ as there is no way of predicting the outcome; however, if the coin were two-headed, the entropy would then be zero as the outcome can be predicted with 100% outcome. The mathematical and algorithmic background of the J48 decision trees are discussed in the background information of the behavioural-based system, so they will not be repeated in this section.

2.2.8.3 Experiments

The data taken from the TShark and once feature selection is a complete run through multiple machine learning algorithms; Bayesian Networks, MLP, J48, KNN, Random Forests and LMT. The highest accuracy achieved was by the J48 algorithm, which achieved a 97.1% accuracy rate. All experiments were carried out in WEKA with a 10-fold cross-validation approach using all ten extracted features. The J48 algorithm achieves the highest accuracy rate with a very low false positive rate. All of the data used in the experiments are extracted from virtual machines run in a VMWare workstation with all the classifiers not being tuned. The dataset comprised 264 benign files and 210 examples of malign files.

2.2.8.4 Discussion

While achieving very high detection rates, this study has its own set of limitations. While achieving a 97.1% detection rate is impressive, the data set used is vastly limited in terms of only having 210 ransomware samples and 264 benign files. The minimal training the models receive on benign behaviour will likely lead to the model becoming confused when it comes to being deployed into the real world unless actual expansion and tuning are carried out. The lack of tuning also reinforces the authors' statement that this research acts as a baseline that other researchers can build on, considering the algorithms have received no tuning to allow them to be purpose-built for ransomware.

2.2.9 API-Based Ransomware Detection

This API-based ransomware detection approach proposed by Almousa et al. aims to detect ransomware infections using Windows API calls. The API-Based detection system achieves a strong ransomware detection rate of 99.18%. The system uses simulated data that is extracted from the Cuckoo sandbox and is used to train a machine-learning classifier. The API system uses data simulated for 10 minutes to obtain the necessary data for detection training (Almousa et al., 2021).

2.2.9.1 Feature Mapping

The features used in this system are taken from Cuckoo execution reports. Almousa et al. identify 249 ransomware and 229 benign features and combine this into a joined feature set of 206 common features. The common features are identified using principle component analysis, a method used to reduce the dimensionality of datasets. Principle component analysis attempts to reduce dimensionality while retaining maximum information.

2.2.9.2 Algorithm

This research uses SVM, Random Forests and k-Nearest Neighbour as its detection algorithms. These algorithms have been proven effective in previous research we have reviewed and can be effective in detecting ransomware given the correct feature sets. The k-Nearest Neighbour algorithm achieves the best results with a 99% accuracy rate.

2.2.9.3 Experiments

The experiments in this research use a dataset of 58 ransomware and 66 benign files, which is limited. The experiments use the random forest, SVM and k-NN; k-NN records the best results with a 99.18% accuracy rate. The data collected for each sample was of 10 minutes of activity; however, this may not prove optimal as there are ransomware strains that could infect and encrypt most files during that time. The dataset is limited, and 58 ransomware samples cannot possibly cover the behavioural patterns of the various ransomware samples. The dataset needs to be expanded significantly.

2.2.9.4 Pros & Cons

Overall, this research is limited because it uses a minimal number of ransomware and benign samples. The system would unlikely maintain its 99% detection rate if exposed to zero-day ransomware. The API calls that span 10 minutes of execution may give ransomware enough time to encrypt most files on a system. The research needs further expansion and more emphasis on detecting zero-day ransomware, as newer variants have the best chance of causing optimal damage. The research also uses API calls without any static or network features; this could be added to improve the system's reliability.

2.2.10 API-Call Based Detection (2)

The research proposed by Sheen & Yadav uses API-call usage to detect ransomware. API calls are widely used in malware detection research, so it is a logical approach. This research aims to identify the most distinguishing API calls to identify ransomware and benign files. This research uses many machine learning algorithms to test its detection capabilities.

2.2.10.1 Feature Mapping

The features for this research are API calls that are common between benign and ransomware samples. The features consist of API calls that were more frequent in ransomware by a factor of at least 0.1 (Sheen & Yadav, 2018). The final feature set is a binary feature set of 160 API calls. The use of an API call constitutes having a particular feature. There is no specific feature selection method or algorithm besides having common features that are more frequent in ransomware than in benign files.

2.2.10.2 Algorithm

Random forests are the most effective algorithm used in this research; Sheen & Yadav. experimented with various algorithms such as IBK, Naïve Bayes, Decision Trees and JRIP. Random Forests being effective is likely due to the binary nature of the feature set; this binary feature set will make it very simple for a random forest to identify the combinations of different API calls that differentiate between benign and ransomware files.

2.2.10.3 Experiments

This approach has a large dataset of 16243 ransomware samples and 3620 benign samples. The dataset used in this research is skewed heavily towards ransomware; the issue with this approach is that any detection system will come into contact with more benign files than malign; therefore, it is important to expose a system to a variety of benign files. Sheen & Yadav recognise the imbalance in the dataset and apply SMOTE (Synthetic Minority Oversampling Technique)to synthesise benign samples to add to the dataset. SMOTE selects samples from the minority class, which in this case is the benign class and uses K-nearest neighbours to find the neighbours near the selected sample. The SMOTE algorithm then creates synthetic samples between the chosen sample and it k nearest neighbours. The highest performing algorithm for the dataset is the random forest with a detection rate of 96.53% and a low false positive rate of 7.5%

2.2.10.4 Pros & Cons

This research proposes an API-based approach to detecting ransomware and shows some promising results; however, the high detection rate does not tell the whole story. The API calls are used as binary features, allowing attackers to disguise ransomware by using API calls that are usually present in benign files to cover the tracks of the ransomware. In addition to using binary features, Sheen & Yadav do not specify what period is used for capturing these API calls; the time taken for the execution is essential because ransomware can encrypt files quickly; therefore, it is essential to catch it early in its execution. The research also fails to take any zero-day detection or evolutionary ransomware into account; these types of attacks are the most prevalent in ransomware.

Using binary features would make a detection system sensitive to concept drift with very slight adjustments to API behaviour in ransomware.

2.2.11 Ransomware Detection with Paranoia Activities

The research proposed by Molina et al. attempts to classify ransomware before its encryption phase by using pre-encryption "Paranoia" activity. The paranoia activity is pre-encryption API calls triggered by ransomware just before its encryption phase. Molina et al. test their method using a different machine and deep learning techniques to find the optimal solution; the optimal solution is achieved by combining random forests and OoW (Occurrence of Words) techniques.

2.2.11.1 Feature Mapping

The feature extraction process focused on finding paranoia activities (Molina et al., 2022). Paranoia activities attempt to realise the presence of a dynamic analyser (Molina et al., 2022). The paranoia activities are identified using previous research that has already defined API calls used to avoid detection. Molina et al. define 23 unique API calls commonly used to evade detection when malware starts executing. Following on from the 23 API calls, Molina et al. use techniques such as the Occurrence of Words (OoW), Bag of Words (BoW), and Sequence of Words (SoW). The OoW technique considers the occurrence of an API call as a binary feature, and the BoW considers the number of times an API call is invoked. The SoW technique considers the sequence of API calls and only considers a unique sequence of API calls; therefore, the same API calls in the sequence are not considered.

2.2.11.2 Algorithms

The algorithms used in this research include K-Nearest Neighbour (KNN), Random Forest, Artificial Neural Network (ANN), Long-Short Term Memory (LSTM), and Bidirectional Long Short-Term Memory (Bi-LSTM). Random Forest achieves the strongest results with a 96.8% recall rate for ransomware; however, it also achieves an 81.8% recall rate for benign files. The random forest algorithm has been discussed in this thesis in section 2.8.2.

2.2.11.3 Experiments

The dataset used in this research comprises 2994 ransomware samples and 438 benign samples. The dataset starts with nearly 20,000 malware files and is sorted into ransomware and benign samples; the benign samples are taken from a fresh Windows install. The API call data is taken from executing samples in the Cuckoo Sandbox and analysing the file activity reports. The Random Forest and OoW combination achieve the highest recall for ransomware. The machine learning algorithms appear to have trouble correctly identifying benign files, with the highest recall rate being 88.64% and the Random Forest achieving a recall of 81.81% for benign files. The low recall for benign files would mean a high false positive rate of ransomware detection. The high false positives could be due to benign files having similar API call patterns to some ransomware; this would be common paranoia activity.

2.2.11.4 Pros & Cons

The research proposed by Molina et al. achieves high ransomware detection statistics; however, the detection rate for benign files is lower than desirable. The high false positive rate is not as severe as having a high false negative rate; however, it would be desirable to have a similar recall for ransomware and benign files. The limitations of this research state that the model would struggle to classify new unseen ransomware; this is a significant shortcoming in the ransomware space due to the rapid evolution and emergence of new ransomware. Ransomware detection systems should be hardened to focus on detecting new unseen strains, and without the ability to do so, it would be severely limited in lifespan and real-world effectiveness.

2.2.12 Swarm-Based Ransomware Feature Selection & Detection

Abbassi et al. propose an automated feature selection method for ransomware detection with machine learning algorithms. The system proposed uses particle swarm optimisation for a behavioural-based detection system; this means the features used for ransomware detection are dynamic and taken during execution. The approach proposed groups features together and performs feature selection based on the significance of a group.

2.2.12.1 Feature Mapping

The feature mapping process starts with organising features into groups and taking features from each group. The features taken from each group are the top-ranked among the group; the mutual information criterion is used to select the highest ranking features in every group so the overall collection of features will have the highest information gain possible. The second stage of the feature selection process involves using logistic regression with particle swarm optimisation. The particles in the swarm are moved into their relative best positions to achieve the best possible position for the swarm.

2.2.12.2 Algorithm

The most effective algorithm used in the experiments was the regularised logistic regression algorithm, which is also used to evaluate the features during stage two of the PSO analysis of the features. The logistic regression is the most effective, and this is not a surprise as this research uses the same dataset as the Elderan system; the Elderan system also found regularised logistic regression to be the most effective algorithm for its data. The regularised Logistic regression algorithm is proven effective and reduces overfitting in classification.

2.2.12.3 Experiments

The dataset used for this research is the Elderan dataset. The Elderan dataset comprises 582 ransomware samples and 942 benign samples. The experiments use a range of machine learning algorithms; Random forests, regularised logistic regression, decision trees, K-nearest neighbours and SVM. The experimental setup uses a k-fold cross-validation approach using the Elderan dataset. The most obvious issue with the experiments is the dataset; the Elderan dataset contains ransomware from 2012 to 2015. Considering this research paper was published in 2022, it does not make sense to only experiment with ransomware that is almost a decade old; attack patterns and ransomware, in general, will have evolved during this time.

2.2.12.4 Pros & Cons

The research proposed by Abbasi et al. provides strong results for detection; however, the detection results achieved by the regularised logistic regression are overshadowed by the random forest without the proposed approach; this poses the question as to whether this approach is even necessary as it introduces computing overhead while not achieving results superior to approaches that do not need it. The second most noticeable issue with this approach is that it does not provide a significant improvement over the baseline approach and, in some cases, makes results worse than its competitors and baseline classifiers. Using an outdated dataset is also an oversight; the Elderan dataset was created in 2016 and would be useful to combine with modern ransomware data.

2.2.13 Pre-Encryption Ransomware Detection

The pre-encryption detection system attempts to detect ransomware before the encryption phase of an attack begins. The pre-encryption detection system acknowledges the irreversible nature of ransomware and the lack of research that considers population drift (Urooj et al., 2021). The pre-encryption detection system addresses that ransomware attacks are evolving, and current research does not consider developing crypto-ransomware variants. The approach proposed by this research deals with the current work in ransomware detection limitations by developing an evolutionary model that works well with developed ransomware variants (Urooj et al., 2021).

2.2.13.1 Feature Mapping

Before mapping the features, the pre-encryption system defines the pre-encryption boundary definition (Urooj et al., 2021). The pre-encryption boundary definition identifies the ransomware behaviour that is associated with encryption. The first phase of this analysis is started when the first cryptographic API is triggered. The pre-encryption phase distinguishes pre-encryption from the encryption phase. The feature set for the pre-encryption data is built using a mutual information method used by various ransomware detection systems covered in this literature review.

2.2.13.2 Algorithm

Urooj et al. use a Stochastic Gradient Descent algorithm to classify ransomware in the pre-encryption stage. The classifier is designed to update itself when misclassification occurs to evolve and keep in touch with the current concept. The stochastic gradient descent is not expanded as this approach can be used with many machine learning algorithms.

2.2.13.3 Pros & Cons

This research proposes an interesting concept; however, it provides no experiments or results that back up the claims that this pre-encryption can detect ransomware effectively while addressing the population drift problem in ransomware detection. Using a stochastic descent algorithm alone is not enough to deal with the dangers of evolving ransomware, as it relies on misclassification to adapt to new ransomware. One infection could prove detrimental to an organisation; therefore, it is essential to address the zero-day ransomware issue without relying on misclassification and retraining.

2.2.14 Deep Neural Networks

In this study, the authors use a network-based approach and a combination of network monitoring with a deep neural network to detect ransomware. AV's standard approach to detection today is matching binary patterns and monitoring API calls (Tseng et al., 2016). These are recognised as signature-based approaches and would not be effective in detecting novel types of malware. If ransomware changes its behaviour in terms of API calls or begins to change its binaries, then it will no longer be seen by the AV until a signature update is released (Tseng et al., 2016). API calls and binary patterns rely on detection after execution starts and cannot detect immediately when infection starts.

Using the analysis of network behaviour in ransomware, a deep neural network is constructed, which is trained on critical payloads selected from packets extracted from total network traffic (Tseng et al., 2016). This method detects infection as soon as it starts and is designed to be implemented on an SDN switch, allowing it to be easily integrated into the real-world network architecture. In terms of the network, ransomware can enter a system in many ways, including malicious emails, drive-by downloads, and compromised websites, to name a few. It is observed through the analysis of ransomware network behaviour that upon infection, ransomware will request a DNS query to the DNS server for the C&C information for a configuration file, then contact the C&C servers, which will give the ransomware further instructions on how to behave. This analysis shows that the DNS query and HTTP requests are the most important for analysing ransomware network traffic. Because domains for C&C servers can vary, HTTP requests become very important (Tseng et al., 2016).

2.2.14.1 Feature Mapping

Deep learning models use a feature engineering method that utilises carefully crafted feature detectors (LeCun et al., 2015). The neural network can start with raw data and develop features as it goes along. This approach shifts the emphasis of the user having to use feature reduction algorithms and analysis techniques to find a feature set from the user to the deep learning algorithm. Due to the advancement of deep learning, many feature engineering techniques are now obsolete (LeCun et al., 2015).

2.2.14.2 Algorithm

Deep Neural networks were chosen for the classification task due to their ability to build a better feature set than a shallow model. The dropout method was introduced to reduce over-fitting, and dropout will randomly ignore neurons during the training process, ignoring their contribution on the forward pass with weight updates introduced during the backward pass. Dropout is represented in eq.2.9.

$$O_i^h = f(y_i^h) = f(\sum_{l<h} \sum_j w_{ij}^{hl} b_j^l O_j^l) \tag{2.9}$$

Where O_i^h is the output of unit i in layer $f(y_i^h)$ is the output vector of the i^{th} layer, w are the weights, and f is the activation function. With b_j^l being the Bernoulli random vector, where $P(b_j^l = 1) = p_j^l$, the dropped neuron will depend on the Bernoulli random vector.

Root mean square propagation (RMSprop) is used to adjust the learning rate, so the step size stays on the same scale as the gradient where λ is the forgotten coefficient, $Q(w)^2$ is the gradient at w. The root square propagation is similar to weight decay; it keeps the moving average of the gradient; this is presented in eq.2.10 and eq.2.11 where for each parameter w^j ν_t is exponential squares of gradients and g_t gradient at time t along w^j.

$$\nu_t = \rho\nu_{t-1} + (1 - p) * g_t^2 \tag{2.10}$$

$$\omega_{t+1} = \omega_t + \triangle\omega_t \tag{2.11}$$

2.2.14.3 Experiments

The dataset constructed comprises 23 different types of ransomware, the most prominent being CryptoWall, TeslaCrypt, CryptXXX, Locky, CrypMIC and Cerber (Tseng et al., 2016). Most ransomware traffic was obtained on traffic analysis websites and traffic from the user's sandbox environment. The average traffic was captured by capturing network data of users carrying out everyday tasks. Initial extraction yielded over 600,000 malware packets; however, under the assumption that the essential packets were DNS and HTTP requests, this is reduced to 0.3% of the initial number down to 2631 packets. The training set used 80 samples of ransomware found in the wild between February 2015 and May 2016, whereas the test set used 77 samples of ransomware found post-June 2016. All inputs are normalised to make inputs set to equal vectors, which is achieved using the Maximum Transmission Unit. Noise caused by samples differing in length is resolved with Central-Slide shifting all inputs to the middle and padding them with zeroes. The most robust result obtained from the deep learning neural network was an accuracy of 93.92%, with a false positive rate of 0.12%.

2.2.14.4 Pros & Cons

The deep learning approach is becoming increasingly popular in terms of malware detection, and research into it is also popping up for ransomware. This approach has many positive aspects while having some limitations that must be monitored. Firstly the deep learning approach used for this model uses ReLU, which increases the speed of the training rate, slow training rates being one of the main hindrances of deep learning models. Network features allow detection before encryption can begin or even earlier in the process, giving this model an edge over those without any network monitoring capabilities. The approach taken by the authors also isolates types of network communications to look for instead of analysing massive amounts of packets; the model focuses on DNS and HTTP requests. The usage of deep learning also removes the need for the author to engineer their features. While this model does have a lot of positive aspects to it, it does have limitations. Firstly, over 70 thousand inputs make the feature engineering process undertaken by the algorithm longer; this can easily be streamlined instead of feeding raw sandbox data into the neural network. The amount of actual ransomware samples used is meagre compared to other studies. Considering deep learning requires large amounts of data, the relatively small amount of data used in this dataset may not be enough to create a robust model. The future of this system is to move execution from a static state to a dynamic execution on an SDN switch set in a natural network environment (Tseng et al., 2016). As infection with ransomware is a process, the authors would like to design an algorithm that compares the contents with the previous or following packets to enhance our detection rate (Tseng et al., 2016). The systems usage of network features alone could be further enhanced by adding behavioural and static data to fully take advantage of the power of Deep Neural Networks.

2.2.15 Long Short Term Memory (LSTM)

This Deep Learning LSTM designed for ransomware detection uses the analysis of an API call sequence to create a metric to identify the behaviour of a process (Maniath et al., 2017). Having analysed multiple ransomware families, they tend to share common behaviours. In particular, ransomware nearly always makes a short-term connection to a command and control centre, deletes shadow volume copies of files, and causes a significant system overhead by conducting massive file operations. This deep learning approach creates a model which can identify these properties and classifies a sample as either ransomware or benign (Maniath et al., 2017). The data for analysis is obtained in Cuckoo Sandbox Modified, a modified version of the cuckoo sandbox designed to circumnavigate anti-sandbox measures.

2.2.15.1 Feature Mapping

The exact features used in this model are not specified; from the specification on columns used in the dataset, 73,989 would suggest that this amount of characteristics contribute to the final feature set used. Deep learning models use a feature engineering method that utilises carefully crafted feature detectors (Le Cun et al., 2015). With feature detectors, the neural network can start with raw data and develop features as it goes along. Despite this, the feeding of 73,989 columns of raw data may be excessive and produce a massive overhead in terms of the LSTM engineering features from a massive amount of raw data.

2.2.15.2 Algorithm

LSTMs: LSTM Neural networks are a Recurrent Neural Network (RNN) capable of learning long terms dependencies (Maniath et al., 2017). LSTMs utilise a memory unit called a cell to retain information for long periods. The LSTM has four neural network layers interacting with each other deciding whether to add or remove information to a cell state. The issue with a regular RNN is that it does not look back very far regarding time steps. LSTMS will include functions which hold on to specific predictions from multiple time steps ago in order for them to continually contribute to predictions the neural network makes until those predictions are deemed to be forgotten. In this sense, an LSTM works much like the human brain in how it learns through experience.

Initially, an LSTM will determine how much previous information a cell must retain; this is decided by a sigmoid layer which acts as a forgetting mechanism; this function is described in Eq.2.12. The sigmoid activation function σ takes x_t and h_{t-1} and returns a value between 0 and 1 as output, which is multiplied by the value of c_{t-1}. An output of 1 for the sigmoid will completely retain the previous value, and a value of 0 will remove it (Maniath et al., 2017).

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.12}$$

The next step is updating the memory cell, the sigmoid layer, which decides which values to update, and the *tanh* layer, a hyperbolic tangent function that returns values between 1 and -1. The *tanh* layer creates c_t, the vector of new candidate values.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.13}$$

$$C_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.14}$$

The old state c_{t-1} is multiplied with f_t, which forgets the values we decided in the previous step. The results is then added to $i_t \cdot c_t$. The resulting candidate value will then be stored in the cell. This stage gathers new information in the cell.

The output h_t is a filtered version of the values in the cell state. The sigmoid layer will decide which part of the cell will be output. The cell's value is passed to the tanh function; the output will be multiplied by the output of the sigmoid layer.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.15}$$

$$h_t = o_t \cdot tanh(C_t) \tag{2.16}$$

2.2.15.3 Experiments

The dataset constructed contains 157 ransomware executables and benign files obtained from online repositories and default programs obtained from a fresh windows install. When executed in cuckoo modified 239 different API calls were identified in different frequencies and order. Of the 239 API calls, 38 are common in all executed samples; these are labelled between 1 and 39. Due to the difference in API sequence length from executable to executable, the API sequence is converted into an integer and appended with 0s to match the most extended sequence length. Each sequence is labelled and converted to an integer; the dataset is split 80/20, with the 20% being for testing purposes. The first column of the input is the label (benign or ransomware), with the remaining 73,989 columns being the input sequence. The LSTM model chooses the most accurate model during the training phase to be put forward for testing. The training layer is built on three layers with 64 nodes. The highest accuracy training model is used to achieve a test accuracy of 96.67%. The sigmoid function and Adam optimiser are applied to this network (Maniath et al., 2017).

2.2.15.4 Pros & Cons

Using LSTM deep learning networks to detect ransomware is a particularly novel approach considering how powerful LSTM is if used correctly. This model is a strong one yielding excellent results; however, it has limitations. The detection rate for a start is very high at 96.67%; however, it must be noted that it is not specified whether these results are on zero-day ransomware samples. The authors used sequences of API calls to train the model to detect ransomware; considering its detection rate, the sequences can make it possible for the system to detect ransomware early in its execution based on the API calls it makes. This system uses an extended timeout to compensate for ransomware that delays execution. The feature engineering system used by deep learning models significantly reduces time taken by creators of the model to reduce features and carry out manual feature engineering. While being a robust model, it does have its limitations, firstly the fact that the dataset uses common API's between all samples, which is 38; means that it relies on ransomware following particular patterns of API usage, which could be rendered obsolete in future if polymorphic ransomware arises which can alter its behaviour dynamically. Finally, this model makes the mistake of having a minimal dataset, using only 157 ransomware samples and benign files from a fresh Windows installation, which is limiting in terms of behavioural samples and the sheer numbers required to build a robust deep learning model. This system's future should firstly be trained on more ransomware samples and a more diverse set of benign programs. The benign programs only being taken from a fresh Windows installation significantly limit the dataset as a whole and give a false representation of the high detection rate. Considering little to no programs in a fresh Windows install will behave in any way close to how a ransomware executable will behave. The use of only 38 API calls could be expanded or a new metric could be proposed to identify ransomware, considering there is no solid rule that ransomware will always rely on the same set of API calls to execute their payloads.

2.2.16 Shallow and Deep Networks

Evaluating Shallow and Deep Networks for Ransomware Detection and Classification uses deep and shallow networks to detect and classify ransomware. The models aim to classify ransomware, differentiating them from benign files and between ransomware families. The use of API invocations is prominent in this research. All experiments are run up to 500 epochs with a learning rate in the range [0.01-0.5] (VinayKumar et al., 2017). The model claims 100% accuracy when classifying ransomware as benign and a 98% accuracy when classifying ransomware into their respective families. The dataset is comprised of API calls from Cuckoo Sandbox.

2.2.16.1 Feature Mapping

The exact selection of the features is not specified, while deep learning algorithms tend to have their methods of feature engineering. The alternative approaches, such as SVM, would require some feature selection process; however, this is not specified either. The features considered around 131 API calls and their frequency from the Cuckoo Sandbox logs are selected and considered (VinayKumar et al., 2017). The feature set does not appear to use feature reduction methods to reduce the feature count.

2.2.16.2 Algorithm

Artificial Neural Networks represent a directed graph in which a set of artificial neurons, generally called units in a mathematical model, are connected by edges (VinayKumar et al., 2017). The most common variants of ANNs are RNNs (Recurrent Neural Networks) and FFNs (Feed-Forward Networks). The algorithm used for this model, Multi-Layer Perceptron (MLP), is a subset of FFN. This network will contain three or more layers. The minimal three layers will contain an input layer, at least one hidden layer and an output layer (VinayKumar et al., 2017). The number of hidden layers will depend on the complexity of the data. All these units will form an acyclic graph with signals moving forward from layer to layer.

2.2.16.3 Experiments

The models are trained using backpropagation with a non-linear activation function on TensorFlow. The gradient descent calculations are accelerated by running TensorFlow in a single NVidia

GK110BGL Tesla k40 (Hinton et al., 2012); this allows for increased training speed (Glorot et al., 2011). The dataset contains seven different ransomware families comprising Cerber, CryptoLocker, CryptoWall, Maktub, Sage and TorrentLocker (VinayKumar et al., 2017). The 131 API calls used are extracted from the Cuckoo Sandbox logs. The training data will be ransomware samples found from January 2015 to March 2016. Testing data uses samples from April 2016 to May 2016. A training rate of 0.1 is used with three network topologies; MLP 1 layer with 1000 units, MLP 2 layer with 1000 and 500 units and MLP 3 layer with 1000, 500 and 250 units. All MLP network topologies have used ReLU activation functions, increasing the training rate. Dropout is used to prevent overfitting (Hinton et al., 2012); this approach randomly removes units during training in the forward pass; this helps the network learn the characteristics of benign and ransomware files. The MLP with three hidden layers performs the best in binary and multi-class classification. The MLP has an astonishing 100% accuracy rate when classifying files as either benign or ransomware. It has a 98% accuracy rate when classifying ransomware into their respective families.

2.2.16.4 Pros & Cons

This model achieves promising results and has many strengths and some weaknesses. The main strength of this approach is that the MLP model with three hidden layers manages to achieve a 100% detection rate on the test data when it comes to classifying between ransomware and benign files. The model allows the classification of files into their respective ransomware families. However, it is not explicitly mentioned how many ransomware samples or benign files are used, making it slightly difficult to judge how robust the model is. Multi-layer perceptrons (MLP) as an approach are strong when classifying problems with a large number of parameters such as this one. In terms of weaknesses, the simplicity of MLP compared with other more complex Deep Learning approaches may undo it when dealing with the next generational ransomware. The reliance on just API calls and no network or static features could be an area this approach misses out on. While being a robust model, it does have its limitations, firstly the fact that the dataset uses standard API's between all samples, which is 38; means that it relies on ransomware following particular patterns of API usage, which could be rendered obsolete in future if polymorphic ransomware arises which can alter its behaviour dynamically. Finally, this model makes the mistake of having a minimal dataset, using only 157 ransomware samples and benign files from a fresh Windows installation, which is limiting in terms of behavioural samples and the sheer numbers required to build a robust deep learning model. Future Work This system's future should be trained on more ransomware samples and a diverse set of benign programs. The benign programs only being taken from a fresh Windows installation significantly limit the dataset and give a false representation of the high detection rate. Considering little to no programs in a fresh Windows install will behave in any way close to how a ransomware executable will behave. The use of only 38 API calls could be expanded, or a new metric could be proposed to identify ransomware, considering there is no solid rule that ransomware will always rely on the same set of API calls to execute their payloads.

2.2.17 AI-Based Ransomware Detection for Digital Substations

The AI-Based ransomware detection approach proposed by Alvee et al. attempts to detect ransomware by converting ransomware binaries into 2-D greyscale images and using a convolutional neural network to classify benign and ransomware images. This research discusses ransomware detection in a substation setting and considers infection of this substation through three infection vectors. The infection vectors are remote access, physical intrusion, devices, and the substation network.

2.2.17.1 Feature Mapping

The feature mapping works differently in this research than in other research we have evaluated because of the unique approach to classification. The proposed system converts ransomware and benign executables into a 1-D grayscale array, which is then converted into a 2-D greyscale array and a 2-D greyscale image. Data augmentation is used to enhance the ransomware dataset in this research. Data augmentation is used to enhance classes with a limited population in a dataset; this reduces the impact of overfitting (Alvee et al., 2021). To enhance attack detection, Alvee et al. use rescaling and sample-wide standardisation. Two convolutional layers and two max-pooling layers are used to extract features from the processed data.

2.2.17.2 Algorithm

The concurrent neural network used in this research is multi-layered ReLu is chosen as the activation function keeps image size constant, and ReLU is good for increasing non-linearity from images with high nonlinearity (Alvee et al., 2021). The pooling layers reduce the noise impact of the features. Two fully connected layers utilise the output from the convolution process and predict the image class. Softmax is used in the layer of CNN, which normalizes the CNN output between 1 and 0, with 1 being ransomware and 0 being benign (Alvee et al., 2021). User configurable hyperparameters are determined through trial and error to find optimal settings for the CNN.

2.2.17.3 Experiments

The experimental dataset consisted of 672 benign samples and 845 ransomware samples. The ransomware samples consisted of the actual ransomware gathered by the researchers and augmented samples created for the experiments. The CNN algorithm used the data with a 90-10 split for training and testing. The CNN algorithm achieves an overall accuracy of 96.22% and is compared with the accuracy of the OpCode-based detection system proposed by (Zhang et al., 2019). The system performs well on the data it is tested on, although it is unclear why this system is only compared to one other system.

2.2.17.4 Pros & Cons

This research achieves strong results and can produce a robust ransomware detection system that uses an adaptable approach. This research uses a conversion of executables to a greyscale 2-D image, allowing the system to detect ransomware before the encryption phase of an attack. The 2-D greyscale images could be considered a static analysis technique and be vulnerable to obfuscation. An attacker understanding the mapping between the visual appearance of an executable and a benign classification could lead to misclassifications. The lack of any behavioural analysis means the ransomware does not need to execute to be detected; however, this also means the approach is vulnerable to the issues faced by static analysis methods. Ths research presented by Alvee et al. does not address zero-day and evolving ransomware either; using a static detection method could be even more susceptible to concept drift than a behavioural detection system.

2.2.18 Static Detection with LSTMs

This research uses static features combined with an LSTM to detect ransomware. The approach taken by this approach involves using the file headers of an executable to determine whether it is ransomware or benign. The file header contains a sequence of bytes, which can be broken down into information that can be used to identify a file (Manavi & Hamzeh, 2021). The theory behind this research is that the file header of ransomware is distinct and unmistakably different from ransomware.

2.2.18.1 Feature Mapping

The features used for this research are the file headers in an executable. The first 1024 bytes of the executable is considered the header (Manavi & Hamzeh, 2021), which is used for the features in the dataset. The numeric value of each byte from 0 to 1024 is noted, and the difference between the common ransomware families and benign files is visually represented.

2.2.18.2 Algorithm

The LSTM has three layers; the first two use the *tanh* as an activation function, and the final dense layer uses softmax as an activation function. The LSTM's use is justified by its ability to learn and remember long-term concepts. An in-depth look at LSTMs is provided in section 2.15.2.

2.2.18.3 Experiments

The experiments in this research provide strong results with an accuracy rate of 93.25%. The same approach with a Multi-Layer Perceptron achieves an accuracy of 91.69%. The experiments ensure that the system is tested on ransomware the system has not encountered before; therefore, it does attempt to address the issue of zero-day ransomware. The dataset used has an equal split of 1000 benign files and 1000 ransomware files; this is a reasonably large number of samples, although it is worth noting that the system will encounter more benign files than ransomware, so the dataset could be skewed towards benign files to capture the true variety of benign files.

2.2.18.4 Pros & Cons

This research produces strong results and uses a unique approach to do so. The system produced yields strong detection rates and can function as a solution to zero-day ransomware effectively. The main issue with the static system is that it relies on ransomware following pre-defined static patterns and has no fail-safe if the static characteristics of the ransomware do not match what it expects. If file headers can be altered, ransomware could easily evade this system's detection; the validity of the zero-day ransomware can be questioned. The properties of the zero-day type ransomware used in the experiments are not confirmed to be different to the ransomware the system is trained on. The system must be tested against obfuscation techniques, primarily as it relies on detecting static features.

2.2.19 IRP Ransomware Detection

This research uses I/O request packets to detect ransomware; I/O request packets are a low-level I/O system operation that is relatively new in the ransomware research space (Ayub et al., 2020). I/O requests are requests between the user and the kernel mode. If the user requests a file to open, the I/O service in the kernel mode carries out the task. The I/O logs are used with an ANN (Artificial Neural Network) to detect ransomware. The ANN is tested with three different configurations to optimise the detection rates for ransomware.

2.2.19.1 Feature Mapping

The features used are a numerical measure of different IRP metrics. The features are divided into IRP Operations, Flags and File System information. The time elapsed for each operation is also noted as a feature. The system also uses file system information for the following features: File Object, Device Object, File Name, Entropy, and Buffer Length. The features are not identified using an algorithm or feature selection method; however, the essential features for I/O operations in the lower levels have been identified in this study.

2.2.19.2 Algorithm

This research uses an artificial neural with ReLU and sigmoid activation functions; the training process is sped up using the ReLU activation function. Adam optimisation is used to reduce computational load and reduces the parameters needed for tuning. Validation loss is monitored to ensure that learning development is present in every training phase; if there is no learning development in the model, the early stopping method is used. The validation set uses 20% of the training set.

2.2.19.3 Experiments

The dataset used in this research comprises 272 ransomware samples from 18 different ransomware families, totalling approximately 90 minutes of ransomware I/O logs. The benign data comes from 11 benign users using benign programs which provide I/O data for benign file operations. Each benign user provides approximately 10 minutes of benign activity for the dataset. The accuracy achieved by the Artificial Neural Network is 99.86% as an average across all the ransomware families tested. The experimental setup also uses a zero-day approach where the neural network is trained on ransomware families not included in the test set; therefore, the test ransomware is samples not seen previously by the classifier. The research does not specify whether concept drift in ransomware is addressed and uses a training-test split of 80/20%.

2.2.19.4 Pros & Cons

This research provides strong results with a detection rate close to 100%. The neural network is also proven effective when detecting ransomware strains it has not seen before. The only flaw in the research is the small sample size of ransomware; in addition to the small sample size of ransomware, the distribution of ransomware families is highly uneven. The classifier may learn behaviour heavily biased towards some ransomware families, which could lead to issues when ransomware evolves and uses I/O behaviour that may deviate from the I/O behaviour of large parts of the dataset.

2.2.20 Siamese Neural Networks

The research proposed by Zhu et al. proposes a siamese neural network that can detect ransomware and class it into its respective family. The Siamese neural network uses entropy features extracted from ransomware binary files. The binary features train and refine a pre-trained network to achieve the most accurate results. The novel aspect of this research is the Siamese neural network, which takes two samples from the same class and outputs the features of each sample it learns; this method helps find the commonalities between different samples of the same class.

2.2.20.1 Feature Mapping

The second novel aspect of this research is the use of entropy features in binary ransomware files. The idea behind using the entropy of the ransomware files comes from the fact that image features can be vague, and the accuracy of image features can be lost in conversion processes (Zhu et al., 2022). The entropy features find features similar from ransomware to ransomware and should allow the classifier to identify ransomware from common elements in their code. Finding new variants of ransomware should be easier using this method as there tends to be reused or common code between variants of ransomware from the same family. Entropy features are not susceptible to white noise and are not too sensitive to spatial distribution changes; code obfuscations or modifications would represent spatial distribution changes to fool detection systems.

2.2.20.2 Algorithm

A Siamese neural network is a neural network that is split into two sub-networks where the weights and hyperparameter settings are the same. The sub-networks take one sample from one class, and the out is the features it learned as common between the two samples. These features are used in the feature embedding of the fully connected layer, while a loss function is used to predict whether the samples processed by two sub-networks belong to the same class or not (Zhu et al., 2022). The proposed system can predict if features learned through different instances are complementary to instances within the same class. Using Siamese neural networks, the proposed system can predict that new ransomware belongs to the ransomware class by identifying common code in existing ransomware samples.

2.2.20.3 Experiments

The dataset comprises 1048 ransomware samples and an unidentified number of benign files; the test-train split is set to 80/20, and the system is benchmarked against lightweight deep learning models like DNN and RNN and heavyweight deep learning networks such as InceptionV3, Resnet50, and VGG16. The proposed model performs significantly better than the benchmark algorithms; however, its detection rate is 88.7%. The detection rate exceeds the benchmarks, but this detection rate seems low compared to existing research. It is unclear what kind of benign data is used and how it differentiates from ransomware.

2.2.20.4 Pros & Cons

The research proposed by Zhu et al. is an interesting concept and allows the system to detect ransomware based on its binary files; this infers that the ransomware can be detected before the encryption begins. The main issue with this method is the computational load deep learning algorithms can have and whether the reward is worth the computational cost. Some detection methods provide high detection rates and focus on zero-day ransomware that does not incur the cost of using a deep learning architecture. In addition to using a deep learning method, this research relies on static detection methods; static detection tends to be the easiest to evade, and it would be helpful to have a dynamic detection engine in addition to a static detection engine.

2.2.21 Analysis Framework

Poudel et al. developed a framework for the Analysis Framework that performs multi-level analysis on ransomware and benign. The multi-level analysis involves analysing binaries, assembly code and function calls such as API calls. The analysis framework uses a combination of binary, assembly and API features to make an informed decision on ransomware detection; the framework uses different machine learning classifiers. This research has achieved strong results with detection rates over 90% for almost all machine learning algorithms.

2.2.21.1 Feature Mapping

The feature set consisted of reverse-engineered binary features, assembly code, and DLL features extracted from the executables. The feature extractor extracts the DLL and assembly features from the ransomware or benign executable. Cosine similarity is used to measure the similarity of two samples based on their DLL and assembly features. Cosine similarity finds which features provide the most discrimination between benign and ransomware, as shown in eq.2.17.

$$\cos\varphi = \frac{P \cdot Q}{|P||Q|} \tag{2.17}$$

2.2.21.2 Algorithm

The most successful approach in this research is AdaBoost, an iterative ensemble learner. The AdaBoost learning method learns from the mistakes of previous iterations of the learner that are considered weak and will turn these classifiers into stronger ones. In a random forest, boosting would be applicable for classifiers with a non-satisfactory accuracy. All the trees in the random forest learn from each other's incorrect classifications to build a stronger classifier. The Gini impurity is shown in eq.2.18. Gini impurity measures the likelihood of misclassification based on a split. The lower the Gini impurity, the better the split. Gini Impurity is shown in eq.2.18 where Y is the total number of classes and p_y is the probability of picking a data point with class y.

$$Gini(n) = \sum_{y}^{|Y|} p_y(1 - p_y) \tag{2.18}$$

2.2.21.3 Experiments

The experiments used a dataset of 302 malware samples(Poudel et al., 2019). The experiment determined whether a machine learning classifier could distinguish ransomware from other types of malware. The total number of ransomware samples used was 178 from this dataset, with an uneven split from various families, most of which consisted of Locky and Teslacrypt. 178 benign samples act similarly to ransomware to further harden the classifiers' against false positives. There are three sets of experiments carried out, one with assembly code features, one with DLL features and the last with both sets of features combined. The third round of experiments is the most successful, with an accuracy of 97.95% with Random Forests; strong results are also achieved with J48 and Random Forests, using AdaBoost. Locky and TeslaCrypt are used in a test set that tests the classifier's ability to determine ransomware families; the most accurate algorithm for this is the Random Forest with AdaBoost.

2.2.21.4 Pros & Cons

This research has strengths and weaknesses. The framework uses various learning to identify the best detection solution. The model is a framework that increases flexibility regarding the algorithms used; it is up to the researchers to identify the most suitable algorithm for their features and data. The first issue with this study is that very few ransomware samples are used to train the classifier. While using various families, the researchers make the mistake of heavily weighting their samples towards just Locky and TeslaCrypt; this is not justified, and it would be logical to include a variety of ransomware families, especially as Locky and TeslaCrypt are not the only prominent ransomware threats. The features used are restricted to assembly and DLL features; these could be expanded to incorporate API or network features to aid early detection.

2.2.22 Feature Selection Based Detection

This research focuses on network behaviour and how it can be used for ransomware detection, which bases its methods on big data (Chang et al., 2018). This approach uses a BiFlow concept to replace packets because data size is reduced significantly; this reduction is a ratio of 1000:1 when using BiFlows over packet data. This system uses six selection algorithms for classification purposes and an additional decision tree model to enhance the performance of the intrusion detection system.

2.2.22.1 Feature Mapping

This research uses 36 network features manually identified by analysis of PCAP files. The algorithms were matched to the features based on six characteristics: gain ratio, information gain, correlation ranking, OneR feature, ReliefF ranking, and symmetrical (Chang et al., 2018). The six feature selection algorithms were used together; the resulting scores are normalised to yield an aggregate score for each feature. This system uses multiple characteristics for feature selection algorithms in an ensemble-like method to get the optimal feature set.

2.2.22.2 Experiments

The dataset uses a combination of Cerber, Locky and benign software. The experiments in this research are not extensive but use different amounts of attributes, depths and numbers of leaves. The most successful configuration was using 25 of the 36 attributes, with 19 leaves and tree size of 31, achieving a precision rate of 90.62%. There is a variation of around 2% for precision between the different configurations used for the decision trees, which vary in leaves and tree depth.

2.2.22.3 Pros & Cons

This study focuses on the feature selection process and alteration of network data. They transform packet data to BiFlow format to reduce data size. The main issue with this research is simply the lack of information about their data and approach. The reasoning behind only selecting Locky and Cerber is not provided, and the dataset is not described. There is also a lack of justification for their selection of features. Using network features alone might not be a complete solution as there are behavioural and static components to ransomware detection which could be considered.

J48 decision tree implicitly performs feature selection; this could make a feature selection algorithm an unnecessary computational expense. The lack of a feature selection algorithm in this research paper could be attributed to this; however, this is not explicitly mentioned. Not using a specialised feature selection algorithm could allow the inclusion of features with no benefit for the classifier. Decision trees can be complex because of how varied ransomware can be; this study uses Cerber and Locky, which means the trees will be specialised in detecting these two families. The diversity of ransomware has not been captured; not capturing the vast diversity of ransomware could be an oversight that leads to misclassifications. Expectations in decision trees are critical in the classification process; while expectations are realistic, the classifications are strong; however, if expectations are irrational, this can lead to errors. There is no mechanism to compensate for unreliable predictions that decision trees can be prone to, especially if there is a shifting concept. Decision trees can be adapted to shifting concepts through local replacement of nodes; however, this is not addressed in this research.

2.2.23 Machine Learning-Based File Entropy Analysis

This ransomware focuses on detection in backup systems, which is ransomware infections in files stored in cloud backups. This research uses the idea that cloud services cannot recognise that the files they back up are encrypted by ransomware. This research proposes using entropy analysis of infected files to differentiate between infected and regular files. The primary motivation is to protect backup systems and backup files; if a ransomware infection occurs, a backup is crucial to system restoration. Encrypted and infected files must be kept away from backup systems. This research has very few negatives besides the need to ensure that the system is trained on an extensive range of ransomware, ensuring they can keep track of the possible effects different types of ransomware can have on files.

2.2.23.1 Feature Mapping

This research does not use standard features like the other studies we have reviewed. Since it is not a traditional detection system that monitors a network's endpoints, it does not necessarily need to have a static, network or behavioural features. Instead of these standard features, the system uses entropy measure, which measures the entropy between the files incoming from the endpoint and the backup server's corresponding backup. Through the entropy measure, it is possible to detect whether the incoming file is infected by ransomware. If system resources permit, machine learning models will be added to this detection system, and the synchronised files in the backup server will be used as a test set to detect ransomware infections (Lee et al., 2019).

2.2.23.2 Algorithm

Entropy is a measure of randomness or uncertainty in the outcome of an event. Entropy can be derived in many different ways. The formula for measuring entropy is shown below in eq.2.19, where p_i is the probability of a class.

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{2.19}$$

The backup system will store the data for the user, and when detecting ransomware, the entropy reference value of each user's backup data is measured. The detection module detects infected files by comparing the measured entropy of the synchronised files with the reference value of the file format received from the backup system (Lee et al., 2019). It is also proposed to store the entropy reference value as the average of a collection of files. It should be noted that the entropy characteristics for each user can differ, so this should be considered when calculating these figures. The backup system learns and measures the entropy of the files stored in the backup system for each user, thereby deriving an optimal reference value specific to the user files, which leads to more accurate detection of the files infected by ransomware (Lee et al., 2019). The data used for the machine learning models in the proposed system consists of file format, the entropy measured by the most common value estimate, the entropy measured by the collision test estimate, the entropy measured by the Markov test estimate, the entropy measured by the compression test estimate, and whether or not the file is infected by ransomware (Lee et al., 2019).

2.2.23.3 Experiments

The results obtained in these experiments appear very impressive, with most of the machine learning models obtaining test set detection rates of 100%. The dataset comprised 1200 files, with 100 examples of each file format and 100 encrypted examples of each file format. The results are promising as the dataset is varied and not small; it comprises over 1000 files.

2.2.23.4 Pros & Cons

This research provides promising results, with a near 100% detection rate of infected files on all the detection algorithms they have used. This aspect of detecting ransomware has not been covered well, and this solution does protect backup systems from taking on encrypted or infected files because backups are the last line of defence against ransomware. Backups must be protected from infection and be overwritten with encrypted files.

This research is different from the other research studies that have been reviewed. The main focus of detection is on file entropy and how the underlying machine learning algorithm interprets the differences in entropy to detect files encrypted by ransomware. The main weakness of this study is that there is no consideration of how ransomware encryption may be altered to avoid a system like this. The entropy calculations between benign and infected files will rely on specific differences and patterns which define how an infected and encrypted file is identified. It remains to be seen whether the system would resist attempts to mask encrypted files as regular files.

2.2.24 Digital DNA Sequencing

This research uses an approach which uses DNA sequencing for ransomware detection. The ransomware sample data is converted to resemble a DNA sequence to detect ransomware before the initial infection stage. This approach is because current ransomware detection studies rely on the analysis of ransomware networks or behavioural patterns, which imply the attack is already taking place during detection. Code obfuscation makes it difficult to detect based on code. It is known that only portions of the code are revealed during certain parts of executions (Khan et al., 2020).

2.2.24.1 Feature Mapping

This system uses MOGWO and BCS to select the framework's optimal features. MOGWO builds on GWO, which is an algorithm inspired by grey wolves. GWO searches for the optimal method for hunting prey. MOGWO divides the objective space into a grid which contains all possible solutions. The archive decides if a solution should be added or not depending on whether the solution dominates any solutions already in the grid. BCS builds on CS (Cuckoo Search) algorithm, which is adapted from the reproduction strategy of cuckoo birds. Along with these two methods, the accuracy determines which function is maximised and which features are chosen.

2.2.24.2 Algorithm

The novelty of this study is the use of digital DNA sequencing. A synthetic DNA representation of a digital artefact rather than biological DNA. A digital artefact is represented by only the characters A, G, C, T where adenine (A), guanine (G), cytosine (C), and thymine (T) molecules, commonly referred to as "bases" (Khan et al., 2020). An example of this process of converting a common dataset to a DNA sequence; a sample with the binary features '0 1 0 0 0 1 0 1 0 1 0' would be converted to 'CGAACGCGCG' which is a digital DNA sequence containing features from the "bases". A DNA sequence should follow constraints considering H-Measure, continuity, melting temperature, and GC Content. DNAct-Ran uses three constraints for the DNA sequences it produces. The $TmConstraint$ as represented in eq.2.20, the $GCConstraint$ as shown in eq.2.21 and the $AT_GCRadioConstraing$ as shown in eq.2.22.

$$Tm = 4°C(G + C) + 2°C(A + T) \tag{2.20}$$

$$GCC = \frac{G + C}{A + T + G + C} * 100\% \tag{2.21}$$

$$(A + T)/(G + C) \tag{2.22}$$

Using the constraints defined by the DNAct-Ran system, a common dataset is converted to a synthetic DNA dataset and, from there, machine learning algorithms are trained on the synthetic dataset to detect ransomware.

2.2.24.3 Experiments

The detection algorithm uses an active learning approach which is broken down into four strategies;

· Query strategies based on uncertainties, where instances with the lowest prediction confidence are queried.

· Query strategies based on disagreement which queries the instances on which the hypothesis space has the most disagreement degree on their predictions.

· Minimise the expected variances and error by labelling the instances on the pool of unlabelled instances.

· Exploiting the structure information among the instances.

The initial classifier uses a linear regression model, which will be used with the active learning principles to train and predict the classification of the data. Once the DNA sequence of the ransomware is learned, the individual families are analysed by machine learning algorithms. The dataset used for this study is the dataset from the EldeRan system, which contains 1524 samples, split into 582 ransomware and 942 benign software samples. The testing was carried out on 150 ransomware and 150 benign samples. The active learning solution achieved an overall accuracy of 87.91%, which is promising because of the novelty. However, detection studies achieve much higher early detection results with much less complex systems.

2.2.24.4 Pros & Cons

In terms of positives, this study uses a highly sophisticated DNA-based detection system focusing on early detection. The detection rates are on the lower end of the scale with an accuracy of 87.9%, which is lower than the majority of the studies that have been reviewed. This research was published in 2020; however, it uses a dataset with software from 2013-2015. The age of the data would be considered unsuitable, especially because through our experiments on the same dataset, we have seen significant concept drift in ransomware since 2015. The data representation as DNA is a novel angle on this topic; however, it is unknown how this approach will react when faced with significant concept drift. It is unknown how the changes represented by concept drift will translate to a system that uses a ”DNA” approach. In terms of future work, the system would need to be trained on more up-to-date ransomware, and it would need to present a higher detection rate. The linear regression algorithm could be changed to something more suitable for ransomware data. Considering that this is the only system we have reviewed that uses linear regression, it may be wise to alter the data to try the machine learning algorithms others have used.

The most successful algorithm used by the DNA-based system was Random Forests compared to Naive Bayes and the Decision Stump. The random forest will be able to capture complex patterns displayed by a variety of ransomware samples in the EldeRan dataset, regardless of how old they are. The main issue faced by this DNA-based approach is its complexity while using Random Forests with a synthetic DNA dataset. DNA is highly diverse and complex; the additional complexity of representing ransomware as DNA coupled with a random forest can represent a high level of complexity. It is uncertain whether the complex and specific patterns generated by the synthetic DNA dataset may cause over-fitting in decision trees. However, because there will be multiple trees in a Random Forest, each tree will have captured different patterns and characteristics in the synthetic DNA which define ransomware; this makes the trees more robust and stable when dealing with unseen ransomware, as concept drift will not affect every tree at once.

2.2.25 Resilient Machine Learning

This research paper evaluates whether machine learning approaches are resilient against ransomware attacks. The authors of this paper test the resiliency and trustworthiness of machine learning algorithms. The resilience of machine learning algorithms is tested using a generative adversarial network (GAN). The GAN generates dynamic features that reduce black-box ransomware classifiers' efficacy (Chen et al., 2019). The main objective of using GAN is to demonstrate that machine learning classifiers must be prepared for what the GAN samples represent. The quality of the GAN sample is evaluated against their statistical similarity with actual ransomware samples. The sample space containing the GAN was investigated; the research provides insight into why these samples cause machine learning models to degrade.

2.2.25.1 Feature Mapping

The execution log of each sample contains a timestamp, event name, targetted file, and file entropy. The four file actions are ”create”, ”delete”, ”rename” and ”change”. The entropy level of the file is combined with the event of a file change (Chen et al., 2019). Each execution log is a sequence of events, and a sample's maximum length is capped at 3000 events. Samples with a sequence of events less than 3000 are padded with zeros, so all samples are the same length.

2.2.25.2 Algorithm

Adversarial Machine Learning attempts to fool a machine learning model by giving the model an input designed to deceive the classifier. These machine learning models are applied to the real world, and the data they deal with may differ or evolve in the case of malware; malware can easily evolve to behave differently from expected. Once these classifiers are identified, adversaries can use malware designed to evade a classifier that does not follow the statistical assumptions made by the model in its training phase.

A GAN (Generative Adversarial Network)is a connected neural network architecture for discriminator and generator (Chen et al., 2019). The generator produces samples from the generated distribution P_G, which is supposed to be as close to the real distribution P_R. The discriminator will classify the samples generated by the generator and decides whether the sample is from P_G or P_R. The discriminator aims to find the real from the fake, and the generator aims to fool the discriminator. By the end of the training, the generator is supposed to maximise fooling the discriminator. These generated samples are put through a quality assessment phase, a sample-based adversarial quality metric. Each sample will be quality tested in order to be sure it is fit for use in testing. This process is repeated until a defined number of samples which pass the quality assessment are generated.

2.2.25.3 Experiments

There are seven models trained on the initial dataset. The models used are Text-CNN, XGB, LDA, Random Forest, Naive Bayes, SVM-linear, and SVM-radial. The ransomware samples are from late 2017 to 2018. The training set contains 1292 benign samples and 3736 ransomware samples. The test set contains 324 benign samples and 934 malicious samples. The training-to-test ratio is 80% to 20%. On the test set, the detection rates were promising, with Text-CNN achieving an accuracy of 98.9% with a true positive rate of 99.98%. The highest performing algorithms were tested on the adversarial samples the GAN produced; the results showed significant degrading with Text-CNN detecting none of the adversarial ransomware samples, XGB detection rate falls to 12.73%, random forest falls to 36.35% whereas SVM radial detects 100% of the adversarial samples. The experiments have proved the lack of resiliency in machine learning models and how the models can be reinforced with the samples generated by the GANs.

2.2.25.4 Pros & Cons

This study takes steps to address a growing issue in the future: adversarial machine learning. The methods used to prove the weaknesses in machine learning have effectively proved an interesting point. The experiments did focus on ransomware, with the datasets being heavily skewed towards ransomware with relatively little benign software. In a real-life scenario, the ratio of ransomware to benign files a system would encounter would be the opposite. It would be wise to include more diverse benign files in training. The varied samples will give the model a good spread of ransomware families, which display diverse and complex behavioural patterns. The features are somewhat limited as they do not take any network or static features into account but use I/O features. The reliance on file entropy and file interaction would suggest needing the ransomware to begin execution before detection. The most important theoretical issue with this study is the need for the generation of potential samples. When it comes to ransomware, there is no guarantee that the changes or evolution of the ransomware can be predicted. Realistically the GAN has to be heavily tuned and specialised to operate within strict parameters that would ensure it behaves like actual ransomware while displaying some behaviour which would make it appear benign to a classifier. The next issue with this approach would be that when altering the statistics of each sample, the actual behaviour it represents must be taken into account. Regardless of the potential issues, the overall concept of the work is promising and raises valid issues.

2.3 Malware Detection

2.3.1 Malware detection training based on GAN-generated zero-day malware

The research proposed by Won et al. explores the use of specialised training for zero-day malware detection. The specialised training involves generating analogous zero-day malware images from an already existing dataset of malware data. The GAN-generated malware is diverse and presents the threats posed by actual zero-day malware; the discriminator learns various features associated with malware using the real and generated malware samples.

2.3.1.1 Feature Mapping

The Frechet Inception Distance (FID) is used as a metric to evaluate the generator convergence. The FID uses pre-trained Inception v3 networks to extract features from the real and generated malware samples (Won et al., 2022). Then model the data distribution for extracted features using a multivariate Gaussian distribution with mean μ and covariance Σ. The FID between the real images x and generated images g is computed below (Won et al., 2022).

2.3.1.2 Algorithm

The algorithm used for malware detection is the discriminator, which is trained without the malware generator which uses adversarial training. The discriminator is trained using both real and generated malware images. The malware images are generated based on the real malware images that are represented as RGB colour images. The objective function for the discriminator is shown below in Eq.2.23, where genuine samples are denoted as P_{data} and generated samples are P_{gen}, where c represents the class and the features are represented by x.

$$L_D = -\epsilon_x \; p_{data}[logp(c|x] - \epsilon_{\hat{x}} \; p_{gen}[logp(c|\hat{x})] \tag{2.23}$$

The generator is designed to generate zero-day malware based on the real malware images in the dataset, and the discriminator is trained separately from the generator.

2.3.1.3 Experiments

The dataset contains 10,868 malware samples that are converted into RGB image files. The dataset contains various malware, including worms, trojans and downloaders. The experiments are split into two parts, the first on the existing malware and the second on the analogous zero-day malware generated by the GAN. The generated zero-day dataset contains 5543 malware samples. The test phases are both successful, with the classifier achieving an accuracy above 99% in both test phases; this is particularly impressive considering the exposure to a high volume of zero-day data.

2.3.1.4 Pros & Cons

Overall, this research provides strong results and has a much-needed focus on zero-day malware attacks. The use of GAN to generate zero-day samples in bulk gives the classifier a good chance of identifying real zero-day threats. Considering the classifier relies on RGB images of malware and benign files, it can be assumed that the detection metrics are static; therefore, the detection must occur before execution. The main issue with static detection methods is that they can be more vulnerable to obfuscation and evasion than dynamic methods. The second major issue with this research is the type of malware used in the dataset; there could be a higher diversity of malware to provide more robustness for detecting different types of malware.

2.3.2 Flow-Based Malware Detection

The flow-based ransomware detection approach proposes using packet flow characteristics instead of port numbers and protocols to detect malicious traffic generated by malware. Yeo et al. use netmate to extract flow information from packet data to create the dataset. The experiments in this research used different machine learning algorithms, the most effective being the random forest algorithm.

2.3.2.1 Feature Mapping

The feature selection program used is netmate; netmate is a feature extractor that takes a network capture file and converts the network capture flow into 35 statistical features. A flow is a sequence of packet IP addresses and port numbers from a source to a destination. The feature extraction involves using a z-score technique to normalise all statistical features (Yeo et al., 2018). The Z-Score measurement defines how far an item is from the mean; therefore, the Z-Score is used to calculate how far each feature of a sample is from the overall mean of that feature in all samples. The calculation for the Z-Score is shown below, where μ is the mean and σ is the standard deviation.

$$Z = \frac{x - \mu}{\sigma} \tag{2.24}$$

2.3.2.2 Algorithm

This research uses four algorithms, and the most consistent is the random forest algorithm. The random forest configuration is a maximum of 128 trees with 16 features per tree and uses the Gini index to split the trees. The experiments also use Convolutional Neural Networks, a Multi-Layer Perceptron and an SVM algorithm; however, these approaches are not consistent with detecting all of the malware used in the dataset.

2.3.2.3 Experiments

This research uses four machine-learning algorithms to detect malware and a dataset of 9 malware families. The test and training splits are not evident; however, the accuracy achieved by the random forest algorithm is 93% overall on all malware families. The number of benign samples used is unclear.

2.3.2.4 Pros & Cons

This research proves that machine learning algorithms can use network flow features to detect malware. The detection rate achieved with random forests is impressive; however, the dataset is limited. Only nine malware families may be an oversight considering the diverse malware ecosystem in the online space.

2.3.3 IoT Network Malware Detection

The research proposed by Pudukotai et al. acknowledges that IoT is expanding, creating a greater possibility for attacks to infiltrate and spread through networks. Detection and quarantine are not enough to prevent the propagation, and traditional control theory does not provide real-time malware control; in addition to the lack of real-time malware control, their difficulty in decoupling malware detection and network performance (Pudukotai et al., 2019). This research proposes HaRM which uses Hardware Performance Counter (HPC) values to detect malware and benign files. The proposed framework uses an execution time of 10ns and can achieve a strong detection rate.

2.3.3.1 Feature Mapping

This research uses HPC traces to detect malware, as software-based malware detection can incur overheads and has a critical latency issue. There is a limited number of available HPCs events in modern microprocessors, with four at most for IoT devices. A few microarchitectural events equal to the number of available HPCs can be monitored simultaneously (Pudukotai et al., 2019). A subset of HPCs representing the most critical malware detection features is selected. The Pearson Correlation algorithm determines the most discriminating features for malware detection. The Pearson Correlation not only determines the most discriminating features but also isolates the feature combination of features that provide the most discrimination and have the least inter-feature correlation.

2.3.3.2 Algorithm

The research uses four machine learning algorithms to detect malware in IoT; these algorithms include Multi-Layer Perceptrons, OneR, Logistic Regression, and Jrip. The highest accuracy is achieved with the Multi-Layer Perceptrons; however, the MLP uses the most power and has the highest level of latency. OneR achieves similar performance with 2, 4 and 8 HPC while having a relatively high detection rate and low latency. OneR is a simple prediction algorithm that generates one rule for each predictor and then selects the rule with the lowest error as its one rule.

2.3.3.3 Experiments

This research uses 3000 malware and benign samples as part of its dataset and uses a k-fold cross-validation approach to evaluate the machine learning algorithms. The OneR algorithm achieves the most stable results by finding the balance between detection, power consumption and latency. This approach is also able to increase network throughput 200%.

2.3.3.4 Pros & Cons

This research attempts to solve the issue of rapid malware detection in IoT networks and provides a solid solution with a strong detection rate. The HaRM system is trained on various malware; however, it is unclear how these malware samples are adapted to IoT. The system does not specify what IoT devices are exempt from its detection capabilities. In addition to a strong detection rate, the HaRM system considers power usage and latency. It is essential to consider these factors as latency in detection can be the difference between detecting malware early or too late. The HaRM system uses a detection period of 10ns, which is a minuscule period when detecting a malware infection. Rapid detection can prevent malware propagation through a network if one system is infected.

2.3.4 Semantic-Based Malware Detection

This research acknowledges that malware has become an increasingly dangerous threat online and antivirus and patching are not always effective in detecting constantly advancing malware. The system proposed is GuardOL which can be used for online malware detection using processor and field-programmable gate array (FPGA) data (Das et al., 2016). GuardOL captures the high-level semantics of malware. GuardOL uses the frequency of system call patterns known to occur in malware and benign files and then trains a Multi-Layer Perceptron to classify malware and benign files using these features.

2.3.4.1 Feature Mapping

The features used in this research are low-level and use OS resources like the file system, memory and process information. Das et al. manually list the most commonly used system calls that malware use during their execution. The system calls that are security critical and modify the state of the OS resources are identified and used as part of the feature set, which is narrowed to the 64 most commonly used by malware. This research uses a grouping of the gathered features called a frequency-centralized feature construction model (FCM) to capture the semantic behaviour of malware. FCM follows the steps of grouping system calls in sets, tracing the memory mapping of the files, tracing the source file of write operations and counting the frequency of the resource-critical system calls.

2.3.4.2 Algorithm

The main algorithm used in this research is the multi-layer perceptron (MLP), which provides the best detection and lowest false positive rates of the tested algorithms. The exact configuration of the MLP is not described; however, its use allows for parallel calculations, and it can leverage FPGA to perform multiple parallel calculations. The early detection with the MLP works with the FCM approach described in the feature selection module.

2.3.4.3 Experiments

The dataset comprises 472 Linux malware samples taken from Virusshare and VXHeaven. The malware types included in this dataset are backdoors, exploit kits, flooders, hacktools, net-worms, rootkits, trojans and viruses. The data set was split 70/30 for testing and training. The detection rate achieved by the GuardOL system was 97%, and 46% of the malware was detected within the first 30% of its execution. 97% of the samples are detected after full execution; however, it can be argued that complete execution would not be desirable as the host system can be damaged before the malware is detected. The execution of the GuardOL system is the syscall extractor, feature constructor, classifier training and run time detector.

2.3.4.4 Pros & Cons

The GuardOL system proposes a novel early detection system for malware. The GuardOL achieves strong detection rates and a low false positive rate; however, as an early detection system, the

GuardOL is flawed as it can only detect 46% of malware in tests during the early stages of execution. The GuardOL system uses system calls and the interactions between malware and hardware to detect malware which is a novel approach and does not rely on traditional antivirus detection techniques. The grouping of features is an interesting idea; however, it narrows the number of system calls used, and this could give room for attackers to input system calls that benign files use to obfuscate the presence of malware. Overall the hardware-based detection system is promising and can be used as part of a more extensive system that can use a complete feature set.

2.3.5 API Sequence Based Detection

This study takes an approach; many of the other reviewed studies have not considered differentiating ransomware from other malware, not just benign software. This research uses API execution sequences which are converted to n-gram sequences. These n-gram sequences identify and differentiate ransomware from other malware and benign software. Each element of the input can be represented as "1" if an n-gram appears in the n-gram sequence or as "0" if the n-gram is not present in the sequence (Seong et al., 2019).

2.3.5.1 Feature Mapping

The features used in this research are Windows API calls. The sequences of API calls of each executable are gathered and then converted to n-grams. The n-gram sequences differentiate between malware, ransomware and benign software. Extraction of the Windows API calls is done by the Intel PIN tool. CF-NCF (Class Frequency Non-Class Frequency) values are calculated for each sample, and these values act as the weights of each element.

2.3.5.2 CF-NCF (Class Frequency-Non-Class Frequency)

CF-NCF acts as an indicator for classification models based on the Term-Frequency-Inverse Document Frequency. This technique is used to emphasise each class's features, giving the further ability to differentiate between malware, ransomware and benign files. CF-NCF calculates weights on an element in a class to provide higher accuracy in classification experiments (Seong et al., 2019). The calculation of this equation can be seen in eq.2.25, 2.26 and 2.27.

$$CF(s, C) = f(s, c) \tag{2.25}$$

$$NCF(s, N) = \log(\frac{1}{0.001 + f(s, N)}) \tag{2.26}$$

$$CF - NCF = CFxNCF \tag{2.27}$$

In these equations, s is an n-gram, $f(s, C)$ is the number of times that the n-gram s occurs in the n-gram sequence C of a particular class. N is the n-gram sequence of the other classes, and 0.001 serves to prevent the denominator from becoming zero (Seong et al., 2019).

2.3.5.3 Experiments

The experiments used six machine learning algorithms: Random Forests, Logistic Regression, Naive Bayes, Stochastic Gradient Descent, K-Nearest Neighbours, and SVM. The dataset comprised 1000 ransomware samples, 900 other malware files and 300 benign files. Only 2064 of the 2200 samples were used due to improper execution of some of these files. The API invocations were limited to 50,000 to limit the size of the samples and regulate the number of n-grams produced. In regards to n-grams, the best results were obtained when the n-value was set to 4, and it was Random Forests which displayed the best rate of accuracy with 97.29%. When detecting ransomware against other malware, Random Forests had the highest accuracy rate of 96.61%. The algorithms were from the Scikit learn libraries.

2.3.5.4 Pros & Cons

This study focuses on distinguishing ransomware from other types of malware. Ideally, detection should stop a ransomware attack before infection or encryption, but in the event of some encryption, the type of reactive measures that need to be taken out if ransomware attacks your machine will be different to other types of malware. Detecting a ransomware attack on one machine can stop it from spreading through a network, which is a crucial aspect of detection. The permanent nature of ransomware attacks would make a system like this useful. The results obtained are promising; however, this study lacks any future-proofing and acknowledgement of evolving ransomware. The dataset used in this research is skewed heavily towards ransomware and malware, with only 300 benign files. The model is far more likely to encounter benign files than malware. The lack of variety in the benign samples means the model could display high false positives when coming across unseen benign files; including a high number of benign files is essential, so the model captures at least some of the diversity in benign files.

2.3.6 Two Stage Ransomware Detection

This research uses a two-staged approach to detecting ransomware (Hwang et al., 2020). The authors acknowledge the increasing diversity in ransomware and the difficulty in detecting ransomware. The Markov model focuses on the API call sequences of the ransomware, and the Random Forest is used to model the remaining data to reduce the false positive and false negative rates.

2.3.6.1 Feature Mapping

This study uses Windows API calls; it is not limited to a specific set or number of API calls. The researchers acknowledge that each executable has varying numbers of API call sequences; however, these are not limited or normalised. The additional features extracted from the analysis tasks are registry operations, file system operations, strings, file extensions, directory operations and dropped file extensions.

2.3.6.2 Algorithm

A Markov chain is a state machine system in which events transition from state to state. The Markov chain operates on the assumption that the future states are all fixed regardless of the present state, and no possibilities outside these states can occur. The probability of future events depends on the state of the previous event. In this case, the state space is 303 of the most common API calls used, as decided by the authors. The Markov chain is built to predict a classification using the built probability model, which uses the sequence of API calls.

2.3.6.3 Experiments

The dataset used in the experiments contained 2507 ransomware samples and 3886 benign samples. The Markov chains are used to decide on a classification based on the API call sequences of a sample, and the Random Forests to classify based on the remaining features. The Markov chains provide a low rate of false negatives, and the Random Forests provide a low rate of false positives. The Markov chain will classify in the first stage, and the Random Forests will classify all the samples the Markov chain classifies as benign to provide a second layer of security. During the experiments, the threshold value of the Random Forest was altered to extract the best results. With a threshold of 0.2, the highest achieved is 97.28%.

2.3.6.4 Pros & Cons

This study takes a double-layered approach to detecting ransomware. The Markov chains combined with a Random Forest have shown promising results, although it is not specified on which samples the Random Forest is trained. The Markov chain may be limited regarding ransomware, which will change over time as its future states are set, and there are limited futures available for each state to go to next. The reliance on specific API calls may be a risk considering any drastic changes in how the ransomware uses API or changes in the sequence of API calls will compromise the Markov chain's ability to predict what will happen next. In terms of the features and dataset used, this research is comprehensive and covers static and dynamic features. The only lacking component is a set of network features. The dataset uses a more significant amount of benign software than ransomware, ensuring the models will be trained to identify diverse benign files from the many families of ransomware.

2.3.7 Multi-Tier Streaming Analytics Model

This paper proposes a hybrid machine learner model, a multi-tiered streaming analytics model (Zuhair et al., 2020). The model classifies each sample into 1 of 14 ransomware families by using 24 different static and dynamic features. The model classifies ransomware versions into their ancestral families numerically. For ransomware which descends from multiple families, the model fuses the multi-descendant families statistically (Zuhair et al., 2020). This model attempts to categorise 0-day models into their respective ancestral family or identify if it has multiple ancestors. This research uses the most extensive dataset of ransomware reviewed in this paper, using a dataset of 35,000 ransomware samples, 500 malware samples and 500 benign files.

2.3.7.1 Feature Mapping

The feature set used in this research includes 14 dynamic features and ten static features. The dynamic features work through accessing queries of files and directories, read/write/delete operations, editing the system's digital certification, modifying system files' headers, and the entropies of buffering data. The static features involve file content and paths and spoofing the links to particular directories (Zuhair et al., 2020). Each sample will have a classification of "01" to "14" so the family it belongs to can be identified. The label "00" is assigned to benign software and the label "99" to other malware types. Naive Bayes is impractical with large feature sets and heterogeneous trait values. However, it spends a short amount of computation time learning the training vectors of traits and predicting their actual classes using Bayes' probabilistic theorem, assuming that all the examined traits are independent of each other (Zuhair et al., 2020). Thus, NB is applied by HML to trace the predictive classes of all overlooked traits in the indecipherable nodes of DT, which optimises the adaptive classification.

2.3.7.2 Algorithm

The proposed learner, HML, is a hybridised version of Naive Bayes and Decision Trees. The main benefit of combining these two classifiers is that they can compensate for each other's weaknesses. Decision trees are known to classify quickly and efficiently on large training sets however are not as effective against previously unseen threats with irrelevant traits (Zuhair et al., 2020). HML trains the fetching batch of trait vectors by cutting the decision edges off the Decision Tree with Naive Bayes pruning margins in an iterative splitting of the training trait vectors into sub-training vectors [55]. The training feature matrix ($T = T_1, ..., T_K$) such that ($T_i = (T_{i,j})_{i \in K, j \in |T_i|}$) with the predictive labels ($P_{class} = C_1, C_2 : C_1 = 1$ and $C_2 = -1$). Each feature vector is represented as ($T_i = C_m, (T_{i,j})_{i \in |T_K| m \in |C_M|}$). The prior class probability $P(C_m)$ is computed in eq.2.28. This predicts how often a class occurs over (T) concerning its trait vector (T_i). The conditional probability is shown in eq.2.28. eq.2.29 represents the relevance between the predicted class (C_m) and its corresponding trait ($T_{i,j}$) as indicated by $P(T_{i,j}|C_m)$.

$$P(T_i|C_m) = P(C_m) \underset{e=1->p}{\Pi} (T_{i,j}|C_m) \tag{2.28}$$

$$C_m = C_i - > P_{ms}(T_i, C_m) \tag{2.29}$$

2.3.7.3 Experiments

The system initially trains itself by building the classifier and the relationships between attributes and their relation to specific families of ransomware. The system will also identify samples which descend from multiple families and what relationships constitute this. The experiments are carried out in three phases; phase one compares HML's accuracy and detection rates to other popular machine learning algorithms. The second phase tests HML against signature-based ransomware detection solutions. The third phase of experiments tests HML against machine learning-based solutions like EldeRan and RANDS. HML consistently outperformed each of the comparative machine learning algorithms and outperformed the machine learning solutions compared to RANDS and EldeRan. The accuracy of the HML system was tested in a realistic environment over 30 days and maintained an accuracy rate of above 97.4% and a mistake rate between 2.2 and 2.6%.

2.3.7.4 Pros & Cons

This research goes far to ensure the system produced is resilient to varied ransomware. The system uses a hybridised version of Decision Trees and Naive Bayes to compensate for each algorithm's shortcomings. The extent of testing carried out in this research is impressive, with the system tested out in a realistic scenario for 30 days to monitor accuracy and mistake rates over 30 days. The other unique aspect of this study is the system's ability to identify which ransomware the samples are related to. Being able to identify the ancestry of the sample will make zero-day samples easier to identify. Unless a sample is truly unique and a new family which does not derive from any previous ransomware families, it is highly likely to be detected by the HML system. A problem may arise if the zero-day samples it encounters are not descendants of any previous ransomware family. This system also attempts to classify as fast as possible, and its results are impressive, with HML classifying faster than every system it is compared to. The time it takes to classify is crucial, as ransomware can do damage if not detected quickly. In terms of negatives, this research study has a few questions about whether the system will be as effective on strains with no known ancestors. The decision to include 35,000 ransomware samples is a step towards the system being trained comprehensively; however, the decision to only include 500 benign software is hard to justify. In a realistic setting, the system would come across far more benign software than ransomware. Therefore it would be logical to train using diverse benign software and ransomware samples. There could be more focus on training the system with benign software which behaves similarly to ransomware.

2.4 Malware Evolution Research

The following key component required research is concept drift, a variant of data shift defined by Moreno-Torres et al. Moreno-Torres et al. describe the types of data shift as covariate shift, concept drift and prior probability shift.

· **Covariate Shift** Covariate shift is defined as $P_{tr}(y|x) = P_{tst}(y|x)$ and $P_{tr}(x) \neq P_{tst}(x)$. The inequality in $P_{tr}(x)$ and $P_{tst}(x)$ is explained by the test data distribution being composed of the union of two Gaussian distributions where the centre of the test distributions has shifted away from the centre of the training Gaussian distribution. $P_{tst}(y|x_0)$ remains the same; however, the training data distribution has changed.

· **Concept Drift/Shift** Concept Drift defined by Moreno-Torres et al. $P_{tst}(x_0)$ remains constant, but $P_{tst}(x_0|y)$ has changed.

· **Prior Probability Shift** Prior Probability Shift occurs where $P_{tr}(x|y) = P_{tst}(x|y)$ and $P_{tr}(y) \neq P_{tst}(y)$; this implies the probabilities for class variables have shifted from training to testing set.

This research focuses on concept drift known to present itself in malware detection systems due to the evolution of malware behaviour. Concept drift is defined by four different types, as explained by (Gao et al., 2007).

· **No Change** In this scenario, both $P(x)$ and $P(y|x)$ remain the same and have not changed regarding how they were when the model was initially trained. It might be helpful to retrain on the most up-to-date data regardless; however, this indicates no concept drift, and the model is stable.

· **Feature Change** In this scenario $P(x)$ has changed, but $P(y|x)$ has remained the same; this means features that may not have been important prior have become essential. The model can be reconstructed in a scenario to fit these feature changes.

· **Conditional Change** In this scenario $P(x)$ has not changed, but $P(y|x)$ has changed; in this scenario expected error could go either way, so it is necessary to reconstruct the model. The issue with this is that there may not be enough new data to reconstruct the model, leading to high variance. It is possible to solve this problem through the weighted combination of new and old data.

· **Dual Change** Both $P(x)$ and $P(y|x)$ change. The expected error could increase, decrease or stay the same depending on the combination of $P(x)$ and $P(y|x)$. It is necessary to retrain in this instance.

2.4.1 Transcend

The transcend system acknowledges the rapid evolution of malware and identifies concept drift as a significant issue in malware classification (Jordaney et al., 2017). Transcend attempts to detect drift in a system will help identify when a model is degrading. Transcend uses nonconformity and p-values to judge if a sample represents concept drift. When a sample is identified as not conforming to the class it is classified into; it will need to be manually inspected by whoever is in charge of managing the system.

2.4.1.1 Feature Mapping

The approach taken in this study is to use the features extracted from the Drebin dataset and Marvin dataset; this includes a combination of permissions, intents, API calls, strings and IP addresses. This feature set is extensive and covers a wide spread of executable behavioural traits. It must be noted that these datasets were developed for the detection of android-based malware.

2.4.1.2 Non-conformity Values

A nonconformity measure measures how different an object z is to set C. This nonconformity measure can be universally used for many machine learning algorithms which use a similarity function to distinguish objects (Shafer & Vovk, 2008). eq.2.30 shows the computation of a p-value. The nonconformity measure is built with a real-valued function $A_D(C/z, z)$, which measures how different an object z is from a set C. The set C is a subset of the data space of object D.. For a set of objects K, the p-value $p^c_{z^*}$ for an object z^* is the proportion of objects in class K that are at least dissimilar to objects in C as z^* (Jordaney et al., 2017). The calculations for the non-conformity values for test objects and the set of objects in K is shown in equation 2.3 and 2.31, respectively. The computation of the p-value for the test object is shown in equation 2.32.

$$a^*_z = A_D(C, z^*) \tag{2.30}$$

$$\forall i \in K.a_i = A_D(C\ z_i, z_i) \tag{2.31}$$

$$p^c_{z^*} = A_D(C\ z_i, z_i) \tag{2.32}$$

2.4.1.3 Algorithm Credibility

The first evaluation metric is an algorithm's credibility in its prediction on a test sample. The transcend system defines $algorithmcredibility$ as $A_{cred}(Z^*)$. Algorithm confidence is defined as the p-value for the test object z^* in relation to the label chosen by the algorithm. This p-value measures the fraction of objects within K that are at least as different from the set of objects C as the new object $z*$ (Jordaney et al., 2017). High credibility means the new object z^* is very similar to the other objects in the class it has been placed in and vice versa.

2.4.1.4 Algorithm Confidence

For a given prediction, algorithm confidence will show how committed the evaluated algorithm is to the choice. Confidence is defined as how different the new object z^* is from the other classes; an example of this would be how different a sample classified as benign is from the all other benign samples. Algorithm confidence is defined as 1.0 minus the maximum p-value among all p-values except the p-value chosen by the algorithm (Jordaney et al., 2017). The formal definition of algorithm confidence is shown below where $A_{conf}(z^*)$ is algorithm confidence and $A_{Cred}(z^*)$ is prediction credibility, and $P(z^*)$ being the maximum p-value for the sample.

$$A_{conf}(z^*) = 1 - max(P(z^*)/A_{Cred}(z^*)) \tag{2.33}$$

The highest possible confidence value is achieved when algorithm credibility is the highest p-value.

2.4.1.5 Experiments

The experiments carried out involve the Drebin and Marvin datasets mentioned earlier. The datasets are split, knowing prior to this that there already is concept drift in these datasets. It must be noted that the threshold for unreliable predictions and desired performance need to be set and are defined based on the scenario, i.e. how many unreliable predictions can be tolerated. The first test comprised training the algorithm on the Drebin dataset, with testing on the Marvin dataset using 4500 malign and 4500 benign files. The algorithm used had a detection rate of 36%, showing that the presence of concept drift caused significant difficulties for the detection algorithm. The retraining was done by adding the test samples that marked unreliable predictions into the training set. The testing is carried out on a new batch of 4500 benign and malign files from the Marvin dataset, which does not overlap with the first experiment. The resulting experiment shows there is an improved detection rate.

2.4.1.6 Pros & Cons

The transcend system provides a good base for solving the concept drift problem in malware detection. The transcend system effectively detects malware showing concept drift and shows that retraining with samples showing concept drift can drastically improve detection rates. Additionally, the Transcend system is proven effective in different feature spaces and data types whether the feature set is already robust to concept drift. The Transcend system. Most importantly, the Transcend system's design does not depend on severe drift to detect low-quality decisions and can be effectively combined with robust feature spaces that may be less susceptible to drift. The transcend system's main shortcoming is its reliance on retraining and human intervention when a drifting sample is detected. The Transcend system cannot deal with drifting samples without human intervention or retraining, which means the chance of misclassification can still exist due to human error.

2.4.2 Prediction with Confidence Based on a Random Forest Classifier

This system uses nonconformity values, similar to the transcend system; however, this method is designed strictly for use with a random forest or decision tree structure (Devetyarov & Nouretdinov, 2010).

2.4.2.1 Feature Mapping

This research study uses generic pre-made datasets to test their results and does no feature extraction of its own.

2.4.2.2 Nonconformity with Machine Learning Algorithms

This NCM (Nonconformity Measure) is based on random forest proximities $P(i, j), i, j = 1, ..., m + 1$, which provide a measure of how close to each other two objects are regardless of their labels and are calculated as a ratio of trees, running through which objects i and j land at the same terminal node. A random forest is constructed for the union of a bag $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ and a new example (x, y) and form the corresponding $(m + 1)(m + 1)$ matrix of proximities for objects $x_1, x_2, ..., x_m, x_{m+1} = x$. The nonconformity measure is the ratio of the average proximity of the example with examples of other classes to the average proximity of the example to examples of the same class. In both averages, we consider only proximities of those k examples with the greatest values of proximities among examples of the same class y and all the other examples. Strictly speaking, $A(x, y) = A(x, y)/A(x, y)$ where i_s and j_s the numbers of examples with $s - st$ greatest value of proximity with example (x, y) among examples labelled with the same label y and among all the other examples, respectively.

$$A(x, y) = \sum_{s=1}^{k} P(i_s, m + 1), A(x, y) = \sum_{s=1}^{k} P(j_s, m + 1) \qquad (2.34)$$

2.4.2.3 Experiments

In this study, in the experiments before using the nonconformity measure, the testing accuracy in datasets with known concept drift was very low; however, it sees significant increases in accuracy after the implementation of the decision tree-based nonconformity measure. The increase in accuracy across all the datasets shows significant improvement.

This study uses nonconformity values combined with random forests to detect samples showing concept drift. Unlike the transcend system, this system only works with decision trees and random forests; therefore, it is limited in that aspect. The results achieved by using non-conformity values in the detection phase are impressive, and the detection rate improves significantly when non-conformity values are applied. The main flaw in this system is its limitation to decision tree-type classifiers. An approach incorporating non-conformity values can easily be adapted to be used in various classifiers.

2.4.3 EACD: evolutionary adaptation to concept drifts in data streams

This study proposes a method of coping with concept drift in non-stationary systems through evolutionary algorithms. This evolutionary algorithm used to select features will use a simulated version of natural selection, likening features to chromosomes to select the optimal feature set for the current data (Ghomeshi et al., 2019).

2.4.3.1 Feature Selection

This evolutionary algorithm is multilayered with a base layer and a genetic layer; these layers act as a natural selection mechanism to find the most robust feature set. The base layer will select a set number of features and save them as feature sets. These feature sets are saved and evaluated. The highest-performing feature sets are passed into a secondary genetic layer that will then "breed" these feature sets together by randomly crossing strong feature sets to create strong offspring. This breeding step is carried out until the accuracy of the overall system is higher on the newest data than on average. This approach is essentially mimicking the genetic process of natural selection, with each feature viewed as a "chromosome", and feature sets from the initial base layer are viewed as the parental generation, with the feature sets being bred in the genetic layer being the children of the feature sets from the base layer. This process is repeated when a concept drifts until the system's accuracy is maximum for the newest block of stream data.

2.4.3.2 Experiments

Considering this research was built for stream data, it is tested on data in a stream form containing concept drift. EACD shows the best overall accuracy for nearly all datasets compared to other algorithms, in some cases significantly. Out of the ten datasets used, the average accuracy of EACD is 87.1%, higher than all other algorithms.

2.4.3.3 Pros & Cons

EACD uses a genetic algorithm to adapt classifiers to concept drift. The EACD system improves the detection rate after retraining once a drift has been detected in stream data. The EACD system presents a strong argument for including genetic algorithms in the feature selection phase of any classifier due to its flexibility when faced with concept drift; however, retraining is triggered by misclassification, and the classifier cannot adapt to drift dynamically. The EACD system is a solid base for building a concept drift classification system; however, it cannot adapt to drift.

2.4.4 A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions

This study focuses on adaptation to concept drift in skewed datasets. In many cases, the detection of threats will be a small proportion of the "normal" data in a dataset. If in a data stream, then a minority of that stream will be abnormal or malicious data. This study finds and adapts concept drift in skewed data streams. This study uses an ensemble and sampling in combination with taking bias and variance introduced by a classifier when making a prediction.

2.4.4.1 Error Decomposition

It is acknowledged that a classifier will estimate the posterior class distribution, but this is not always the true probability of the classification. The remaining errors are decomposed into bias and variance. The bias measures the difference between the expected probability, and the variance measures the changes in estimated probability using varied training sets (Ghomeshi et al., 2019). The output of a classifier is defined below in Eq.2.35.

$$f_c(x) = P(c|x) + \beta_c + \eta_c(x) \tag{2.35}$$

This makes $P(c|x)$ the posterior probability of class c, given input of x with β_c being the bias and η_c being the variance of the classifier. Combining this classification method with a sampling of previous positive samples into the current training set reduces the variance at the cost of slightly increased bias. Lastly, the introduction of an ensemble further reduces the variance.

2.4.4.2 Experiments

For the experiments, synthetic data with different concept changes are generated. The sampling ensemble used in this study is compared with two other baseline concept drift detection systems on a variety of datasets with synthetic concept drift introduced to them, and the sampling ensemble is the most accurate across all datasets. The datasets were modified to contain varying skew and concept drift levels to capture real-life scenarios accurately.

2.4.4.3 Pros & Cons

This system focuses on concept drift in skewed datasets; this is appropriate for malware detection as benign files tend to be encountered far more than malware files. It is crucial to consider skewed data when detecting concept drift, as this is the likeliest scenario in the malware detection space. The main flaw in this system is that it increases the bias of a classifier; however, its positive results when detecting concept drift. The system is also tested on various concept changes through generated synthetic data.

2.4.5 Dynamic integration of classifiers for handling concept drift

Tsymbal et al. acknowledge that concepts in machine learning classifiers change with time and do not remain stable for a long time. This research attempts to use ensemble integration techniques to help classify samples under concept drift at the instance level. Each base classifier is given a weight proportional to its accuracy in the tested instance, and the best base classifier is selected (Tsymbal et al., 2008). The experiments in this research use synthetic data sets that simulate abrupt and gradual concept drifts with real-world antibiotic resistance data.

2.4.5.1 HEOM (Heterogeneous Euclidean Overlap Metric) Calculations

In HEOM, the Euclidean distance is used with numeric features and the overlap distance with categorical features to find a distance between two instances x_1 and x_2 as shown below, where m is the number of features. The HEOM metric can plot instances in n dimensional space and measure the normalised euclidean distance of an instance from other instances; this allows the comparison of proximity between benign and malign samples in n dimensional space. For a numeric feature a the distance is normalised by the width of the range of values of the corresponding feature on the training set $range_a$.

$$d_{heom}(x_1, x_2) = \sqrt{\sum_{a=1}^{m} heom_a^2(x_1, x_2)} \tag{2.36}$$

Where

$$heom_a(x_1, x_2) = \frac{|x_{1a} - x_{2a}|}{range_a} \tag{2.37}$$

2.4.5.2 Experiments

The datasets used in this research represent the SEA concepts problem and the rotating hyperplane problem. The concepts in the data are based on antibiotic resistance in nosocomial infections; however, it is stated that the concepts in this domain are stable despite changing over time; therefore, it may not represent real-world concept drift that may be abrupt. The algorithm used is the ensemble learner in the WEKA API. The system achieves an accuracy of 85% with a Naïve Bayes classifier ensemble with the drift detection handled by a HEOM measurement.

2.4.5.3 Pros & Cons

This study focuses on detection with concept drift; it uses a euclidean distance metric to detect drift, yielding positive results. The datasets used are not relevant to malware; however, it represents how HEOM can adapt classifiers to concept drift. The main flaw in this research is that the base classifiers are weak, as the most robust detection rate achieved in this study was 85%. The positive results concerning concept drift indicate that this approach could be transferred to other fields.

2.4.6 Machine Learning Detection for Malicious URLs

This research attempts to classify malicious URLs while taking concept drift into account. The URL detection is based on lexical, host-based and content-based features. The algorithms used to detect URLs are the Gradient Boosted Trees, Random Forests and Deep Neural Networks while using the Heterogeneous Euclidean Overlap Metric (HEOM) to detect concept drift. Singhal et al. state that the loophole in this research is the availability of training data to attackers, which could allow attackers to adapt their methods to trick the classifiers. Singhal et al. propose that attackers introduce false concept drift using their knowledge of URL data to obfuscate malicious URLs. The research proposed by Singhal et al. proposes using HEOM to detect artificially induced concept drift and retrain classifiers to adapt to it.

2.4.6.1 Feature Mapping

The feature set is split into lexical, host-based, and content-based. The lexical features are extracted from the URL (Singhal et al., 2020), including URL length, host length, host token count, path length and signal number. The host-based features include location and autonomous system number. The content-based features include the URL contents and the functions it invokes. The feature extractor works on raw data to extract these features.

2.4.6.2 Algorithms

The algorithms are the Gradient Boosted Trees, Feed-Forward Neural Networks and Random Forests. The most accurate algorithm is the Random Forests; however, the algorithm's configuration has not been elaborated on.

2.4.6.3 Concept Drift Detection

The concept drift detection is required because of the artificial concept drift introduced by attackers to evade detection by machine learning classifiers. The concept drift detection deployed by Singhal et al. detects concept drift in real-time. The drift detection module focuses on detecting the difference in data distribution between the old training data and the newly collected feature vector data (Singhal et al., 2020). For every malicious URL in the dataset, the closest malicious URL in the old training dataset is observed, and if the average distance between new and old malicious data is above a specified threshold, it is decided that concept drift has occurred. Concept drift is also said to have occurred if a steep drop-off in accuracy is noted in the validation set. Once concept drift is detected, the supervised learner is retrained with the new data to adapt to the new concept. The measurements for HEOM are shown below.

$$d_{heom}(x_1, x_2) = \sqrt{\sum_{a=1}^{m} heom_a^2(x_1, x_2)} \tag{2.38}$$

Where

$$heom_a(x_1, x_2) = \frac{|x_{1a} - x_{2a}|}{range_a} \tag{2.39}$$

2.4.6.4 Pros & Cons

The results obtained in this study are promising, and the HEOM metric appears to be accurate when detecting concept drift; however, the system requires retraining to adapt to concept drift which means that the system will inevitably misclassify drifting samples before the system is retrained to adapt to the new concept. This research provides one piece of the puzzle, but its adaption technique is flawed.

2.4.7 Information Gain Concept Drift

The research proposed by Hussain et al. attempts to detect concept drift using an instance's Euclidean distance and mutual information gain. Concept drift detection is done in stages, and the mutual information and Euclidean distance measurements are compared against thresholds for these values representing drift. The dataset used for this research is the SEA dataset; however, it is unclear what features are used for the classification.

2.4.7.1 Euclidean Distance

Euclidean distance was applied with the k-means clustering algorithm to measure the distance from a centroid. The distance from the centroid determines similarity; therefore, being beyond a threshold distance would imply concept drift. The centroids of the clusters help determine the distance of the sample from where it is expected to be. The calculations for Euclidean distance are shown below where $p_x and p_y$ are the x and y values for sample p and the corresponding q values .

$$d(p,q) = \sqrt{(p_x – q_x)^2 + (p_y - q_y)^2} \tag{2.40}$$

2.4.7.2 Mutual Information

The mutual information measures the mutual dependence between two variables, and in this research, it is used to determine the mutual dependency between samples in the training and test sets. The calculation for mutual information is shown below.

$$M(X,Y) = H(X) + H(Y) – H(X,Y) \tag{2.41}$$

$H(X)$, $H(Y)$ are the marginal entropies of the test and train dataset, and $H(X,Y)$ is the joint entropy of both the train and test dataset. The normalised mutual information is a normalised value between 0 and 1 (Hussein et al., 2021). The mutual information measures the correlation between the test and training data; therefore, a value below a certain threshold is defined as concept drift.

2.4.7.3 Pros & Cons

The results achieved in this research are promising and show that the mutual information measure and Euclidean distance can be used to detect concept drift. The test data suggests accuracy lower than 90%; however, the boosted and bagged trees achieve high accuracy rates when combined with the mutual information measure. The Euclidean distance fails to achieve accuracy rates above 90% but can still outperform the classifiers that did not use the Euclidean Distance. It must be noted that the adaption to drift involves retraining and cannot classify samples taking individual drift into account; this could lead to misclassification until drift is detected and the classifier is retrained for the new concept.

2.4.8 RDDM

Reactive Drift Detection Method (RDDM) is based on DDM, which is known to be efficient and straightforward when detecting drift when concepts have lasted for a long time. RDDM is designed to be reactive and discard older instances of long concepts and, therefore, be able to react faster to drifts and increase overall accuracy. The RDDM method is designed similarly to DDM as a drift detection method that can operate in networks or financial transaction monitoring. RDDM is tested against DDM, EDDM, ECDD and STEPD and achieves more accurate results than alternate drift detection methods.

2.4.8.1 RDDM Drift Detection

DDM works in real-time to detect concept drift by measuring the error rate in a classifier and judging how well it performs in real time. Barros et al. acknowledge that it is necessary to periodically reduce the number of instances in a long stable concept to reduce the performance loss suffered by DDM. DDM suffers in performance because of the number of instances that would be considered when defining a concept that could number in the many thousands. Thousands of instances that make up a concept would mean many misclassifications are needed to affect the mean error rate and detect drift. In systems that detect network threats or banking fraud, latency issues can mean costly errors. RDDM aims to keep track of gradual and abrupt shifts in concept by limiting the maximum number of instances in the classifier.

2.4.8.2 Pros & Cons

RDDM consistently outperforms the drift detection methods it is compared to across multiple datasets and can achieve a higher accuracy across many datasets that show concept drift. The main issue with the DDM-based drift detection methods is that it involves monitoring the error rate; the Early Drift Detection Method (EDDM) manages the distances between errors and the error rate in a classifier. The problem with relying on misclassifications to detect drift is that this approach cannot be viable in systems that cannot afford an increase in misclassifications before retraining. In critical malware detection systems, an approach like DDM, EDDM or RDDM would lead to multiple malware infections before concept drift is detected and addressed. DDM-based methods also rely on slowly retraining to new concepts and have not presented a solution that would strengthen a classifier to be resistant to concept drift and not have any alternative to it but to retrain.

2.4.9 Android Malware Concept Drift

Guerra-Manzanares et al. observe that current malware detection research fails to consider the rapid evolution of malware. Machine learning detection solutions have shown promise and an ability to detect malware; however, they tend to degrade in scenarios that involve concept drift. Concept drift in malware detection is represented by the evolution of malware behaviour and attackers changing malware behaviour to evade machine learning solutions. This research addresses concept drift by integrating an app's internal timestamp into prediction.

2.4.9.1 Drift Handling

The proposed adaption method to drift uses a pool of classifiers like an ensemble; these classifiers are trained on past data to classify new incoming data (Guerra-Manzanares et al., 2022). The real-time prediction process chooses the best classifiers dynamically to perform accurate predictions. The classifier pool is modified to add classifiers trained on new data and remove low-performance classifiers. The main goal is to keep performance optimal by updating the classier pool with models trained on new data. The use of an ensemble allows for the constant refreshing of classifiers and the chance that some classifiers may be better adapted to drift than others.

2.4.9.2 Concept Drift Categorisation

This research aims to handle concept drift and to analyse the changes and differences in android malware. The permutation feature importance technique is used to analyse the essential features from data during different periods; the android malware was collected between 2008 and 2020. The permutation feature importance technique determines the importance of a feature for the model by assessing the decrease in the model's performance after a random permutation for the specific feature is performed while keeping the other features unchanged (Guerra-Manzanares et al., 2022).

2.4.9.3 Experiments

The dataset used for the experiments in this research is the KronoDroid dataset which contains 28,343 malware samples and 34,981 benign samples. The dataset contains applications from 2008 to 2020, which is the majority of the time android has been available. The overall accuracy of the system using a Random Forest classifier is 95%. The system does not just focus on the classification under drift but on identifying the features that are key to classification in android. The method of updating classifiers in the pool helps keep the list of essential features up to date.

2.4.9.4 Pros & Cons

Overall this approach is one of the most extensive available for malware detection under concept drift. The individual modules of drift detection, handling and categorisation attempt to form a comprehensive solution to concept drift in malware. Analysing classification data to update the essential features is particularly useful and will help improve malware classification's overall reliability. There is not enough emphasis on the drift detection method, as it relies on retraining and constantly updating the classifier pool. Maintaining an extensive array of classifiers can be computationally expensive, and having to retrain constantly opens the possibility of needing misclassification before concept drift is detected.

2.5 Discussion On Ransomware Detection & Malware Detection with Concept Drift

Having reviewed various studies on malware detection, ransomware detection and concept drift detection and adaption, it is clear that ransomware detection research lacks concept drift integration in the classification process. Malware and ransomware detection follows the template of training and test data; in some cases, classifiers will be tested on zero-day ransomware by exposing the classifier to ransomware on which it has not been trained. Exposing a classifier to zero-day ransomware or malware will help test its ability to stand the test of time; however, concept drift would mean that the behavioural patterns of the ransomware have changed. If ransomware samples evolve to behave differently from what a classifier expects, the classifier will experience a loss in performance and misclassify at a higher rate. Zero-day samples do not necessarily mean evolved samples, as new variants of already existing ransomware or malware can always appear with minor tweaks but still fit into a pattern that a machine-learning system can identify correctly. Evolved zero-day malware or ransomware are new variants of malware or ransomware or completely new families that do not fit the pattern of behaviour or composition for what a classifier would define as malware or ransomware.

Current concept drift malware studies like Transcend (Jordaney et al., 2017) attempt to identify concept drift in malware detection systems; however, the approach taken is to detect concept drift as it occurs and to retrain once it has reached an unacceptable level. Detection of concept drift is a crucial part of solving concept drift-related detection issues in ransomware; however, it is only part of a complex issue. The approach taken by the studies reviewed detects concept drift and relies on retraining to maintain the performance of a classifier. The approach for retraining has been handled differently from study to study, with some approaches using thresholds for accuracy to determine to retrain points and some studies using proprietary metrics to determine prediction quality and whether the level of concept drift is reasonable or unacceptable. Some approaches involve ensemble learning that discards weak-performing classifiers and retains robust classifiers while constantly retraining to maintain high accuracy levels. The lack of drift incorporation in classification means a classifier cannot deal with a drifting sample, only that the system knows a sample is drifting and may require retraining in the future.

The concept drift detection methods that have been reviewed use various methods to detect concept drift and are fit for purpose. Some approaches use sliding windows, and others use distance metrics and prediction probability to determine the fitness of a prediction. There are many effective methods to detect concept drift; however, these methods require an additional layer to make them useful for the prediction phase of a classifier.

2.5.1 Open Research Directions

After reviewing the current research space for ransomware detection and malware detection with concept drift, open research directions can be considered. None of the systems uses concept drift metrics in predictions; incorporating concept drift into predictions will allow a classifier to reliably make predictions for drifting samples without relying on immediate retraining. Additionally, using concept drift in a prediction system will mean that the retraining points can be more accurately identified, as the system will know what drift levels are acceptable in the real world. Focusing on data is also essential; accurately capturing evolving malware species and how they evolve is important when prepping a system for real-world deployment. Creating datasets that can accurately represent real-world ransomware or malware evolution is essential if a system is prepared to anticipate future concept drift or evolution. There is also a question of what metrics and distance measures are best suited for detecting and quantifying concept drift. In the concept drift detection space, different similarity metrics are used for detecting drift; however, it is important to test drift detection frameworks and approaches with different similarity and distance metrics to ensure the optimal solutions are found.

2.6 Feature Selection

Feature Selection plays a big part in building a machine learning classifier; the main reason is that datasets can have large amounts of features, and users cannot manually identify the best features. High dimensionality can increase the complexity of a machine learning algorithm and use features that provide no valuable information in the classification process. Feature selection algorithms can significantly reduce dimensionality and produce optimal feature sets for a classifier to classify samples. The research reviewed in this section uses a large variety of feature selection algorithms; different feature selection algorithms have different positives and negatives. The most common feature selection is the mutual information criterion used by significant research proposed by (Chang et al., 2018) (Abbassi et al., 2022) (Daku et al., 2018) (Hasan & Rahman, 2017) and (Sgandurra et al., 2016). The idea of using information gain focuses on isolating features that provide the most information related to the class of a sample. The information gain algorithm decides which features are most important by splitting data on one feature and calculating how well the data is split based on a particular feature.

Nature-inspired algorithms prove effective in feature selection; (Khan et al., 2020) use MOGWO, a feature selection algorithm based on the hunting patterns of grey wolves. The evolutionary algorithm is also proven effective in creating feature sets to combat the onset of concept drift in machine learning systems (Ghomeshi et al., 2019). The adaptability of nature-inspired algorithms and their proven ability based on their real-world counterparts make them a healthy option when deciding on a feature selection algorithm. Nature-inspired algorithms being used to adapt to concept drift makes them particularly useful as concept drift is a common problem in machine learning classifiers and a severe issue to malware detection solutions.

Some feature selection algorithms use text analysis to identify the most prominent features. The Occurrence of Words (OoW) and Sequence of Words (SoW) techniques are used to identify frequently occurring words and frequently occurring sequences of words (Molina et al., 2022). Molina et al. use the SoW technique to identify "paranoia" activities in ransomware samples that give away the early stages of a ransomware infection. Techniques which rely on specific patterns and occurrences of words could help identify features for ransomware detection; however, it is essential to ensure the use of sequences does not leave room for an attacker to use sequences of actions to obfuscate the malware's true intentions. Text analysis is a lesser-used technique than the alternatives presented in this section.

Some researchers manually identify features for ransomware detection; the main shortcoming of this approach is that human error can lead to missing key features. In addition to human error, it is not straightforward to manually identify statistical relationships between features, whereas an algorithm can do so.

2.7 Performance Metric

Table 2.1 Performance Metrics

Metric	Calculation	Value
TPR (True Positive Rate) / Recall	$\frac{TP}{(TP+FN)}$	Correct classification of Ransomware.
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$	Benign software classed as Ransomware.
False Negative Rate (FNR)	1-TPR	Ransomware classed as benign.
Precision	$\frac{TP}{TP+FP}$	Proportion of ransomware classifications, that are actually ransomware.

Table 2.1 shows the performance metrics we use to evaluate the performance of the machine learning algorithms and the FeSAD framework, which uses machine learning algorithms.

2.8 Concluding Remarks

This chapter has reviewed the operation of ransomware, the current state of ransomware and malware detection studies related to concept drift. We observe that different features are used for the approaches we have looked at, such as API calls, file activities, low-level input and output activity, and network activity. The most common approach used by multiple researchers is API calls, and this is because API calls can give insight into the behaviour of the ransomware. API calls can be used to infer file activities and some network activity, making it no surprise that it is widely used. API calls have been proven helpful for early detection with multiple studies such as Sgandurra et al., Molina et al. and Urooj et al.; these studies achieve strong detection rates and being able to detect before encryption is a crucial aspect of ransomware detection. The best-performing results are not simple to distinguish as the approaches use different methods of detection and have different approaches to testing, with some studies using zero-day simulations and one using population drift as well, whereas some do not account for zero-day threats. A large portion of studies that achieved detection rates above 96% used a tree-based algorithm or the Bayesian network. Neural network and deep learning approaches are also capable of solid results, which are only three to five per cent less effective than the tree-based and Bayesian approaches, so further evaluation would be needed to determine what algorithms work with the chosen feature set.

Chapter 3

FeSAD Framework

This chapter discusses FeSAD, Feature Selection and Ransomware Detection Framework with Machine learning using Adaption to Concept Drift. The FeSAD framework aims to combat performance degradation in machine-learning ransomware detection systems caused by concept drift. The FeSAD framework uses statistical and probability metrics to make a machine learning system resilient to ransomware evolution and lengthen times between retraining. The FeSAD framework consists of the Feature Selection Layer (FeSA), the Drift Calibration Layer and the Drift Decision Layer. This chapter will first focus on the FeSA feature selection layer and its operation. The Drift Calibration Layer's operation will be explained as its link to the Feature Selection Layer. Finally, this chapter will explore and elaborate on the Drift Decision Layer; the link the Decision Layer has with the other two components will be explained in detail.

3.0.1 Novelty

The FeSAD framework proposes a method that considers concept drift at every stage. A machine learning classifier can be prone to concept drift for numerous reasons; the reliance on anti-virus software on machine learning classifiers makes the presence of concept drift in ransomware dangerous. The FeSAD framework provides three layers of protection for a ransomware detection system.

- Firstly FeSAD uses a feature selection algorithm that is proven effective when adapting to concept drift.
- Existing research in concept drift adaption relies on retraining based on misclassification; this is an approach that cannot be considered for ransomware detection.
- The FeSAD framework considers concept drift when training a machine learning classifier during its test phase, allowing machine learning classifiers to be more long-lasting and robust when facing concept drift.

3.0.2 Security Requirements for a Ransomware Detection System

- **FeSA feature selection Layer:** The FeSA Layer in FeSAD needs to act as a practical feature selection algorithm that improves the accuracy of a ransomware detection system while reducing dimensionality. The FeSA layer must provide a feature set that increases detection rates while maintaining robust performance under concept drift.
- **Accurate Detection of Ransomware:** The FeSAD framework must accurately detect ransomware that it has encountered and it has not encountered. The FeSAD detection framework must be effective in detecting zero-day ransomware. The classifier used by the FeSAD framework must detect ransomware outside the distribution of ransomware the classifier is trained on. The detection mechanism of FeSAD must be adaptable to concept drift scenarios. Concept drift scenarios imply the ransomware is a zero-day strain and displays characteristics that the classifier does not expect; the concept drift would mean the ransomware has evolved.
- **Ability to Adapt to Concept Drift without Retraining:** The FeSAD framework needs to identify concept drift and classify samples showing concept drift reliably without retraining. Existing systems require drop-offs in accuracy and misclassifications to identify concept drift. Once concept drift is identified, existing systems require retraining immediately on new samples. The FeSAD framework needs to identify concept drift and classify samples accordingly to prevent accuracy reductions and catastrophic misclassifications.
- **Ability to Identify Benign Samples:** The FeSAD framework must accurately distinguish between ransomware and benign files by creating a baseline behaviour for ransomware and benign files that will avoid high numbers of false positives.
- **Ability to Adapt to Concept Drift without Retraining:** The FeSAD framework needs to identify concept drift and classify samples showing concept drift reliably without retraining. Existing systems require drop-offs in accuracy and misclassifications to identify concept drift. Once concept drift is identified, existing systems require retraining immediately on new samples. The FeSAD framework needs to identify concept drift and classify samples accordingly to prevent accuracy reductions and catastrophic misclassifications.
- **Alert Users when Retraining is Required:** The FeSAD framework is robust when classifying ransomware samples that show concept drift; however, it is essential to know when the underlying classifier requires retraining. Concept drift would imply that the underlying classifier is less confident of predictions and needs to rely on the concept data presented by FeSAD to make an accurate classification. Once the classifier's prediction confidence drops below a threshold, it is clear the classifier can no longer classify the incoming data; therefore, it needs retraining as a last resort.

3.1 Machine Learning-based Ransomware Detection

Machine Learning detection for malware and ransomware, in particular, has been proven effective and is widely applied in real-world scenarios (Sgandurra et al., 2016). Machine Learning algorithms can learn a concept and classify samples based on the rules and patterns determined during a training phase. Machine Learning approaches to ransomware detection are crucial because the reliance on outdated heuristic approaches to malware detection is unrealistic and ineffective. The rapid growth of the internet and the constant evolution of malware makes it impossible to keep track of every new malware variant. The accelerated rate of new malware in the modern era emphasises the need for a detection system to keep up with changing trends and evolving threats. Machine Learning detection systems can provide the following advantages for ransomware detection.

- Ability to learn ransomware behavioural and static concepts that can help detect ransomware without the need for heuristic features.
- The ability to use a variety of approaches and techniques to adapt to a problem space.
- Ability to detect zero-day ransomware that has not been seen before.
- The ability to retrain and adapt to new ransomware concepts.
- The ability to work with advanced feature selection methods to increase the efficiency and accuracy of the machine learning algorithm.

Machine Learning algorithms have many advantages and positive applications, especially in the ransomware detection space; however, there are also challenges faced by machine learning approaches that need to be addressed when applying them to a problem.

- Overfitting is an issue in machine learning in which the algorithm learns a concept that is heavily adapted to the training data that is not necessarily as prevalent in the test data or the real world.
- The data used to train a machine learning algorithm needs to be adequately prepared and varied to prepare a system to work in a real-world scenario; poor quality data leads to a poor quality system. The features used in the data needs to effectively detect the malware.
- Concept drift is a problem that can affect machine learning algorithms in many real-world fields such as malware (Jordaney et al., 2017), and this applies to a greater extent in ransomware detection. Concept drift in a machine learning scope means the concept learned by the classification algorithm has now shifted and needs to be updated or adapted. Ransomware is a quickly evolving malware type that can cause much damage through even one misclassification; therefore, concept drift must be considered when using a machine learning classifier to detect malware as dangerous as ransomware.
- Machine learning detection systems require monitoring and maintenance to ensure they are up to date and running as intended. Detection systems that detect changes in malware patterns usually require immediate retraining; this is seen in work presented by Singhal et al. and Jordaney et al.
- Some Machine learning solutions that address concept drift require ensemble learning systems with multiple classifiers, relying on switching between classifiers that are stronger on a current concept (Ghomeshi et al., 2019).

3.1.1 Ransomware Attacks

Despite the knowledge of ransomware in recent years and the added integration of more sophisticated cyber defence systems, ransomware attacks have remained a significant threat to individuals and businesses. Ransomware attackers are constantly developing and adapting threats to new vulnerabilities, and this often causes damage to businesses financially, legally, and their reputation. With ransomware attack vectors changing to adapt to new flaws and exploit methods, it is almost impossible to prevent a ransomware attack with 100% certainty. Some of the major attacks that have occurred recently are described in this chapter.

· **Revil 2022** Revil ransomware was used to attack Australia's Medibank health insurance group. The Medibank hack was damaging in terms of encrypted data, and the attackers managed to access the confidential data of 9.7 million customers. The data stolen from Medibank was subsequently put on the dark web, including names, birth dates, passport numbers and health insurance claims. The data stolen was also used in follow-up attacks which involved identity fraud. Revil, also known as Sodinokibi, is a Russian RaaS operation used in multiple prominent ransomware attacks (Kerner., 2022).

· **LockBit 2022** Lockbit ransomware was used to attack the Toyota motor company and their partners, Kojima Industries, Denso and Bridgestone. As a result of these attacks, Toyota had to shut down plants in Japan and Central and North America. Denso identified the Pandora group as the initiators of the attack(Kerner., 2022).

· **Maze 2020** Maze ransomware attacked IT services giant Cognizant in 2020 and spread laterally through their network while stealing unencrypted data from their servers. Cognizant was warned that stolen data would be leaked if ransoms were not paid. Maze is usually spread through phishing emails that contain zipped folders, which are difficult to scan for malicious code. Once Maze is in a network, it will scan for vulnerabilities and attempt to move laterally through the network once it has exploited a vulnerability (Freed., 2021).

· **PureLocker 2020** PureLocker encrypts servers similar to Ryuk and is written in PureBasic. PureLocker is written in PureBasic, a language used on various platforms, and it is difficult to detect malicious software. PureLocker uses evasion techniques to appear normal in sandbox software by performing regular tasks in a sandbox (Freed., 2021).

· **Ryuk 2019** Ryuk targeted tribune newspapers and a water company in North Carolina, with affected newspapers having to run downscaled versions of their daily news without classified ads. Ryuk used RDP software and Trickbot to encrypt servers, endpoints, and backups and disabled windows restore points to force a payment between 15 to 50 bitcoin. Ryuk aimed to spread through servers and was discovered to originate from a North Korean hacking group; the language of the attacked machines needed to be changed to Russian, Belarusian or Ukrainian (Freed., 2021).

· **REvil 2019** REvil is ransomware originating from Russia; this is deduced as REvil will not execute in Russia or surrounding areas. REvil is prominent due to its data exfiltration approach; it not only encrypt data but also threatens to expose data taken from the victims if ransoms are not paid. REvils most notable target is British foreign exchange firm Travelex which was hit with a ransom demand of $6million. The attack would cripple Travelex, send the company into administration, and eventually insolvency. REvil exploits vulnerabilities in Oracle servers and Pulse Connect Secure VPN (Freed., 2021).

- **WannaCry 2017** WannaCry was the start of global ransomware operations; WannaCry was the first major ransomware attack that grabbed public attention with the malware spreading globally. The WannaCry malware exploited NSA cyber weapons, doublepulsar and eternalblue, to attack Microsoft SMB servers. WannaCry notably infected the UK's NHS and rendered thousands of patient records unreachable. The WannaCry outbreak infected hundreds of thousands of computing devices are was eventually halted when a malware researcher activated an internal kill switch (De Groot., 2017).

3.1.2 Critical Challenges for a Ransomware Detection System

- **Detecting Zero-Day Threats:** It is relatively simple for an antivirus system to detect known threats as there are threat hunting teams constantly finding new threats and ensuring AV databases are updated with their signatures; this process also applies from a ransomware detection standpoint. Ransomware detection is most needed when detecting zero-day threats. Zero-day ransomware threats could contain various elements that make it a zero-day such as having new infection vectors or new behavioural patterns that could make it difficult for detection systems to identify.

- **Detection Before Irreversible Damage:** A ransomware attack could happen at any point and occur for various reasons, and in some cases, it may infect an end-point on a network that then allows the ransomware to propagate. It is vital that if an infection has occurred, it is stopped before it spreads through a network or does irreversible damage to an end-point.

- **Adaption to Concept Drift:** Concept drift is an issue that affects machine learning algorithms in general, and it occurs when there is a change to the components of a prediction. The components of the prediction could be features and how they relate to a prediction or behavioural changes in the subject of the predictions. Concept drift in ransomware is prevalent due to the rapid evolution of the malware type. Ransomware evolution is fast-paced, and a single misclassification by a detection system can be catastrophic to individuals and businesses.

- **Adversarial Learning:** Attackers may be able to mimic systems used to detect ransomware and generate samples that can evade these detection systems. The sensitivity of a ransomware detection system to change is crucial as it must be trained with enough variation to understand the whole ransomware space. Overfitting to training data must be avoided misclassifications.

- **Attack Vectors:** Ransomware has many infection vectors; therefore, detecting and preventing attacks from all infection vectors is not always possible. New attack vectors are the cornerstone of ransomware attacks because of their ability to bypass security to load ransomware onto systems for them to infect end-points and servers. It should be assumed that no network is safe from ransomware attacks and that the possibility of ransomware infection exists due to security vulnerabilities or human error factors. The variety of attack vectors and the possibility of exploiting human error make post-infection detection a crucial aspect of ransomware detection and protecting the broader network beyond an infected end-point.

3.2 FeSAD Ransomware Detection Framework with Machine learning using Adaption to Concept Drift

The FeSAD framework in Figure 3.1 aims to help ransomware machine-learning detection systems make accurate predictions under concept drift. FeSAD is also designed to prolong a detection system's period without retraining by considering concept drift. The FeSAD framework consists of three central components: the FeSA layer, the Drift Calibration Layer, and the Drift Decision Layer. The FeSA layer consists of a genetic feature selection algorithm that creates a strong feature set that should provide a layer of resistance to concept drift. The genetic algorithm was chosen as the feature selection algorithm because current research shows that genetic algorithms provide strong adaptability to concept drift. Additionally, experiments carried out in this research have shown that using the genetic algorithm as a feature selection algorithm reduces the performance degradation introduced by concept drift. The FeSA layer calculates the components required to measure drift and predict while the system is experiencing drift. The FeSA layer calculates the initial "drift" in the training data to identify the boundaries for variation in the ransomware and benign files in the training data. The FeSA layer uses the drift calibration layer to update its drift thresholds; this process is carried out to measure the change in drift between different ransomware evolutions; this keeps prediction accuracy high and remains robust over time. The Drift Decision Layer of FeSAD can identify what samples show concept drift and when it is time to retrain. The approach taken by the FeSAD framework aims to prolong periods between retraining while reducing the effect of concept drift on a machine learning detection system. There is a set threshold for the proportion of samples displaying abnormal or unacceptable drift the system can encounter before retraining must be carried out. The retraining threshold exists to identify when the underlying classifier no longer has a grasp on the current concept being shown in the data.

3.3 FeSAD Framework Design

3.3.1 Proposed System

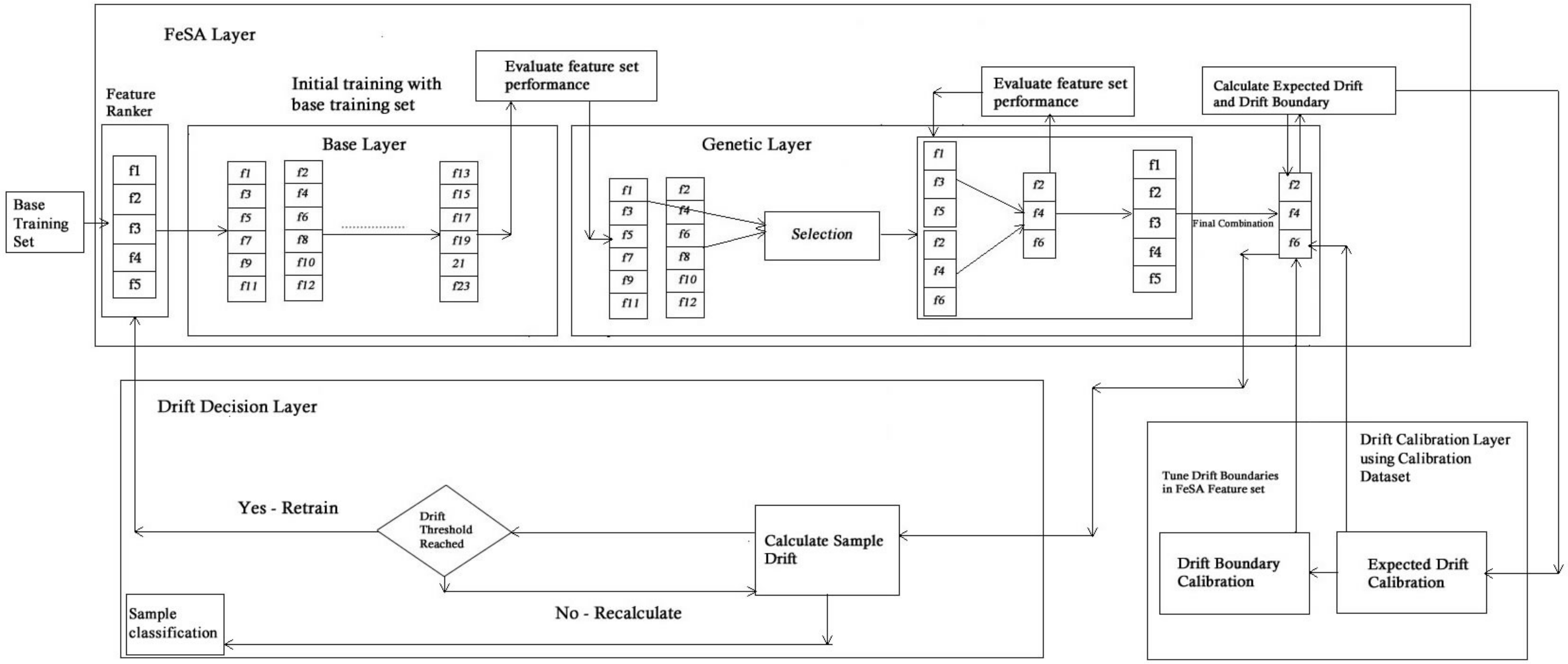
FeSA Layer
Feature Ranker
Initial training with base training set
Evaluate feature set performance
Base Layer
Genetic Layer
Evaluate feature set performance
Calculate Expected Drift and Drift Boundary
Base Training Set
f1 f2 f3 f4 f5
f1 f3 f5 f7 f9 f11
f2 f4 f6 f8 f10 f12
f13 f15 f17 f19 21 f23
Selection
Final Combination
Drift Decision Layer
Yes - Retrain
Drift Threshold Reached
Calculate Sample Drift
No - Recalculate
Sample classification
Drift Calibration Layer using Calibration Dataset
Tune Drift Boundaries in FeSA Feature set
Drift Boundary Calibration
Expected Drift Calibration

Figure 3.1: FeSAD Framework

3.3.2 FeSAD Summarised

- **FeSA Layer:** The FeSA layer generates feature sets using the FeSA feature selection algorithm, and the best-performing feature set is identified to use in the following two layers of the framework. The FeSA feature selection approach is proven to produce robust feature sets that show a degree of resistance to concept drift. The base layer generates feature sets and these feature sets flow onto the evaluation. The genetic layer takes the top performing feature sets from the base layer and this flows to the selection process where pairs of strong feature sets are bred together and this process is repeated until the performance of the generated offspring is no longer improving. The drift boundaries and values are calculated for the optimal and stored in the genetic layer. The drift values calculated are fed into the Drift Calibration Layer. The robustness of the feature sets is judged by their ability to reduce the performance degradation a classifier experiences due to concept drift. The FeSA layer is modified in the FeSAD framework to use a modified similarity metric that identifies potentially drifting samples in the training data set. Based on the training data set, the FeSA layer calculates the initial thresholds for abnormal and normal samples. The FeSAD framework is equipped to use different similarity metrics combined with prediction probabilities to measure concept drift; this approach is different to something like the NCM used in Transcend as it uses metrics derived from the underlying algorithm and statistical measures outside the underlying algorithm. Approaches like the NCM (Jordaney et al., 2017) are effective; however, there is room to add reliability seeing as the underlying algorithm is degrading due to concept drift; therefore, it would be logical not to derive all measures to combat drift from a degrading machine learning classifier as the decisions made by the algorithms will inevitably be affected by the drift.

- **Drift Calibration Layer:** The drift calibration layer uses a calibration dataset to measure the shifts in drift between the training and calibration set. The drift values calculated in the flow to the drift calibration layer and the change in the drift between the training and calibration data. The changes in drift are fed back to the optimal feature set and stored in the framework. The updated drift values are then fed into the Drift Decision Layer. The calibration dataset is a dataset that contains a different distribution of ransomware and benign samples; the ransomware in the calibration set is expected to evolve from the training ransomware and, therefore, should present concept drift when compared to the training data. The calibration layer analyses the drift observed in the calibration data compared to the training set and adjusts the thresholds for abnormal drift. The updated thresholds for drift are sent to the FeSA layer, and these thresholds are used in the drift decision layer. The main difference between the Drift Calibration Layer and other current work is that the Drift Calibration Layer aims to observe the changes in drift across different distributions to anticipate future drift; this allows for the theoretical expansion of the lifespan of the classifier used by the FeSAD Framework.

- **Drift Decision Layer:** The drift decision layer uses the drift thresholds determined using the drift calibration layer and the feature set produced by the FeSA layer to make decisions on incoming test samples. The drift decision layer calculates whether a sample shows drift beyond what is acceptable for benign or ransomware files. There is a user-set threshold for the proportion of samples displaying abnormal or unacceptable drift the system can encounter before retraining must be carried out.

3.3.3 FeSA Feature Ranker

The FeSA feature ranker identifies the features that present the highest information gain. Information gain reduces the complexity of the generated important features because random feature selection requires multiple selections to find the optimal set. The FeSAD framework controls ensure the feature sets have a base of essential features; this ensures high-quality feature sets. The Feature Ranker algorithm is proposed to decide which features need to be enforced in all feature sets. The feature ranker is the base of the FeSAD framework because it ranks features in order of importance and attaches a numerical value to this ranking. Information gain is calculated according to Eq 3.1. The ranker algorithm uses information gain to isolate the most important features; it determines information gain and then ranks features to gain information. Information gain is the reduction in entropy after a dataset is split on an attribute. Entropy is defined as a measure of randomness in information; therefore, the higher the entropy, the harder it is to conclude the data.

Information gain (IG) is the reduction in entropy when splitting a dataset on an attribute and is calculated in eq.3.1, c_i represents the ith class category i.e. ransomware or benign, and $p(c_i)$ is the probability of ith category. $p(c_i|x)$ is the conditional probability of the ith category given the feature x appears. and $p(c_i|\overline{x})$ is the conditional probability of the ith category given the feature x does not appear.

$$IG(x) = -\sum_{i=1}^{m} p(c_i)\dot{l}og(p(c_i)) + p(x)\sum_{i=1}^{m} p(c_i|x) \cdot log\, p(c_i|x) + p(\overline{x})\sum_{i=1}^{m} p(c_i|\overline{x}) \cdot log\, p(c_i|\overline{x}) \tag{3.1}$$

The feature set, taken from the feature ranker, is defined in eq.3.2. The variable a depends on the features chosen by the feature ranker; this is the number of important features chosen as "important". The feature ranker has a minimum threshold for features with an information gain equal to or greater than 0.5 to be considered an "important" feature. Based on experimental observations, features that tend to be most prominent across more than one distribution had a value greater than or close to 0.5. The information gain value of 0.5 was also achieved by a relatively small proportion of features, meaning the list of features is kept relatively small, and only the crucial features would be enforced as necessary for all feature sets. The extensive experimental observations yielded this value as the most appropriate for our dataset; however, this value will vary for other datasets and real-world networks. The data and the information gained derived from said data will invariably be different from data set to data set. Z is defined as the set of essential features, which every feature set is built around, $|Z|$ is the cardinality of this feature set. An algorithmic representation of the feature ranker is shown in Algorithm 1. The ranker eliminates features deemed to provide 0 information gain and features that provide inadequate information gain. The ranker identifies features with zero information gain and excludes them from the base and subsequent genetic layers.

$$|Z| = a \tag{3.2}$$

Algorithm 1 FeSA Feature Ranker

```
Input: Initial features $x_0, ..., x_i$
Output: Important feature set $Z$
1: for Initial Features $x_0$ to $x_i$ do
2:     Calculate Information Gain for each feature using:
       $IG(x_i) = -\sum_{i=1}^{m} p(c_i) + p(x)\sum_{i=1}^{m} p(c_i|x) \cdot log\, p(c_i|x) + p(\overline{x})\sum_{i=1}^{m} p(c_i|\overline{x}) \cdot log\, p(c_i|\overline{x})$
3:     if $IG(x_i) \geqslant 0.5$ then
4:         Add $x_i$ to important feature set $Z$
5:     end if
6:     Return important feature set $Z$
7: end for
```

Algorithm 1 shows the operation of the feature ranker. The feature ranker takes an initial set of features x_0 to x_i and calculates each feature's information gain IG. If the feature x_i has an information gain value above or equal to 0.5, it is added to the important feature set Z. Each feature in the important feature set is denoted as z_i.

3.3.3.1 FeSA Fitness Function

The fitness function used by FeSA calculates the average detection rate, information gain and accuracy amongst all feature sets in the current generation. Feature sets that achieve above average in all three criteria; detection rate, information gain and accuracy, are passed on to the next generation. The fitness function is used in the base and genetic layer; it uses the ranker's values, but indirectly as opposed to directly, in its calculations. The ranker will enforce features with the highest information gain and eliminate features with no information gain, thus ensuring that the feature sets produced in the base and genetic layers will provide as much information as possible while removing excess features. The fitness function initially used only detection or accuracy to measure a feature sets fitness for purpose; however, configurations with the ransomware detection rate and overall accuracy yielded high false positives in the test phase.

3.3.4 Base Layer

The FeSA base layer generates the initial population of feature sets required by a genetic algorithm. The base layer randomly generates feature sets from a pool of initial features, containing the important features defined by the feature ranker. The initial population is required to generate strong feature sets in the genetic layer; the main difference between the base layer and a regular population layer is that the ranker has already defined a set of features enforced in each generated feature set. The ranker enforcing important features in the base layer feature sets means the first generation of feature sets will have a higher performance than feature sets created with entirely random combinations of features. The starting generation at a high-performance level allows the subsequent generations to reach maximum performance earlier than a conventional genetic algorithm.

$$|N| = (\frac{r}{100} \cdot T(f)) + |Z| \tag{3.3}$$

The number of features selected per feature set is shown in eq.3.3, where r is a number less than 100 and, when divided by 100, gives the proportion of the original feature pool included in every generated feature set, and N is a feature set generated by the base layer. N is calculated as r divided by 100, which obtains a proportion of $T(f)$, $T(f)$ being all the of the features in the initial feature pool plus $|Z|$ which is the important feature set chosen by the ranker. Duplicates are prevented as the important feature set $|Z|$ is removed from the selection pool for the base feature sets as the important features are added to each feature set at the start of the feature set generation. Generating feature sets is repeated as often as the user defines and defines the population size for each generation. The number of repetitions is balanced with a defined size for each feature set; the repetitions should be proportional to the size of the feature sets. There are many features in the feature pool; therefore, possible combinations are regulated to avoid massive computational costs. FeSAD evaluates the feature sets generated using the initial population with a random forest classifier. The user's use of the underlying algorithm is flexible and determined based on their features and data. The random forest is used because of its performance with the API call and registry activity features; however, this approach is adaptable and can be used with any feature set and underlying algorithm. The feature sets are evaluated on overall accuracy, average information gain per feature and ransomware detection rate; therefore, only the most suitable feature sets are passed onto the next phase. The structure of the feature set generation layer is shown in algorithm 2. The abbreviations used in algorithm 2 are as follows, True Positive (TP), False Positives (FP), False Negatives (FN), and True Negatives(TN). Algorithm 2 shows the operation of the FeSA base layer. Features from the feature pool are taken, and feature sets are generated; these feature sets include the important feature sets Z, denoted as Y_i. The average detection rate, Information gain and accuracy of every feature set generated, Y_i, is calculated, and feature sets showing above-average performance are placed in the H, high-performance feature sets.

Algorithm 2 FeSA Base Layer

```
Input: Initial features $x_0, ..., x_i$, Important feature set $Z$,
Number of features to take from initial features $n$
Output: High performance feature sets $H$
 1: $m$ →maximum number of feature sets
 2: Average detection rate $d_r = 0$
 3: Average accuracy $a_r = 0$
 4: Total detection rate $t_d = 0$
 5: Feature set count $i = 0$
 6: while feature set count $i < m$ do
 7:     Generate feature set $Y_i$ taking $n$ features from Initial features
 8:     Add important features $Z$ to $Y_i$
 9:     Calculate detection rate of $Y_i$ using:
10:     True Positive Rate (TPR) = $\frac{TP}{TP+FN}$
11:     Calculate accuracy of $Y_i$ using:
12:     Accuracy (ACC) = $\frac{TPR}{TPR+TN+FP+FN}$
13:     Calculate information gain $IG(x_i)$ of $Y_i$
14:     $i$++
15: end while
16: Calculate average detection rate $d_r$ using:
17: $d_r = \frac{\sum_{i=0}^{m} TPR(Y_i)}{m}$
18: Calculate average accuracy $a_r$ using:
19: $a_r = \frac{\sum_{i=0}^{m} ACC(Y_i)}{m}$
20: Calculate average information $g_r$ using:
21: $g_r = \frac{\sum_{i=0}^{m} IG(x_i)}{m}$
22: for $Y_0 ..., Y_m$ do
23:     if detection rate & accuracy & information gain of $Y_i > d_r$ & $a_r$ & $g_r$ then
24:         Add $Y_i$ to high-performance feature sets $H$
25:     end if
26: end for
        return $H$
```

Algorithm 2 shows the operation of the base layer. The base layer starts with a feature pool $x_0, \ldots, x_i$ and uses the feature ranker to identify a set of essential features Z. The number of maximum feature sets generated by the base layer is defined as m, and the user defines this. The base layer will generate feature sets from the feature pool and then add the important feature set, Z, to each feature set generated. The ransomware detection rate, the overall accuracy and information gained by the feature set is calculated for each feature set; this helps determine the overall health of the feature set. The average ransomware detection rate d_r, accuracy a_r and information gain g_r of every feature set in the base layer is also calculated. Each feature set's detection rate, accuracy and information gain are compared with the average metrics for all the feature sets in the base layer and feature sets that have a detection rate, accuracy and information gain higher than the average for the population are put in the high-performance feature set group, H. The high-performance feature set group is passed onto the genetic layer.

3.3.5 FeSA Genetic Layer

The FeSA genetic layer acts as the crossover phase in a genetic algorithm. The genetic layer is designed to generate strong feature sets; the strong feature sets from the base layer are combined to create these feature sets through a breeding approach. The feature sets are expected to reach optimal performance after the crossover phase has been completed multiple times. The feature sets produced in the genetic layer are candidates for the optimal feature set. The genetic layer uses feature sets from the base layer and will combine the high-performance feature sets using the uniform crossover method to yield higher-performance feature sets. The genetic layer has the advantage of containing important features enforced in the base layer. The theory behind the crossover phase is that combining features from high-performance feature sets will yield higher-performance feature sets in an evolutionary manner where the performance increases with every new generation.

The genetic selection layer is a breeding mechanism for the highest-performing feature sets taken from the base layer. The genetic layer comprises ”parent” and ”offspring” feature sets. The ”parent” feature sets are high-performance feature sets from the base layer. The ”offspring” feature sets are produced by choosing two-parent feature sets and combining them with a crossover function. Figure 3.2 shows the operation of the crossover phase.

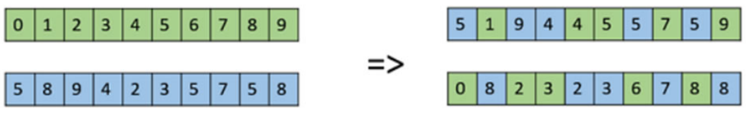

Figure 3.2: Feature Breeding

High-performing ”parent” feature sets will be combined using uniform crossover, generating ”offspring” feature sets. The offspring feature sets produced should perform better than the preceding generation; if this is not the case the algorithm will terminate and the optimal feature set will be identified from the last generation that produced feature sets stronger than the previous generation. Uniform crossover takes two ”parent” feature sets and combines them. For each corresponding feature in each parent feature set, the feature the offspring feature set receives is determined by a probability of 0.5. The probability is representative of a coin flip where there is an equal probability of both parents’ features having an equal chance of being present in the offspring. The crossover function is user-defined; however, the FeSAD framework needs to enforce particular features into feature sets; the uniform crossover function is the most efficient way to retain the important features during the breeding phase. The offspring feature sets are evaluated based on accuracy, detection, and overall information gain per feature and the feature set with the highest average of these metrics is chosen as the optimal feature set. As per experimental results, this phase requires only one generation to generate the optimal feature set; performance does not improve beyond the first generation showing the efficiency of this approach. The structure of the genetic layer is shown in algorithm 3.

Algorithm 3 FeSA Genetic Layer

```
Input:  High performance feature sets $H$
Output:  Optimal feature set $O_i$
1: $m$ →Max feature set count
2: $n$ →Current feature set count
3: Offspring feature set $T$
4: while $n < m$ do
5:     Select random base feature set 1 $H_{rand1}$ from set $H$
6:     Select random base feature set 2 $H_{rand2}$ from set $H$
7:     Perform uniform crossover using $H_{rand1}$ and $H_{rand2}$ & Generate mixed feature set $O_i$
8:     if Duplicate features are detected then
          Replace duplicate feature with a random feature $x_i$ from feature pool
9:     end if
10:    Add $O_i$ to offspring set $T$
11: end while
12: return Optimal $O_i \in T$ which has the highest average detection rate, information gain &
    accuracy.
```

Algorithm 3 shows the genetic layer of the FeSA architecture. Features in a feature set are numbered from 1 to n; each feature is assigned a number; however, they are placed in the feature set according to the order in which they are randomly chosen therefore, two feature sets could have the same feature but in different slots. To avoid duplicates, the algorithm performs the crossover but checks if the chosen feature is already present in the offspring feature set; the alternative is then chosen and if this is also a duplicate, the mutation function is activated. The High-performance feature sets from the base layer. Two random high-performance feature sets are selected H_rand1 and H_rand2 , and uniform crossover is carried out to mix the two high-performance feature sets to create a new feature set O_i. Mixing the high-performance feature sets is repeated m times until completion. The best performing of these newly generated feature sets is stored in set T. The best-performing feature set out of the newly generated feature sets in T is selected as the feature set used by FeSAD for classification. The genetic layer terminates if the current generation produces no offspring that outperforms the previous generation.

3.3.6 Mutation

The mutation function in a genetic algorithm is the introduction of diversity. A mutation would mean an offspring feature set inheriting a feature not present in either parent feature set in a feature selection context. During the crossover phase, duplicate features are prohibited from being in a feature set. If there is a feature set with duplicate features, duplicates will be replaced with a random feature from the feature pool, leading to a 0.01% mutation rate. The low mutation rate eliminates unnecessary randomness from the FeSA architecture.

3.3.7 FeSA Layer Summary

The feature selection component of FeSAD is essential because of the need for long-lasting features that provide an initial layer of protection against concept drift. Feature selection is a crucial aspect of machine learning classifiers because they reduce the dimensionality of a dataset and increase efficiency. The hooked API calls in a Windows system numbers in the hundreds; therefore, it was crucial to identify essential features instead of saturating the classifier with hundreds of irrelevant features. Identifying important API features is critical because the behaviour of ransomware at its core has to remain constant; ransomware will always aim to encrypt files and hold them for ransom. Using a genetic algorithm to adapt feature sets to concept drift has been proven effective (Ghomeshi et al., 2019). The genetic algorithm used for the FeSA component in the FeSAD framework is not conventional because it uses information gain to identify essential features before the feature sets are built. Using the feature ranker reduces the number of generations needed for the feature selection algorithm to reach an optimal solution.

3.3.8 Base Drift Calculations

Once the FeSA layer has generated the optimal feature set through its genetic algorithm, the base concept drift values for the dataset using the optimal feature set are calculated. The base drift values are calculated using combinations of prediction probabilities or prediction metrics and similarity or distance metrics, and this process is outlined in section 3.5. The base drift calculations are done to assess the level of variation in the training data and what can be considered normal and abnormal based on the drift values obtained from the training data. The experiments in this research use prediction probabilities as the best-performing models on the data were probabilistic; however, this can be interchangeable with values that can provide a metric for prediction quality, such as SVM with a distance from the hyperplane. The FeSA layers calculate the drift present in the initial dataset and will set thresholds and expectancies for drift based on the drift shown by correct and incorrect classifications in the cross-validation phase. The drift values present in the base dataset are used to calculate the changes in concept drift over time and the expected levels of concept drift in the future. The FeSA layer collaborates with the drift calibration layer to adjust its drift tolerances and expectancies based on the changes in drift it observes between the base drift values in the FeSA layer and the drift values observed in the drift calibration layer.

- **Expected Drift:** Drift is considered normal based on the analysis of the training dataset.
- **Abnormal Drift:** Drift is considered abnormal based on the analysis of the training dataset.
- **Maximum Drift:** Drift considered beyond what is acceptable for a classification; drift beyond the maximum threshold suggests a misclassification.

3.4 Drift Calibration Layer

Concept drift represents the relationship between a sample's attributes and its classification over time. The FeSAD framework attempts to quantify the concept change over time using a distance metric and the classifier's prediction probability. An easy-to-interpret numerical figure can allow users to identify a sample's normal or abnormal behaviour. The Drift Calibration Layer allows the FeSAD framework to predict reliably under concept drift. The Calibration Layer takes the drift values extracted from the feature set in the Base Layer and compares the differences in drift between the base dataset and the calibration datasets. The Drift Calibration Layer will analyse ransomware's drift patterns in different periods to determine what acceptable, abnormal, and maximum drift thresholds should be in place. The Drift Calibration Layer adjusts the drift acceptance values calculated in the Base Layer and determines the thresholds or the drift predictions in the

Drift Decision Layer. The Drift Calibration Layer is the most important and novel aspect of the FeSAD framework, as it allows a classifier to work together with a concept drift calculation system to remain robust and reliable under concept drift.

3.4.1 Pre-set Values

· **Unreliable Prediction Threshold** The framework will have a predefined number of unreliable predictions which, once exceeded, the system will need to be trained regardless of its accuracy. The drift adaptation system should not be relied on solely if it becomes obvious that the algorithm the framework is built on has no idea what it is doing and cannot be certain of a large proportion of its classifications; the retraining process should be executed so the framework can be better suited to the current concept.

3.4.2 HEOM (Heterogeneous Euclidean Overlap Metric): Phase 1

The HEOM metric was chosen as a starting point for the drift metric design as it was already proven effective in drift detection by research reviewed in the literature studies such as Tsymbal et al. and Singhal et al., who have used the HEOM metric directly in the malicious URL space to detect concept drift in detection systems. The HEOM metric is built from the Euclidean distance, so provided that the features chosen provide a distinguishable difference between ransomware and benign behaviour, a metric like HEOM should work effectively when identifying drift. The HEOM metric also provides a versatile way of normalising values for features that may differ numerically in size. The heterogeneous Euclidean overlap metric (HEOM) is a distance metric that measures distances between two data samples with continuous and nominal attributes. Using the HEOM metric gives an insight into how close or far away statistically a sample is from the other samples it has been classified with, giving insight into how reliable a prediction may be. The flexibility of HEOM allows the feature set to be modified in the future if necessary; however, it currently uses only continuous features. The HEOM metric measures the distance between two values x and y given an attribute a. The FeSAD framework uses a modified implementation of the HEOM metric to measure concept drift per sample and overall concept drift in a system. The equation for the standard HEOM calculations is shown in Equation 3.4 and Equation 3.5.

$$d_{heom}(x_1, x_2) = \sqrt{\sum_{a=1}^{m} heom_a^2(x_1, x_2)} \tag{3.4}$$

Where

$$heom_a(x_1, x_2) = \frac{|x_{1a} - x_{2a}|}{range_a} \tag{3.5}$$

The standard HEOM equation measures the distance between two samples x_1 and x_2 in relation to feature a, where m is the number of features. Equation 3.5 ensures no discrepancies due to large variances between the values for each feature by normalising them with $range_a$. The HEOM measurements for each sample are taken as a ratio by measuring the HEOM distance of a sample from all other ransomware and benign samples and dividing these values. Equation 3.6 shows how the HEOM distance from ransomware for a sample is calculated, and equation 3.7 shows how the HEOM distance from benign samples is calculated. In both cases, n represents the number of benign or ransomware samples the sample is compared to during the calculation. The instance number is represented by i_r or i_b and is compared to another ransomware instance j. The HEOM metric is used in the FeSAD framework uses the distance metrics of HEOM to calculate the distance of a sample from all other samples in the class it is placed in. The HEOM metric also measures the distance from the opposite class to the class where a sample has been placed. In the case of FeSAD, a sample classified as ransomware will be analysed to measure its HEOM distance from other ransomware and benign samples. The base HEOM values will show how far ransomware is from other ransomware and how far benign samples are from ransomware. The FeSA Layer uses the HEOM metric to measure the average HEOM distance from ransomware to ransomware and benign to ransomware in the feature set generated by the FeSA layer; these values are then used in the next phase of the drift calculation.

$$heom_r = \frac{\sum_{i,j=1}^{n} d_{heom}(x_{i_r}, x_{j_r})}{n} \tag{3.6}$$

$$heom_b = \frac{\sum_{i,j=1}^{n} d_{heom}(x_{i_b}, x_{j_r})}{n} \tag{3.7}$$

3.4.3 HEOM ratio calculation

The HEOM ratio h, for a sample, represents the ratio of a sample's distance to ransomware and distance to benign files; the average HEOM ratio for ransomware is significantly lower than the average HEOM ratio value for benign files. The HEOM ratio value that uses the HEOM distance of the sample from all ransomware samples, $heom_r$ and the HEOM distance of the sample from all benign files, $heom_b$; this assumption is based on the fact that the HEOM is built on the Euclidean distance; therefore, statistically ransomware and benign files should be distinguishable from each other as long as the features chosen are sound. The calculations for generating the HEOM ratio value for a sample are shown in equation 3.8. The HEOM ratio value h should be much smaller for ransomware than for benign samples due to how the ratio calculation is constructed. Benign files will have a larger h than ransomware files because their average distance to ransomware $heom_r$ should be significantly larger than their distance to benign files $heom_b$. The ratio value is calculated equally for both classes, so the HEOM ratio is relative to the distance to ransomware for both benign and ransomware classes.

$$h = \frac{heom_r}{heom_b} \tag{3.8}$$

3.4.3.1 Prediction Probability

Prediction probability is derived from probabilistic classifiers that use metrics to determine the most likely class prediction. The higher the prediction probability, the more confident a prediction can be. Prediction probability is extractable from all probabilistic machine learning algorithms, and the FeSAD framework is designed to incorporate prediction probability into its classifications. Prediction probability can be substituted with similar machine learning metrics like SVM and the distance from the hyperplane or a clustering algorithm's distance from the centroid.

3.4.3.2 Prediction Weights

A sample's HEOM ratio can tell us how far or close to its classification a sample is, but it cannot provide 100% accuracy as with any metric. A weight is introduced to the HEOM values to balance any uncertainty the HEOM calculations introduce. Based on initial observations of ML performance with my data, the prediction probability appears to have a strong rate of identifying incorrectly classified samples, as most observed misclassifications had a lower prediction probability than correctly classified samples. Combining two high-performing metrics with a strong possibility of identifying misclassifications will increase the reliability of the drift metric and the probability it can correctly identify misclassifications, with one of the metrics taking input from the underlying machine learning algorithm and the second metric being the objective of the underlying algorithm. The weight for benign predictions is calculated by dividing the probability of the benign prediction by the average probability of an incorrect benign prediction. The introduction of the weights eliminates any false positives introduced by using the HEOM calculations alone. The calculations used for generating the weights for ransomware, w_r and benign predictions, w_b are shown in equation 3.11 and equation 3.12, respectively. For the weights added to the HEOM ratio, p is the probability of the prediction, $\bar{p_{x_r}}$ and $\bar{p_{x_b}}$ being the average probability of incorrect ransomware or benign prediction respectively. The average incorrect prediction values are calculated using the training set generated by FeSA. Equation 3.9 and 3.10 show the process for calculating the average probability for an incorrect ransomware and benign prediction with n being the number of incorrect predictions for either ransomware or benign files.

$$\bar{p_{x_r}} = \frac{\sum_{i=0}^{n} p_{xr_i}}{n} \tag{3.9}$$

$$\bar{p_{x_b}} = \frac{\sum_{i=0}^{n} p_{xb_i}}{n} \tag{3.10}$$

$$w_r = \frac{p}{\bar{p_{x_r}}} \tag{3.11}$$

$$w_b = \frac{p}{\bar{p_{x_b}}} \tag{3.12}$$

3.4.3.3 Weighted HEOM Calculation

The weighted HEOM combines a sample's HEOM and prediction probability in our dataset. The weighted HEOM measures drift, the average drift in our dataset, the expected drift, and the drift of individual test samples. The calculation for the weighted HEOM for a ransomware prediction is shown in equation 3.13, and the calculation for the weighted HEOM of a benign prediction is shown in equation 3.14. A logarithmic value is used for both calculations to shrink the values into a distinguishable range. The calculations for both values differ as the desired effect of the weight is different for both classifications.

$$hw_{b_i} = \log(h_{i_b} \cdot w_b) \tag{3.13}$$

$$hw_{r_i} = \log(\frac{h_{i_r}}{w_r}) \tag{3.14}$$

Ransomware weighted HEOMs are shrunk if the prediction weight is high; however, benign weighted HEOMs increase if the prediction weight is high; this is because the HEOM ratio is measured based on how far each sample is from ransomware on average, meaning that benign weighted HEOMs hw_{b_i} being higher than ransomware hw_{r_i} is desired. Low prediction weights would have the inverse effect on the HEOM values; ransomware weighted HEOMs would increase, and benign HEOMs would decrease. The overview of this approach is that weaker predictions would have a greater effect on the drift value; this is because the weaker a prediction, the further away from the norm a prediction is as the standard for predictions is high from the baseline classifier so that weaker predictions would imply a higher level of drift and greater uncertainty from the classifier. Algorithm 4 shows how the weighted HEOM calculations are made. The HEOM distance values from ransomware samples, $heom_r$ and benign samples $heom_b$ are calculated for each sample in the training set, and a ratio value is calculated to generate h. The weighted HEOM calculations are carried out for correct ransomware and benign classifications in the training set, and the average weighted HEOM for both correct benign and ransomware classifications are generated. Algorithm 4 has a computational complexity of $O(n.logn)$).

Algorithm 4 Weighted Calculations

```
Input: Ransomware Training Samples $x_{i_r}, ..., x_{n_r}$, Benign Training Samples $x_{i_b}, ...x_{n_b}$,
Ransomware sample prediction weights $w_r$, Benign Sample prediction weights $w_b$
Output: Average Weighted HEOMs $\bar{hw}_{r_c}$, $\bar{hw}_{b_c}$
1: for Ransomware Samples in training set $x_i....x_n$ do
2:     Calculate HEOM value from Ransomware samples
3:     $heom_{r_i} = \frac{\sum_{j=1}^{n} d_{heom}(x_{j_r}, x_{n_r})}{n}$
4: end for
5: for Benign Samples in training set $x_i....x_n$ do
6:     Calculate HEOM value from Benign samples
7:     $heom_{b_i} = \frac{\sum_{j=1}^{n} d_{heom}(x_{j_b}, x_{n_b})}{n}$
8:     Calculate HEOM ratio for samples
9:     $h_i = \frac{heom_{r_i}}{heom_{b_i}}$
10: end for
11: for training ransomware samples $x_i...x_n$ do
12:     Calculate Weighted HEOM
13:     $hw_{r_i} = \log(\frac{h_i}{w_{r_i}})$
14: end for
15: for training benign samples $x_i...x_n$ do
16:     Calculate Weighted HEOM $hw_{b_i} = \log(h_i \cdot w_{b_i})$
17: end for
18: Calculate average correct benign weighted HEOM only using the weighted HEOM of correct
    benign predictions:
19: $\bar{hw}_{b_c} = \frac{\sum_{i=0}^{n} hw_{bc_i}}{n}$
20: Calculate average correct ransomware weighted HEOM only using the weighted HEOM of
    correct ransomware predictions:
21: $\bar{hw}_{r_c} = \frac{\sum_{i=0}^{n} hw_{rc_i}}{n}$
22: return $\bar{hw}_{r_c}$ return $\bar{hw}_{b_c}$
```

3.4.3.4 Expected Drift Threshold Generation

The FeSAD framework uses drift calculations that formally measure how far from an expected point a value is; our drift values attempt to formalise and quantify concept drift in a machine learning detection system. FeSA calculates the average weighted HEOMs for correct benign and ransomware classifications during the training phase. The average weighted HEOM for correct ransomware classifications is $\bar{hw}_{r_c}$ and the average weighted HEOM for correct benign predictions is $\bar{hw}_{b_c}$. The drift calculations measure the difference between the average weighted HEOM in the training set and the weighted HEOM of an individual sample. The weighted HEOM for a sample i is represented with hw_{r_i} for ransomware and hw_{b_i} for benign files. The calculation for drift in a sample classified as ransomware is shown in equation 3.15, and the calculation for drift in a sample classified as benign is shown in equation 3.16.

$$D_{r_i} = hw_{r_i} - \bar{hw}_{r_c} \tag{3.15}$$

$$D_{b_i} = hw_{b_i} - \bar{hw}_{b_c} \tag{3.16}$$

The calibration set shown in Figure 3.1 is used to calculate the expected drift in the system; the expected drift is calculated to allow the system to know what changes in weighted HEOM values to expect. Expected drift D_{e_r} for ransomware and D_{e_b} for benign is calculated by taking the average drift in the calibration set and the average increase in drift f, in datasets from different ransomware distributions. The average increase in drift f is calculated by measuring the average drift in the training data and comparing the average drift in the samples showing increased drift in the calibration data. The calculation for the average drift for ransomware in the calibration set is shown in equation 3.17, and the average benign drift for benign predictions is shown in equation 3.18.

$$\bar{D_r} = \frac{\sum_{i=1}^{n} D_{r_i}}{n} \tag{3.17}$$

$$\bar{D_b} = \frac{\sum_{i=1}^{n} D_{b_i}}{n} \tag{3.18}$$

Expected drift is calculated by combining the average drift in the calibration set and the average increase in drift for ransomware f_r and benign files f_b from different ransomware and benign distributions, so in this case, the different distributions would be training and calibration sets; this is shown in equation 3.19 and equation 3.20 for expected drift in ransomware and benign samples, respectively. The calibration dataset contains data from a different distribution to the training dataset; an example would be the training data from 2013-2015, a calibration set from 2018 data and a test set of data from 2019.

$$D_{e_r} = \bar{D_r} \cdot f_r \tag{3.19}$$

$$D_{e_b} = \bar{D_b} \cdot f_b \tag{3.20}$$

Algorithm 5 shows how the training and calibration sets drift calculations are carried out. The drift for correct ransomware and benign classifications is calculated using the training set. The increase in drift,f_r and f_b for both correctly classified ransomware and benign files is calculated using the calibration set. The expected drift for benign and ransomware samples is calculated by combining the expected drift calculated in the training set and the increased drift noted in the calibration dataset. Algorithm 5 has a computational complexity of $O(n)$.

Algorithm 5 Drift Calculations in FeSA & Drift Calibration Layer

```
Input:  Weighted HEOMs $\bar{hw}_{r_c}$, $\bar{hw}_{b_c}$, $f_r$, $f_b$
Output: Expected Drift, $D_{b_r}$, $D_{e_r}$
 1: for All correct classifications in the training set do
 2:     Calculate drift for correct ransomware classifications:
 3:     $D_{r_i} = hw_{r_i} - \bar{hw}_{r_c}$
 4:     Calculate of drift for correct benign classifications:
 5:     $D_{b_i} = hw_{b_i} - \bar{hw}_{b_c}$
 6: end for
 7: Calculate average drift in training set for benign files:
 8: $\bar{D}_r = \frac{\sum_{i=1}^{n} D_{r_i}}{n}$:
 9: Calculate average drift in training set for ransomware files:
10: $\bar{D}_b = \frac{\sum_{i=1}^{n} D_{b_i}}{n}$
11: Change in ransomware drift calculated using the calibration set: $f_r$
12: Change in benign drift calculated using the calibration set: $f_b$
13: Calculate expected ransomware drift using: $D_{e_r} = \bar{D}_r \cdot f_r$
14: Calculate expected benign drift using: $D_{e_b} = \bar{D}_b \cdot f_b$
15: return $D_{e_r}$ return $D_{e_b}$
```

3.4.3.5 Drift Thresholds

The drift thresholds represent levels of drift that require different courses of action; the drift calibration layer allows the FeSAD framework to define what is classified as normal and abnormal drift. Normal drift would require no action as it is within the levels of drift the framework expects. Abnormal drift is beyond the standard threshold for drift; however, it is not beyond the maximum threshold. Abnormally drifting samples are recorded, and too many abnormal samples suggest a need to retrain. The drift calibration layer also defines drift boundaries; drift beyond the drift boundary would suggest a misclassification.

Abnormal Drift Abnormal drift is defined as a sample displaying drift beyond expected drift D_{e_r} or D_{e_b}. The FeSAD framework uses a drift factor metric that determines whether a test sample displays abnormal or normal drift; it is shown in equation 3.21 and equation 3.22. D_r is the drift displayed by a ransomware test sample and D_b is the drift displayed by a benign test sample.

$$D_{f_r} = \frac{D_r}{D_{e_r}} \tag{3.21}$$

$$D_{f_b} = \frac{D_b}{D_{e_b}} \tag{3.22}$$

If the D_{f_r} value for a sample classified as ransomware is above 1, this would mean the sample is showing abnormal drift for a ransomware classification; this is also the case for D_{f_b} of a sample classified as benign. The abnormal drift threshold is decided by the expected drift value D_e.

The expected drift value taking drift from many different distributions allows the abnormal drift threshold boundary to identify true outliers while not flagging samples that represent drift that can be expected. The threshold taking concept drift into account should extend the time between retraining while flagging samples displaying abnormal drift. Once z number of samples have been identified as showing abnormal drift, the system needs to be retrained. The user defines the z value.

Maximum Drift FeSAD calculates the drift thresholds using the training and calibration sets. D_{max} is a level of drift that a sample cannot exceed. Exceeding the D_{max} would suggest an incorrect prediction, so a benign sample displaying drift beyond D_{max} would be considered ransomware and vice versa for samples classified as ransomware. D_{max} is defined as the point at which a sample, according to the weighted HEOM metric, is closer to the opposite class it is classified as. An example of this scenario would be a sample classified as benign by the underlying classifier, but the sample's drift level would exceed D_{max}, therefore it would be closer to ransomware than benign files; it would be reclassified as ransomware. D_{max} is calculated based on the training and incorrect classifications in the calibration set.D_{max} is calculated using the training set shown in equation 3.23. The midpoint is defined and is D_{max}. If the drift of a sample is beyond D_{max}, then it will have a $D_{f_{max}}$ greater than 1; this is determined by equation 3.24.

$$D_{max} = \frac{h\bar{w}_{r_c} - h\bar{w}_{b_c}}{2} \tag{3.23}$$

$$D_{f_{max}} = \frac{D_i}{D_{max}} \tag{3.24}$$

3.4.4 Drift Decision Layer

The Drift Decision Layer uses the drift thresholds calculated using the FeSA and Drift Calibration Layer to accurately and reliably classify samples showing concept drift. The FeSAD framework uses its drift thresholds defined by the FeSA and calibration layer to identify each sample's drift value D_i and decide if it is abnormal and beyond D_e or drift beyond the drift boundary, $D_{f_{max}}$. The Drift Decision Layer keeps track of the abnormal samples and will recommend retraining once the threshold, z, for abnormal drifting samples has been reached. Algorithm 6 shows how the drift decision layer works when determining whether a classification shows unacceptable drift. The expected drift values from algorithm 5 determine whether a sample displays abnormal or unacceptable drift. There is a defined threshold z for the amount of abnormal or unacceptable samples the system can encounter before retraining is advised. Algorithm 6 has a computational complexity of $O(n)$.

Algorithm 6 FeSAD Drift Decision Layer

Input: Expected Benign Drift D_{e_b}, Sample Drift D_i, Max Drift D_{max}, max abnormal samples z

Output: Drift Max Factor $D_{f_{max}}$

```
1: Abnormal count $a$
2: Drift Factor $D_{f_b} = \frac{D_i}{D_{e_b}}$
3: **if** $D_{f_b} > 1$ **then**
4:     $D_i$ = abnormal drift
5:     $a = a + 1$
6: **end if**
7: Drift Factor Max $D_{f_{max}} = \frac{D_i}{D_{max}}$
8: **if** $D_{f_{max}} > 1$ **then**
9:     $D_i$ = unacceptable drift
10:    $a = a + 1$
11: **end if**
12: **if** $a > z$ **then**
13:    Retrain Recommended
14: **end if**
15: **return** $D_{f_{max}}$
```

3.4.5 Underlying Algorithms

The FeSAD framework functions with almost any machine learning or probabilistic deep learning algorithm. The FeSAD framework uses a similarity metric as an independent function that does not rely on the machine learning classifier and the prediction probability metric from the underlying algorithm. Prediction probability is easily extracted from an algorithm that uses probability to represent the most likely classification. The probability of a prediction is represented as $P(y|x)$; the closer the value is to 1, the more reliable the prediction is. Prediction probabilities for a two-class problem like ransomware detection will always be above 0.5; therefore, it is essential to scale the validity of a prediction probability based on this. In theory, the underlying algorithm will degrade as it experiences concept drift, and the prediction probability of classifications will begin to reduce. In addition to being compatible with various underlying algorithms, the FeSAD framework offers the flexibility to use different algorithms in the FeSA Layer and the Drift Calibration and Decision Layers; the possibility of different algorithms working together could be advantageous for detection.

3.5 HEOM Alternatives: Phase 2

We selected similarity metrics to test the performance and flexibility of the FeSAD framework. The FeSAD framework was designed to have as many interchangeable parts as possible, allowing it maximum freedom to adapt to different concept drift scenarios. The following subsections will formalise the approach used by the FeSAD framework with the HEOM metric to allow the framework to use different similarity metrics in the same context as the previously presented HEOM metric. The application of the similarity metrics tested is precisely the same as the application of the HEOM metric; however, thresholds and allowances for drift have to be recalculated and calibrated due to the different values produced by another metric. This component of the FeSAD framework operates under the assumption that the underlying features will show statistical differences when concept drift is occurring; additionally the use of probabilities or alternative algorithm based measures will confirm the concept drift or provide an alternative view. The underlying assumption is that the API calls identified by the FeSA layer will provide an accurate baseline to distinguish between benign and ransomware files and significant statistical and probabilistic deviations from these baselines imply misclassification. Similarity metrics like HEOM capture statistical differences between samples; the only instance where concept drift would directly affect the drift measurement metrics is if the ransomware files start to behave exactly like the benign files with zero distinguishable difference between the two.

3.5.1 Distance Ratio

The Distance Ratio represents the ratio of a sample's distance from ransomware to a samples distance to benign files. The distance is measured using the respective similarity measure. The distance ratio metric measures the ratio of the distance of a sample to ransomware files to the distance of a sample to benign files. Eq.3.25 and 3.26 measure the average distance of a sample from ransomware and benign files respectively and Eq.3.27 is the ratio value of these two values. Benign files will have a larger $dist_i$ than ransomware files because their average distance to ransomware $dist_r$ should be significantly larger than their distance to benign files $dist_b$.

$$dist_r = \frac{\sum_{i,j=1}^{n} d_{(x_{i_r}, x_{j_r})}}{n} \tag{3.25}$$

$$dist_b = \frac{\sum_{i,j=1}^{n} d_{(x_{i_b}, x_{j_r})}}{n} \tag{3.26}$$

$$dist_i = \frac{dist_r}{dist_b} \tag{3.27}$$

3.5.1.1 Prediction Weights

The prediction weights are shown below in Eq.3.28 and 3.29. The weights are calculated by dividing the prediction probability of the sample by the average incorrect prediction for ransomware $\bar{p_{x_r}}$ for ransomware and $\bar{p_{x_b}}$ for benign predictions. The average prediction probabilities for incorrect classifications are derived from the training set.

$$w_r = \frac{p_i}{\bar{p_{x_r}}} \tag{3.28}$$

$$w_b = \frac{p_i}{\bar{p_{x_b}}} \tag{3.29}$$

3.5.2 Weighted Distance Calculation

The weighted distance combines prediction probability and the metrics extracted from the distance or dissimilarity measure. The weighted distance calculation for samples classified as ransomware is shown in Eq.3.30. and the weighted distance for a sample classified as benign is shown in Eq.3.31. The weights are designed to shrink ransomware distance values if the prediction probability is high and increase benign distance values if the prediction probability is high therefore creating greater separation between samples when the algorithm is confident in its predictions and samples are statistically similar to its respective predicted class.

$$dw_{b_i} = \log(dist_i \cdot w_b) \tag{3.30}$$

$$dw_{r_i} = \log(\frac{dist_i}{w_r}) \tag{3.31}$$

3.5.3 Drift Thresholds

The thresholds for drift boundaries that determine drifting samples and classifications that have a high probability of being wrong are calculated using Eq.3.22 to 3.23 and are applied to the alternative metrics in the same way.

3.5.4 Canberra Metric

The Canberra Distance metric is a weighted and refined version of the Euclidean distance; this metric can be used to calculate the distance between points in a vector space and is able to handle data with high dimensionality. The Canberra Distance metric has been used effectively in computer intrusion detection by Emran & Ye and is proven effective when detecting dissimilarity between samples in high dimensionality data by Shirkhorshidi et al. The Canberra metric is presented below.

$$d(p,q) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |p_i|} \tag{3.32}$$

The distance between vector p and vector q is calculated using the absolute values of p_i and q_i. The i^{th} value can be used per feature.

The Canberra Distance metric calculates the distance between therefore it is necessary to calculate the average distance between ransomware and other ransomware and the average distance between ransomware and benign files. The calculations work the same way as with the HEOM metric calculations in section 3.4.2

$$canb_r = \frac{\sum_{i,j=1}^{n} canb(x_{i_r}, x_{j_r})}{n} \tag{3.33}$$

$$canb_b = \frac{\sum_{i,j=1}^{n} canb(x_{i_b}, x_{j_r})}{n} \tag{3.34}$$

3.5.4.1 Canberra Ratio Calculation

$$c_i = \frac{canb_r}{canb_b} \tag{3.35}$$

The c_i represents the ratio value of Canberra distance from ransomware over Canberra distance from benign files; therefore, the c_{ratio} value for ransomware files should be smaller than for benign files as all measurements are taken in relation to ransomware and benign files should be statistically further away from ransomware files.

3.5.5 Weighted Canberra Metric

The calculations for the weighted Canberra Metric are shown below in Eq.3.36 and 3.37.

$$cw_{b_i} = \log(c_i \cdot w_b) \tag{3.36}$$

$$cw_{r_i} = \log(\frac{c_i}{w_r}) \tag{3.37}$$

3.5.6 Cosine Similarity

Cosine similarity measures the angle between two non-zero vectors. The cosine similarity measures similarity by calculating the cosine of the angle between two vectors. In our case, the vector will be ransomware or benign samples. Although the angles are those that are measured, it is intended to provide the linear distance between the data points. Cosine similarity dictates the higher the value, the higher the similarity between the two vectors (Hidalgo et al., 2019). The Cosine Similarity metric has been successfully used to detect concept drift by Hidalgo et al. Cosine similarity is independent of vector length and is suitable for data with high dimensionality; it has been commonly used to measure similarity between documents irrespective of their size difference (Shirkhorshidi et al., 2015). Eq.3.38 shows the calculations used for calculating the cosine similarity with a and b considered samples and n being considered a feature.

$$cos\theta = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2}\sqrt{\sum_1^n b_i^2}} \tag{3.38}$$

Eq.3.39 calculates the average cosine similarity between ransomware and other ransomware files and Eq.3.40 calculates the average cosine similarity between ransomware and benign files.

$$cos_r = \frac{\sum_{i,j=1}^n cos(x_{i_r}, x_{j_r})}{n} \tag{3.39}$$

$$cos_b = \frac{\sum_{i,j=1}^n cos(x_{i_b}, x_{j_r})}{n} \tag{3.40}$$

3.5.6.1 Cosine Ratio Calculation

$$cos_i = \frac{cos_r}{cos_b} \tag{3.41}$$

The cos_i represents the ratio value of Cosine similarity from ransomware over Cosine similarity from benign files.

3.5.7 Weighted Cosine Metric

The calculations for the weighted cosine are shown below in Eq.3.42 and 3.43. We avoid using a log value as the cosine similarity metric already has a clear bounding between -1 and 1; therefore, it does not require further normalisation to be effectively integrated with the prediction probability weights.

$$cosw_{b_i} = (c_i \cdot w_b) \tag{3.42}$$

$$cosw_{r_i} = (\frac{c_i}{w_r}) \tag{3.43}$$

3.5.8 Pearson Correlation

The Pearson correlation is a similarity metric commonly used to cluster data in the genetic expression field (Shirkhorshidi et al., 2015). The Pearson correlation metric is presented in Eq.3.44, where there is a comparison between x and y using the i^{th} value for a sample x and y. In the context of data samples, the Pearson correlation measures the correlation between two samples using each sample's features i with the maximum of n features. The mean value for each feature i is represented as $\overline{x}$ for the first samples and $\overleftarrow{y}$ for the second sample.

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}} \tag{3.44}$$

The Pearson correlation is effective as a similarity metric when used with high dimensionality data according to the study conducted by Shirkhorshidi et al. The Pearson correlation has the ability to clearly define the similarity between two samples which makes it appropriate to work with our data. Shirkhorshidi et al. state the Pearson correlation provides strong results with high dimensional data; however, our data would not be considered high dimensional. Using a metric that may not be effective on its own with low-dimensional data gives the FeSAD framework a good test and allows us to identify potential areas to improve the framework's approach.

$$pc_r = \frac{\sum_{i,j=1}^{n} r(x_{i_r}, x_{j_r})}{n} \tag{3.45}$$

$$pc_b = \frac{\sum_{i,j=1}^{n} r(x_{i_b}, x_{j_r})}{n} \tag{3.46}$$

3.5.8.1 Pearson Correlation Ratio Calculation

$$pc_i = \frac{pc_r}{pc_b} \tag{3.47}$$

The pc_i represents the ratio value of Pearson correlation for ransomware to ransomware and ransomware to benign files. The pc_i is calculated using the average Pearson correlation between ransomware and ransomware which is shown in Eq.3.45 and the calculation for the average Pearson correlation between ransomware and benign files is calculated in Eq.3.46.

3.5.9 Weighted Pearson Metric

The calculations for the weighted Pearson correlation are shown below in Eq.3.48 and 3.49.

$$pc_{b_i} = (pc_i \cdot w_b) \tag{3.48}$$

$$pc_{r_i} = (\frac{pc_i}{w_r}) \tag{3.49}$$

3.5.10 Drift Calculations for Alternative Similarity Metric

$$D_{r_i} = |hw_{r_i} - \bar{hw}_{r_c}| \tag{3.50}$$

$$D_{b_i} = |hw_{b_i} - \bar{hw}_{b_c}| \tag{3.51}$$

Equations 3.50 and 3.51 describe the drift calculation process using similarity metrics that may yield negative values. These formulas operate the same way as the drift calculations described in Eq.3.15 and 3.16; however, they use absolute values to measure the difference between the weighted similarity measures. The logarithm conversion is also only used on metrics that provide positive results. Logs are not utilised on metrics such as the cosine similarity and the Pearson correlation due to them producing negative values.

3.5.10.1 Prediction Probability Alternatives

The FeSAD framework can operate with machine learning algorithms that do not use a prediction probability metric as long as an alternative metric for prediction certainty can be derived. An example of an alternative to prediction probability would be the clusters and the distance from the centroid or the SVM algorithm's distance from the hyperplane.

3.6 Concluding Remarks

The FeSAD framework described in this chapter uses three different layers, which focus on helping a machine learning algorithm to be resilient and adaptive to concept drift. The FeSAD framework takes a unique approach to address concept drift in ransomware detection, as it focuses on building the machine learning system to be resilient to concept drift while using metrics from the algorithm and statistical metrics independent of the machine learning algorithm to judge the validity of classification. The main strength of the FeSAD framework is that it helps boost a classifier's ability to classify when concept drift occurs. The FeSAD framework can also help users identify when a machine learning classifier is degrading to where retraining is unavoidable. The FeSAD framework uses different elements to combat concept drift; however, it is essential to note that the elements used for the different layers are proven effective for concept drift detection and resilience to concept drift. The HEOM metric used by the FeSAD framework is modified to effectively identify concept drift when combined with the prediction probability taken from the algorithm. The drift metric allows the user to define what is inside and outside the scope of what can be acceptable drift and what can be unacceptable. The approach of using calibration sets to prepare a classifier for drift it will experience in the future is also highly beneficial; this is under the consideration that the concept drift displayed by ransomware is linear; therefore, the drift a classifier sees does not have to increase gradually as time passes, instead drift levels can fluctuate over time.

Chapter 4

Experimental Setup & Evaluation

This chapter presents the experimental results of FeSAD and the implications of these results; the FeSAD system was evaluated as a whole, and the FeSA layer is evaluated on its performance as a feature selection algorithm. This chapter also presents the experimental results obtained by testing various machine learning algorithms on different ransomware datasets which use different features and ransomware data. The experiments with machine learning algorithms were carried out as it was essential to identify the most efficient and reliable algorithms which would give the FeSAD system the best chance of performing well.

4.1 Data Procurement & Processing

This section covers the data used for the FeSAD framework and how it was procured and analysed. The data processing procedure involved developing a feature extraction program. FeSAD framework works with API call data taken from the Cuckoo Sandbox environment; this data is not produced in a format that is ready for use in a machine learning algorithm. The data taken from Cuckoo Sandbox needs to be processed and made available in a CSV or ARFF format for machine learning algorithms to process.

4.1.1 Sample Procurement

In order to be able to generate the data for experiments, ransomware and benign samples are needed to run through the sandbox to collect raw data generated from the analysis tasks. The Elderan dataset provides the hashes of 1582 ransomware and benign samples. The samples ranged from 2013 to 2015, so they were considered quite dated, especially benign ones. The first option for malware samples was Virus Total, one of the largest malware repositories online; however, to obtain specific samples, they required a paid subscription which was out of budget. Some samples are obtained from Hybrid Analysis, which was essential for acquiring new malware and benign samples; however, considering there were over 500 ransomware samples, hybrid analysis can only account for a small percentage of them. The bulk of the samples will be procured from VirusShare.com. While relatively simple, this process will be lengthy, with the benign files also being quite challenging to find, considering how dated some of them are. Considering there are no repositories like VirusShare, which hold everything in one place, the benign files needed to be taken from many freeware sites. The benign files added to the dataset were not all in the Elderan dataset, and it is more appropriate to select new benign files and benign files that behaved similarly to ransomware. Newer ransomware samples can be obtained from hybrid analysis, and these were selected due to their popularity and infection rates from the year they originated, ranging from 2016 to 2021. Table 4.1 shows the ransomware families included in the dataset.

4.1.2 Analysis Environment

Various malware analysis tools were evaluated for ransomware analysis; it was important for research to observe and understand a ransomware infection. The primary restraint when choosing a malware analysis platform was cost, as it had to be a free and effective analysis tool; this ruled out some of the most popular tools like CrowdStrike Falcon and Solar Winds. The malware analysis platform chosen was Cuckoo Sandbox, an open-source and industry leader in open-source malware analysis tools. Cuckoo Sandbox provided static, dynamic, and network features for samples; this was adequate to analyse the effects and impact of ransomware. Cuckoo Sandbox uses virtual machines to run malware to extract behavioural, static and network data. However, this would be inadequate as modern ransomware is equipped with anti-sandbox features that allow them to detect when they are being executed inside a sandbox or virtual machine; samples of this nature will not behave as they usually would in an infection scenario. The virtual machines used for all analysis tasks will be hardened by deploying the virtual machines using VMCloak and Paranoid Fish. VMCloak deploys hardened virtual machines with methods that combat anti-sandbox technology, while Paranoid Fish will detect aspects of the virtual machine which might expose it as a VM and allows the user to fix any issues. The virtual environments will not have their internet connections disabled; however, they will operate in a sandboxed manner which simulates a stable internet connection while stopping the samples from jumping out of the sandbox and into the network. The Cuckoo sandbox environment can run in Windows and Linux; the Cuckoo Sandbox instance will be deployed in a Linux environment with Windows virtual machines, using ransomware which can only be executed in a Windows environment. This measure is taken to prevent infection of the host in the unlikely event the ransomware escaped the Windows VMs it was being executed. The virtual machines the analysis tasks ran on will be clones and will be ensured to be identical to each other. The setup used by Cuckoo Sandbox is shown in figure 4.1.

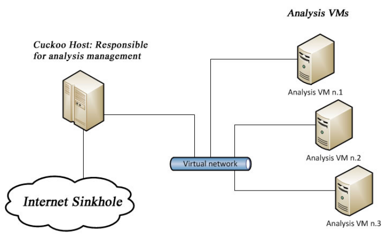

Figure 4.1: Cuckoo Sandbox Setup

4.1.3 Analysis Tasks

Analysis tasks are the next stage of the data procurement process; the samples needed to run through the Cuckoo sandbox and produce an analysis report. The analysis report would then need to be run through a feature extraction program. The analysis tasks should take roughly 5 minutes to complete but take up significant space on a hard drive, sometimes up to 2GB. The tasks can be run in batches; once the batch is complete, the analysis data will be moved onto an external hard drive. This process is simple as long as hard drive space is not an issue, but since each system is limited to a 200GB partition, Linux installation may require manual intervention. The sandbox analysis tasks run must be managed as each task is not guaranteed to be completed without issue. Two minutes is the default time setting for an analysis task set by Cuckoo; this may be repeated with a one-minute window if the framework needs to be refined and tuned for even earlier detection. The data is timestamped; therefore, it is possible to extract short data periods if required.

4.1.4 Datasets

We have used three datasets to evaluate the FeSAD framework; we produced one, and the other two are obtained from external sources.

4.1.4.1 FeSAD Ransomware Dataset

Our dataset contains API call and registry behaviour data taken from ransomware and benign files from 2013 to 2021; this dataset was created to capture the behavioural data for ransomware across nearly a decade. This dataset would capture ransomware and its evolution from its early stages; this is the perfect way to simulate concept drift and gives us an ideal way of testing the FeSAD framework. This dataset contains 720 ransomware samples and 2000 benign file samples. The data set is split with 460 ransomware samples from 2013-2015, 65 from 2016-17, 65 from 2018, 65 from 2019 and 65 from 2020-2021. There are 900 benign files from 2013-2015 and 1100 from 2016-2021, split into the same yearly batches as the ransomware files but with 275 benign files in each batch.

4.1.4.2 Elderan Dataset

This dataset contained raw data from cuckoo sandbox dumps similar to the data we produce and are produced for the research purposes of the Elderan system [20]. This dataset contained ransomware and benign files from 2012 to 2015. The raw data dump dataset contained API behaviour, registry interactions, file interactions and some file drop information. We split this dataset into data from 2012, 2013, 2014 and 2015 and then evaluated the FeSAD system. The Elderan system aimed to detect ransomware early; therefore, this dataset was an appropriate test for the FeSAD framework. The Elderan dataset contains 942 benign samples and 582 benign samples.

4.1.4.3 Navarra University Dataset

This dataset used network traffic based on a file-sharing system and I/O features for ransomware and benign files collected between 2015 and 2019. This dataset is conveniently set up for zero-day testing, with the testing set comprising exclusively of data the training set will not contain. The Navarra university contains network traffic data from 70 strains of ransomware and 2500 hours of network data from benign files. The Navarra university dataset is designed to evaluate ransomware detection systems and contains ransomware spanning four years, so it fits our purposes and tests the FeSAD framework with network data. The Navarra University dataset contains 3867 ransomware behavioural samples and 11159 benign behavioural in its training set and 114 ransomware behavioural samples and 14000 benign behavioural samples in it's zero-day test set.

4.1.5 Data Pre-Processing

Data pre-processing is needed after the raw data analysis files are produced using Cuckoo Sandbox. The feature extraction programs are written in Java and sweep the Cuckoo report JSON files looking for specific API-related features. The extraction program written for the EldeRan data initially only involved grouping a large number of features in a pre-existing dataset CSV; this will be particularly time-consuming because this program has to be written in multiple phases. The first phase involves sweeping a file with the feature list to find out which set of features existed in which index grouping of the CSV. Then using this information extracted from the feature list, the next phase is to write an extraction program which would group the initial 30,970 features into 12 features and then output this into a new CSV ready to be used on Machine Learning platforms like WEKA. The feature extraction programs for the subsequent experiments are created using Java. The features are pre-defined as the groups of hooked API calls for the experiments; the feature extraction program would be given two folders which contained Cuckoo analysis data files for the malign and benign data samples. The list contained 17 features, a grouping of over 300 API calls. The feature extractor counts the number of occurrences of the API calls from each group for each sample. The final features include hooked API calls, registry activity and network activity.

Table 4.1 Ransomware Families

Ransomware	Year of Variant Origin
PGPCoder	2012
Reveton	2013
Kollah	2013
CryptoLocker	2013
Locker	2013
Critroni	2014
Kovter	2014
Matsnu	2014
CryptoWall	2014
TeslaCrypt	2015
Trojan-Ransom	2015
Locky	2016
Petya	2016
Chimera	2016
BadRabbit	2017
Cerber	2017
WannaCry	2017
GandCrab	2018
Ryuk	2018
Scarab	2018
WannaCry	2019
Stop	2019
SamSam	2019
Virlock	2019
Rakhni	2019
Maze	2020
REvil	2020
Conti	2020
SunCrypt	2020
RagnarLocker	2020
Sodinokibi	2020

Dharma	2021
MedusaLocker	2021
Avaddon	2021

The ransomware in Table 4.1 comprises a dataset of 700 ransomware samples and 2000 benign samples. The ransomware from 2013 to 2015 was identified using the Elderan dataset, which compiled and published a ransomware dataset in January 2016. Most of the older ransomware samples pre-2016 were found on the virusshare repository; these samples were used in relatively large attacks for their time. The ransomware samples from the Elderan dataset represent an era where ransomware was not as mainstream and popular as it is now; ransomware was not used as much for large-scale attacks on businesses as they are today. Arguably, the first major ransomware attack that garnered global attention was the WannaCry ransomware which targeted healthcare and power infrastructure. The ransomware samples chosen post-2015 were included due to their involvement in prominent attacks on businesses and industry and their use in zero-day attacks. Using a wide range of popular ransomware gives the best chance of introducing concept drift to a machine-learning model, as it is highly likely that new ransomware with new functionality like data exfiltration and new propagation methods will present different behavioural patterns. We aimed to capture an even spread of 20 variations of each family; however, this was not possible due to the lack of availability or existence of diversity in some ransomware families. Due to availability issues, we have as evenly spread as possible between each ransomware family.

4.2 Analysis Tasks

The analysis tasks are set to two minutes per execution; the idea behind this is that this will provide enough behavioural data to determine whether a sample is ransomware before irreparable damage is done to a system or network. The median encryption time for ransomware is 42 minutes (Braue, 2022), with popular modern ransomware like Avaddon, Ryuk and REvil taking between 10 to 30 minutes to complete encryption. The two-minute window means that detection can be considered early and prevents the ransomware from encrypting all the critical files on the system and propagating and infecting multiple systems on a network. The two-minute window also provides enough information about a sample's behaviour to determine whether it is ransomware or benign. Cuckoo Sandbox captured a two-minute window of API calls and network behaviour that the FeSAD system can process to detect ransomware. Cuckoo Sandbox used Windows 7 virtual machines that are hardened with VMCloak and ParanoidFish to avoid the VMs being identified as an analysis environment by the ransomware. Modern ransomware is known to evade sandbox environments, so ensuring this would not prevent the extraction of accurate behavioural data was essential.

4.3 Machine Learning Experiments Phase 1

The first phase of machine learning experiments was used to determine the most effective algorithms for ransomware detection with behavioural data. The machine learning experiments used different feature sets and machine learning algorithms to identify the best combination of features and algorithms. The first feature set was derived using the EldeRan dataset; this dataset used the original dataset from the Elderan dataset. The raw Elderan dataset used 14,000 features; therefore, its dimensionality needed to be reduced to avoid accumulating useless features. The features were identified as categorical and separated into the following 12 categories; API Calls, File Drops, Registries Deleted, Registries Opened, Registries Read, Registries Written, Files Deleted, Files Written, Files Opened, Files Read, Directories Created, Directories Enumerated, and class. Based on the reviewed literature, it was simple to identify the most commonly used machine learning algorithms for ransomware detection; these algorithms were used in the first phase of experiments. The feature sets resistance to concept drift is also tested; this is done by training on data from 2012 to 2014 and testing on data from 2015. The configurations for the machine learning algorithms used are presented in table 4.2, the formulas used to measure performance metrics are presented in table 4.3, and the results for the experiments are presented in table 4.4. The confidence intervals for the FeSA and FeSAD experiments are calculated using a Z-score of 1.96 with a confidence level of 95%. The confidence level of 95 is appropriate for a detection system as it provides a high confidence level in the predictions and is suitable for our smaller test sets when testing for concept drift adaptability.

Table 4.2 ML Configurations

Algorithm	Configuration
GTB (Gradient Tree Boosting)	· **ZMax Value:** 3.0 · **Classifier:** Random Tree · **Likelihood Threshold:** 1.7976931 · **Weight Threshold:** 100
Random Forest	· **Bag Size Percent:** 100 · **Max Tree Depth:** No Max Defined · **Iterations:** 100
SVM	· **SVM Type:** C-SVC · **Eps:** 0.001 · **Gamma:** 0 · **Kernel Type:** Radial Basis Function: Exp($-gamma\ \|u \dot{-} v\|^2$) · **Loss:** 0.1 · **Nu:** 0.1
Logistic Regression	· **Max iterations:** -1 · **Ridge:** Ridge: 1.08 x 10^8
J48 Decision Tree	· **Confidence Factor:** 0.25 · **MinNumObj:** 2 · **Folds:** 3
Deep Neural Network (10 fold Cross Validation)	· **Instance Iterator:** Default · **Neural Network configuration - Leaky RELU(Rectified Linear Unit) alpha:** 0.01 · **Optimisation:** Stochastic Gradient Descent · **Updater:** Adam · **Beta1MeanDecay:** 0.9 · **Epsilon:** 1.08 x 10^8 · **Learning Rate:** 0.1 · **Weight Initialisation Vector:** XAVIER · **Gradient Normalisation threshold:** 1.0 · **Intermediate evaluation:** 5

MLP (5 Hidden Layers)	· **Hidden Layers:** 5 · **Learning Rate:** 0.3 · **Momentum:** 0.2 · **Validation Threshold:** 20
MLP (10 Hidden Layers)	· **Hidden Layers:** 10 · **Learning Rate:** 0.3 · **Momentum:** 0.2 · **Validation Threshold:** 20
Bayesian Network	· **Estimator: Simple Estimator, alpha** : 0.5 · **Search Algorithm:** Simulated Annealing

Table 4.3 Phase 1 Experiments

Method	Old Gen Detection Rate (2012-2014)	New Gen Detection Rate (2015)
Logistic Regression	**Detection Rate:** 79.6%(75.7%, 80.8%) **False Positive Rate:** 19.0%(12.7, 25.3%) **Recall:** 0.796(0.756, 0.836) **Precision:** 0.81(0.750, 0.870)3	**Detection Rate:** 70.1%(64.1%, 76.1%) **False Positive Rate:** 25.3%(19%, 31.6%) **Recall:** 0.701(0.641, 0.761) **Precision:** 0.730(0.670, 0.790)
SVM	**Detection Rate:** 76.6%(72.5%, 80.7%) **False Positive Rate:** 16.0%(12.4%, 19.6%) **Recall:** 0.766(0.726, 0.806) **Precision:** 0.830(0.790, 0.870)	**Detection Rate:** 58.7%(51.5%, 65.9%) **False Positive Rate:** 31.5%(24.8%, 38.2%) **Recall:** 0.587(0.515, 0.659) **Precision:** 0.625(0.575, 0.695)
J48	**Detection Rate:** 85.2%(79.0%, 88.7%) **False Positive Rate:** 12.3%(9.1%, 15.5%) **Recall:** 0.852(0.815, 0.892) **Precision:** 0.874(0.844, 0.904)	**Detection Rate:** 83.2%(77.8%, 88.6%) **False Positive Rate:** 15.6%(10.4%, 20.8%) **Recall:** 0.832(0.782, 0.882) **Precision:** 0.842(0.792, 0.892)
Bayesian Networks	**Detection Rate:** 83.2%(79.5%, 86.9%) **False Positive Rate:** 18.5%(14.7%, 22.3%) **Recall:** 0.832(0.792, 0.872) **Precision:** 0.842(0.802, 0.882)	**Detection Rate:** 76.1%(69.6%, 82.6%) **False Positive Rate:** 26.4%(20%, 32.8%) **Recall:** 0.761(0.691, 0.831) **Precision:** 0.742(0.692, 0.792)
GTB	**Detection Rate:** 80.4%(76.5%, 84.3%) **False Positive Rate:** 15.4%(11.9%, 18.9%) **Recall:** 0.804(0.764, 0.844) **Precision:** 0.842(0.792, 0.892)	**Detection Rate:** 75.4%(69.1%, 81.7%) **False Positive Rate:** 22.5%(16.4%, 28.6%) **Recall:** 0.832(0.782, 0.882) **Precision:** 0.780(0.720, 0.840)
MLP(5 Hidden Layers)	**Detection Rate:** 83.3%(79.6%, 87.0%) **False Positive Rate:** 16.6%(13.0%, 20.2%) **Recall:** 0.833(0.793, 0.873) **Precision:** 0.834(0.794, 0.874)	**Detection Rate:** 81.4%(75.7%, 87.1%) **False Positive Rate:** 18.0%(12.4%, 23.6%) **Recall:** 0.833(0.783, 0.883) **Precision:** 0.819(0.759, 0.879)
Random Forest	**Detection Rate:** 87.6%(84.4%, 90.8%) **False Positive Rate:** 11.0%(8.0%, 14.0%) **Recall:** 0.876(0.846, 0.906) **Precision:** 0.89(0.860, 0.920)	**Detection Rate:** 84.3%(79.0%, 89.6%) **False Positive Rate:** 15.2%(10.0%, 20.4%) **Recall:** 0.843(0.793, 0.893) **Precision:** 0.847(0.797, 0.897)
Deep Learning	**Detection Rate:** 83.0%(76.3%, 86.7%) **False Positive Rate:** 18.7%(14.9%, 22.5%) **Recall:** 0.830(0.790, 0.870) **Precision:** 0.816(0.776, 0.856)	**Detection Rate:** 75.8%(69.6%, 82.0%) **False Positive Rate:** 25.8%(19.5%, 32.1%) **Recall:** 0.758(0.698, 0.816) **Precision:** 0.746(0.686, 0.806)
MLP(10 Hidden Layers)	**Detection Rate:** 83.0%(79.3%, 86.7%) **False Positive Rate:** 16.6%(13.3%, 19.9%) **Recall:** 0.830(0.790, 0.870) **Precision:** 0.830(0.790, 0.870)	**Detection Rate:** 79.7%(73.9%, 85.5%) **False Positive Rate:** 19.1%(13.4%, 24.8%) **Recall:** 0.797(0.737, 0.857) **Precision:** 0.807(0.747, 0.867)

The results in Table 4.4 are obtained using a 10-fold cross-validation method. Cross-validation was chosen instead of a percentage split because it covers the entire dataset and gives us an idea of how the system performs on all data as if it were training data. Table 4.4 shows the test results using the raw data collected for the EldeRan system. Exposing each classifier to zero-day threats tests the ability of the classifier to detect threats it has not been trained on. When evaluated with cross-validation, the average detection rate achieved across all classifiers is 82.4%, and the average false positive rate is 16.1%. The average detection rate achieved when exposed to zero-day threats is 76.1%, with an average false positive rate of 22.2%. The results in Table 4.4 show that none of the algorithms achieved a detection rate above 90% and would misclassify a higher percentage of ransomware than what is acceptable; this issue would only worsen over time as concept drift sets in and ransomware changes. We can easily identify the better-performing algorithms by evaluating which has detection statistics above the average and a false positive rate below the average. The results show that the Random Forest, Multi-Layer Perceptron, and J48 decision tree show the most robust detection results while showing the least degradation when exposed to zero-day ransomware. The random forest depth could be edited as it currently uses an unlimited depth setting which does not necessarily equal the best results in terms of false positives. The Random Forest and Bayesian network are relatively lightweight to run and train compared to the Multilayer Perceptron, and it will need a significant performance advantage over the alternatives to be used. The Multilayer Perceptron's main issue is that it requires more than one or two hidden layers to be considered a viable option; we observe that it needs at least five to limit large numbers of false positives. The Elderan data provided a good idea of what algorithms could provide strong results; however, it could not be built on because it is impossible to recreate the testbed used by Sgandurra et al. The testbed needs to be identical for all the experiments for all results to be considered accurate. The Elderan data contained data from 2012 to 2015, which leaves up to seven years of ransomware data to add. The settings for the three best performance algorithms could be tuned to further increase performance in the next phase of experiments.

4.3.1 Machine Learning Experiments Phase 2

The second phase of machine learning experiments involved rebuilding the dataset in a way that concept drift could be accurately tested and in a way, the dataset could be built on and expanded. Multiple research papers have been reviewed using API calls to detect ransomware, which has yielded promising results. API calls give an insight into how ransomware interacts with a system in the earliest moments of execution; therefore, it is a logical choice for features for ransomware detection. Network features also give an idea of how ransomware behaves regarding the network; ransomware commonly makes network communications with a command and control server, so it makes sense to attempt to use network communications to identify the patterns for these communications. The final features used for the second phase of experiments included hooked API calls, registry activity, and UDP and TCP connections. New data taken from new Cuckoo Sandbox analysis tasks were used in the second phase of experiments. The results of these experiments are shown in Table 4.5, and the configurations for the machine learning algorithms used are the same as the experiments in Table 4.4. The final feature count for this dataset was 327; this was considered inefficient for a classifier, so the Genetic Search feature selection algorithm was used in these experiments to reduce the dimensionality of the data. The ransomware and benign data used in these experiments ranged from 2012 to 2017.

Table 4.4 API-Call Dataset Experiments

Method	**Detection Rate on 13-15 Samples**	**Detection on 16/17 Samples**	**13-15 False Positive**	**16/17 False Positive**	**13-15 Precision**	**16/17 Precision**
Regularised Logistic Regression	88.6%(86.0%, 91.2%)	62.3%(49.2%, 75.4%)	13.6%(10.8%, 16.4%)	32.1%(19.5%, 44.7%)%	0.864(0.834, 0.894)	0.66(0.53, 0.79)
Gradient Tree Boosting	94.6%(92.5%, 96.7%)	77.4%(66.1%, 88.7%)	5.5%(3.8%, 7.3%)	19.2%(8.6%, 29.8%)	0.96(0.94, 0.98)	0.80(0.69, 0.91)
Random Forests	94.9%(93.1%, 96.7%)	79.2(68.4%, 90.0%)	5.6%(3.8%, 7.4%)	17.6%(7.3%, 27.9%)	0.95(0.93, 0.97)	0.82(0.72, 0.92)
SVM	33.1%(29.3%, 36.9%)	17.0%(7.0%, 27.0%)	36.1%(32.3%, 40.0%)	70.4%(58.2%, 82.6%)	0.48(0.13, 0.82)	0.19(0.08, 0.30)
J48 Decision Tree	89.8%(87.3%, 92.3%)	79.2%(68.3%, 90.1%)	8.0%(5.8%, 10.2%)	17.9%(7.6%, 28.2%)	0.92(0.90, 0.94)	0.83(0.73, 0.93)
Deep Neural Network Using 10 (Fold Cross Validations)	95.1%(93.3%, 96.9%)	84.9%(75.3%, 94.5%)	9.6%(7.3%, 11.9%)	13.6%(4.6%, 22.6%)	0.91(0.89, 0.93)	0.86(0.77, 0.95)
Multi-Layer Perceptron (5 Hidden Layer)	96.4%(94.9%, 97.9%)	85.0%(75.4%, 94.6%)	8.0%(5.8%, 10.2%)	14.0%(4.7%, 23.3%)	0.92(0.90, 0.94)	0.86(0.77, 0.95)
Multi-Layer Perceptron (10 Hidden Layers)	95.8%(95.3%, 96.0%)	58.5%(45.3%, 61.7%)	9.0%(6.8%, 11.3%)	35.5%(22.7%, 58.3%)	0.91(0.89, 0.93)	0.62(0.48, 0.62)
Bayesian Networks	93.4%(91.4%, 95.4%)	83.0%(73.0%, 93.0%)	13.3%(10.5%, 16.1%)	17.4%(7.2%, 27.6%)	0.87(0.84, 0.91)	0.83(0.73, 0.93)

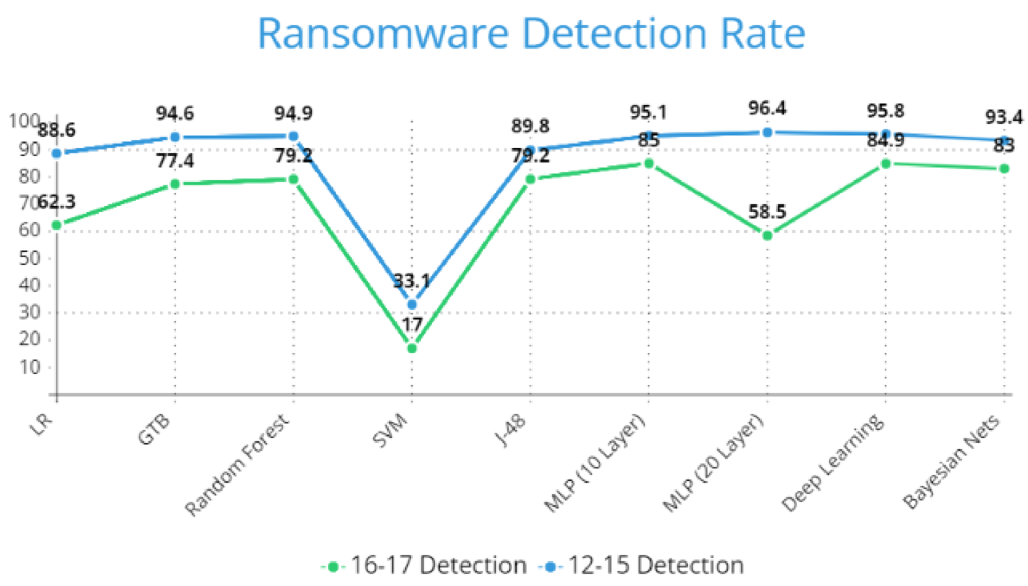

Figure 4.2: API-Call Dataset Ransomware Detection Rates

The results in Table 4.5 show that the API call and network activity features can help classify ransomware as all but two of the classifiers achieved detection rates above 90% for ransomware from the same distribution as them. Using 10-fold cross-validation evaluates the classifier's overall performance; the classifier that achieves the best detection rate is the Multi-Layer Perceptron with five hidden layers, followed by the Weka Deep Learning library and the Random Forest. The random forest has a strong detection rate; however, despite not having the highest detection rate, it has the lowest false positive rate by a significant margin. The Multi-Layer Perceptron and Deep-Learning approaches see the lowest degradation in performance when exposed to concept drift, as shown in Figure 4.2.

The concept drift in these experiments is represented by testing ransomware and benign samples from 2016-17 when the model is trained on data from 2012-2015. The algorithms that perform at the highest level in the 2nd phase of testing are the Multi-Layer Perceptron, Deep Neural Network and the Random Forest. The feature set appears to be effective with most of the algorithms that have been used; the only algorithm that appears to be completely unsuitable is the Support Vector Machine, as it appears unable to distinguish ransomware from benign files but is relatively strong when classifying benign files. The false positive rate tends to rise under concept drift, as shown in Figure 4.3. In terms of statistical significance, the FeSA algorithm shows a statistically significant detection rate advantage over Best First and Greedy Stepwise in the 13-17 batch, and Best First, Genetic Search and Greedy stepwise in the 13-18 batch.

This phase of experiments has shown that API calls, registry activity and TCP and UDP behaviour can be used with machine learning algorithms to detect ransomware infections. The detection period between one to two minutes also provides early detection that limits the damage to a system and can help prevent propagation through a network. Using a feature selection algorithm and specialised features known to be effective in ransomware detection has provided robust results, especially if comparing the results to the machine learning detection results in the phase 1 experiments, as shown in Figure 4.4. The SVM is the only algorithm that appears stronger

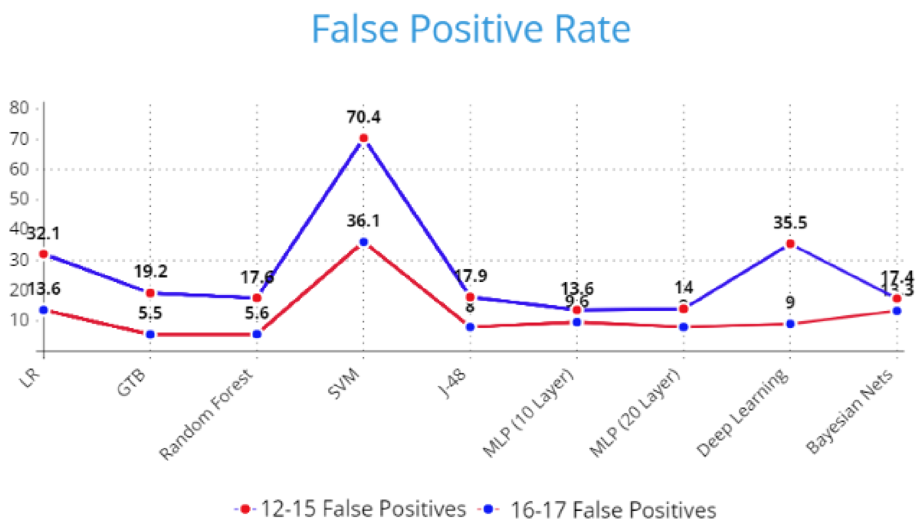

Figure 4.3: API-Call Dataset False Positive Rates

with the phase 1 feature set.

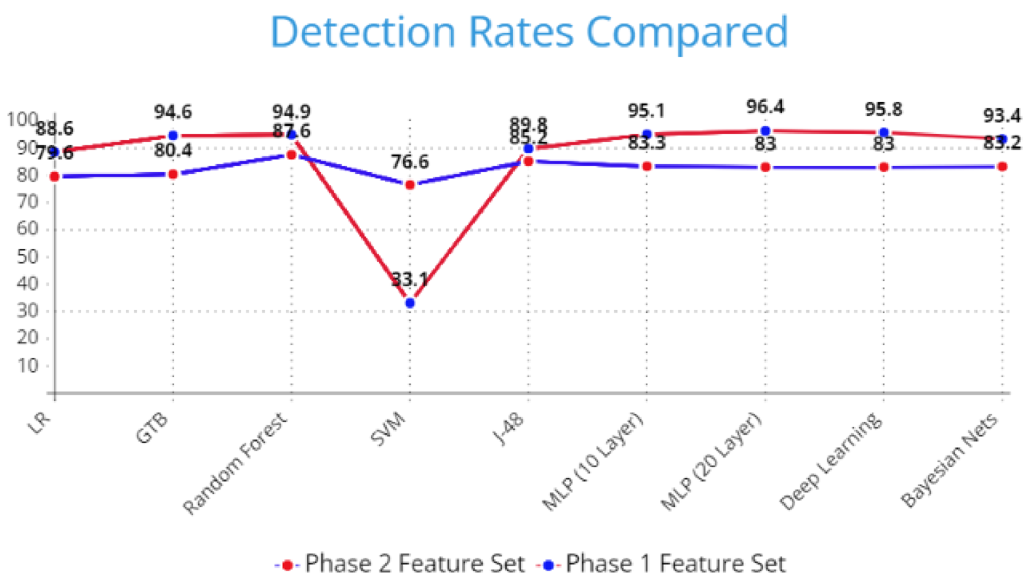

Figure 4.4: API-Call Dataset Detection Rates Compared

Figures 4.2 and 4.3 show that concept drift degrades the performance of the classifiers in critical areas such as detection and false positives. None of the classifiers maintained their detection rates when exposed to concept drift, which shows that ransomware develops over time. The concept drift in ransomware exposes potential flaws in detection systems and that ransomware can interact slightly differently with the OS and still fulfil its purpose. The initial high performance of the algorithms combined with their susceptibility to concept drift showed that it is essential to consider concept drift when using a machine learning classifier for ransomware detection. The concept drift degradation shown in the phase 2 feature set is significantly less than the feature set from phase

1; this means the feature set is appropriate for systems that deal with concept drift.

4.4 FeSA Experiments

This stage of experiments builds on the experimental results obtained during the two stages of testing machine learning algorithms. The highest-performing algorithms were the neural network, deep learning approaches, and the Random Forest. The FeSA Layer builds the feature set used by the FeSAD framework; therefore, it is essential to test its viability as a feature selection algorithm and its ability to create feature sets that can provide a layer of protection against concept drift. The results of these experiments are shown in Table 4.6; the FeSA layer is tested against other popular feature selection algorithms in terms of detection on samples from the same distribution. The FeSA layer is also tested on its ability to produce feature sets that can remain robust when concept drift occurs; this is also compared to other feature selection algorithms. The results of the concept drift experiments are presented in Table 4.6.

Table 4.5 Experimental Results without Concept Drift

	FeSA	**Best First**	**Evolutionary Search**	**Genetic Search**	**Greedy Stepwise**	**Harmony Search**
Time Complexity	$O(n) + O(gnm))$	$O(n \cdot Log(n))$	$O(gnm))$	$O(gnm))$	$O(n)^2$	$O(n)$
Feature Count	**32**	**20**	**110**	**106**	**20**	**33**
Trained and tested on 13-15	Detection: 96.1%(93.9%, 97.6%) FPR: 5.5%(3.1%, 8.8%) Precision: 0.942(0.924, 0.954) Recall: 0.961(0.939, 0.976)	Detection: 92.0%(89.5%, 94.5) FPR: 6.7%(4.9%, 9.4%) Precision: 0.940(0.920, 0.960) Recall: 0.920(0.902, 0.935)	Detection: 95.7%(93.5%, 97.2%) FPR: 4.4%(2.5%, 5.9%) Precision: 0.955(0.940, 0.967) Recall: 0.957(0.935, 0.972)	Detection: 93.2%(90.5%, 95.1%) FPR: 6.2%(4.3%, 8.6%) Precision: 0.942(0.926, 0.955) Recall: 0.932(0.905, 0.951)	Detection: 90.4%(87.4%, 92.7%) FPR: 4.4%(2.4%, 5.9%) Precision: 0.963(0.951, 0.974) Recall: 0.904(0.874, 0.927)	Detection: 93.0%(90.3%, 95.0%) FPR: 5.9%(4.5%, 7.3%) Precision: 0.944(0.925, 0.958) Recall: 0.930(0.903, 0.950)
Trained and tested on 13-17	Detection: 96.7%(94.9%, 98.1%) FPR: 5.0%(3.9%, 7.3%) Precision: 0.942(0.904, 0.938) Recall: 0.967(0.949, 0.981)	Detection: 90.1%(87.2%, 92.6%) FPR: 8.0%(5.9%, 10.8%) Precision: 0.918(0.900, 0.935) Recall: 0.901(0.872, 0.926)	Detection: 95.3%(93.0%, 97.0%) FPR: 5.2%(3.8%, 7.3%)% Precision: 0.948(0.933, 0.961) Recall: 0.953(0.930, 0.970)	Detection: 95.3%(93.0%, 97.0%) FPR: 5.2%(3.8%, 7.3%) Precision: 0.948(0.933, 0.961) Recall: 0.953(0.930, 0.970)	Detection: 90.1%(87.5%, 92.7%) FPR: 8.0%(5.9%, 10.8%) Precision: 0.931(0.914, 0.945) Recall: 0.901(0.8735, 0.927)	Detection: 93.0%(90.3%, 95.0%) FPR: 5.9%(4.7%, 7.1%) Precision: 0.944(0.928, 0.957) Recall: 0.930(0.903, 0.950)
Trained and tested on 13-18	Detection: 96.3%(94.2%, 97.7%) FPR: 5.2%(3.8%, 7.3%) Precision: 0.941(0.925, 0.958) Recall: 0.963(0.942, 0.977)	Detection: 91.5%(88.8%, 93.8%) FPR: 7.0%(5.0%, 9.4%)% Precision: 0.935(0.917, 0.949) Recall: 0.915(0.888, 0.938)	Detection: 95.7%(93.5%, 97.2%) FPR: 5.4%(3.7%, 7.7%) Precision: 0.946(0.931, 0.959) Recall: 0.957(0.935, 0.972)	Detection: 91.5%(88.8%, 93.8%) FPR: 6.5%(4.6%, 9.0%) Precision: 0.927(0.909, 0.942) Recall: 0.915(0.888, 0.935)	Detection: 91.5%(88.8%, 93.8%) FPR: 7.7%(5.5%, 10.3%) Precision: 0.927(0.905, 0.939) Recall: 0.915(0.888, 0.938)	Detection: 95.0%(92.8%, 96.8%) FPR: 5.5%(3.7%, 7.7%) Precision: 0.945(0.929, 0.958) Recall: 0.950(0.928, 0.968)

The feature set generated by the FeSA layer maintains a detection rate above 96% when tested on the same distribution as the training was carried out; this outperforms the genetic and evolutionary search algorithms. Outperforming, the genetic and evolutionary search algorithms were necessary as the FeSA Layer is a modified genetic algorithm. The FeSA layer uses 32 features, the least amount of feature selection algorithms we have tested against. We chose 32 after experimenting with different configurations as the API calls were broken into 16 categories; therefore, we aim to capture at least 2 features from each category. It is important to note that the FeSA layer allows the user to set the feature number. The 32 layers selected by FeSA can outperform the Evolutionary algorithm and Genetic algorithm, which both select over 100 features; these feature selection algorithms do not have the advantage of the feature ranker. The FeSA algorithm feature set consistently shows the highest detection rates across three distributions it is tested on and offers a competitive false-positive rate despite not being the lowest. The Evolutionary Search algorithm achieves an average false-positive rate of 5%, whereas FeSA achieves an average of 5.2%. In the context of statistical significance, for 13-15 data, FeSA performance is close to the other algorithms, with the difference in FeSA, Best First, Genetic Search, Evolutionary Search, and Harmony Search being statistically insignificant. For 13-17, the difference between FeSA and Evolutionary Search, Genetic Search and Harmony Search is statistically insignificant. For 13-18 data, the difference between FeSA and Evolutionary Search, Genetic Search, and Harmony Search can be considered statistically insignificant. The FeSA Layer works with an underlying machine-learning algorithm, and the algorithm chosen for these experiments was the Random Forest. The Random Forest consistently achieves strong detection results while not being as computationally taxing as the neural network approaches that have also achieved strong results. The FeSA layer relies on the detection and accuracy metrics of the underlying algorithm as a fitness function. The algorithms the FeSA Layer is compared to do not perform comparatively well when using an underlying machine learning algorithm as a fitness function; therefore, their default fitness functions are used. The FeSA Layer's advantage when using an underlying machine learning algorithm is that the features are guaranteed to work well in the corresponding classification algorithm used in the Drift Calibration and Drift Decision Layers. It has been observed that the underlying machine learning algorithm used to generate the feature sets works best when using the feature set to classify samples. The features generated in the FeSA Layer also provide a low false positive rate compared to other feature selection algorithms. When compared with the Genetic Search and Evolutionary Search, the FeSA Layer needs two generations to reach its optimal solution instead of the former algorithms requiring 20 generations to reach its optimal solution.

Table 4.6 Experimental Results with EldeRan Dataset

	FeSA	**Best First**	**Evolutionary Search**	**Genetic Search**	**Greedy Stepwise**	**Harmony Search**
Feature Count	**32**	**258**	**160**	**106**	**14**	**36**
Trained and tested on 12-13	Detection: 99.5%(97.2%, 99.9%) FPR: 5.1%(3.1%, 7.8%) Precision: 0.952(0.931, 0.967) Recall: 0.995(0.972, 0.999)	Detection: 98.0%(95.0%, 99.4%) FPR: 7.0%(4.7%, 9.9%) Precision: 0.933(0.907, 0.949) Recall: 0.980(0.950, 0.994)	Detection: 95.5%(91.6%, 97.9%) FPR: 9.7%(7.0%, 13.1%) Precision: 0.920(0.895, 0.940) Recall: 0.955(0.916, 0.979)	Detection: 92.3%(87.3%, 95.3%) FPR: 5.6%(3.5%, 8.2%) Precision: 0.955(0.945, 0.970) Recall: 0.923(0.873, 0.953)	Detection : 98.0%(94.9%, 99.4%) FPR: 6.9%(4.7%, 9.9%) Precision: 0.934(0.910, 0.951 Recall: 0.980(0.949, 0.994)	Detection: 81.0%(74.8%, 86.2%) FPR: 12.7%(8.9%, 15.5%)% Precision: 0.904(0.873, 0.923) Recall: 0.810(0.748, 0.862)
Trained and tested on 2014	Detection: 97.5%(94.2%, 99.1%) FPR: 8.4%(5.5%, 11.1%) Precision: 0.918(0.895, 0.940) Recall: 0.975(0.942, 0.991)	Detection : 83.9%(78.1%, 88.8%) FPR: 13.2%(9.9%, 16.7%) Precision: 0.871(0.840, 0.896) Recall: 0.839(0.781, 0.888)	Detection: 92.4%(87.3%, 95.3%) FPR: 6.2%(3.9%, 8.8%) Precision: 0.939(0.907, 0.949) Recall: 0.924(0.873, 0.953)	Detection: 89.0%(83.8%, 93.0%) FPR: 9.5%(6.8%, 12.8%) Precision: 0.905(0.879, 0.928) Recall: 0.890(0.838, 0.930)	Detection: 84.5%(78.7%, 89.2%)% FPR: 13.5%(9.9%, 16.9%)% Precision: 0.866(0.840, 0.895) Recall: 0.845(0.787, 0.892)	Detection: 95.5%(91.6%, 97.9%) FPR: 11.1%(8.1%, 14.4%) Precision: 0.893(0.862, 0.914) Recall: 0.955(0.916, 0.979)
Trained and tested on 2015	Detection: 97.1%(93.6%, 98.9%) FPR: 12.8%(9.6%, 16.4%) Precision: 0.884(0.851, 0.905) Recall: 0.971(0.936, 0.989)	Detection: 94.0%(89.8%, 96.9%) FPR: 16.2%(12.5%, 19.9%) Precision: 0.853(0.819, 0.878) Recall: 0.940(0.898, 0.969)	Detection: 96.0%(92.3%, 98.3%) FPR: 9.50%(6.80%, 12.81%) Precision: 0.910(0.884, 0.932) Recall: 0.960(0.923, 0.983)	Detection: 95.5%(91.6%, 97.9%) FPR: 13.6%(10.8%, 18.0%) Precision: 0.898(0.848, 0.948) Recall: 0.955(0.916, 0.979)	Detection: 94.7%(92.1%, 97.3%) FPR: 16.1%(12.5%, 19.7%) Precision: 0.856(0.826, 0.884) Recall: 0.947(0.921, 0.973)	Detection: 94.0%(89.8%, 96.9%) FPR: 14.4%(11.1%, 18.3%) Precision: 0.867(0.836, 0.893 Recall: 0.940(0.898, 0.969)

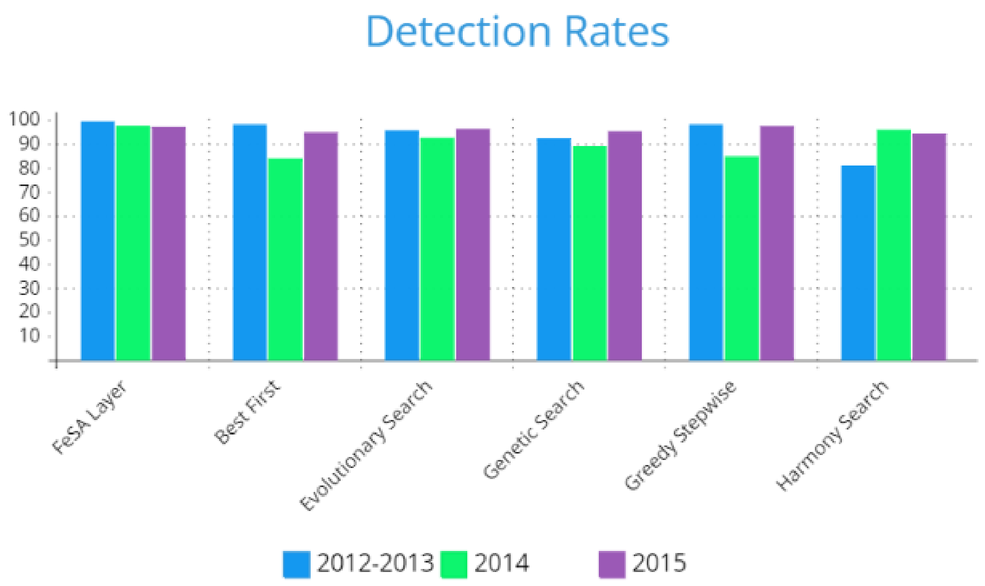

Figure 4.5: EldeRan Dataset Ransomware Detection Rates

Table 4.6 shows the FeSA algorithms performance on the Elderan dataset; in the 12-13 data, there is no statistical significance in the difference in performance between FeSA and the competing algorithms; The 2015 data shows no statistical difference between the algorithms and FeSA. The FeSA Layer identifying 32 significant features out of over 14,000 shows its ability as a general feature selection algorithm that can yield feature sets for effective ransomware detection. Figure 4.5 demonstrates the FeSA Layer's performance against other feature selection algorithms in terms of detection rate. Figure 4.6 shows the FeSA Layer's performance against other feature selection algorithms in terms of false positives.

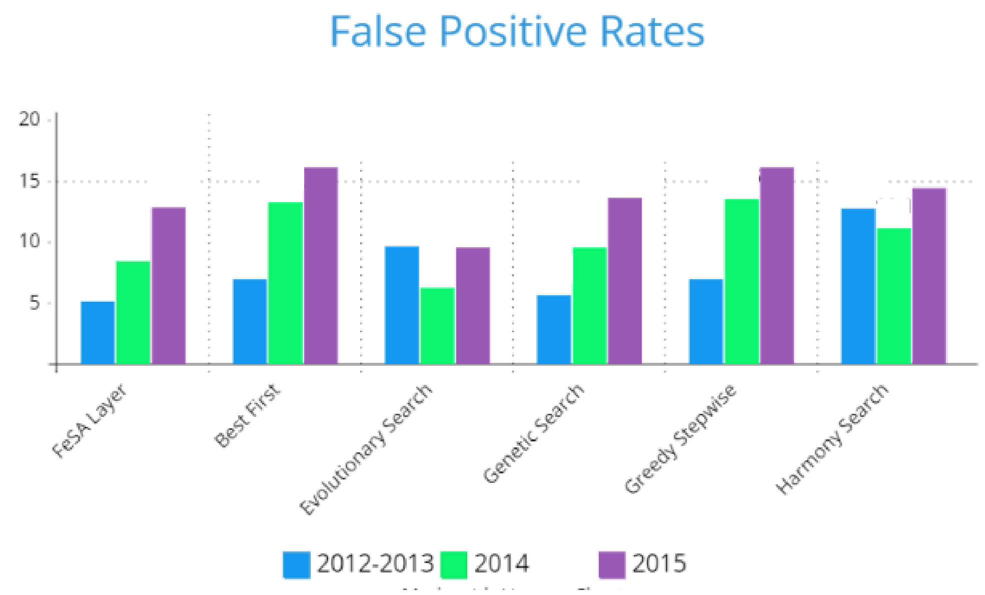

Figure 4.6: EldeRan Dataset False-Positive Rates

Table 4.7 Experimental Results under Concept Drift

	FeSA	**Best First**	**Evolutionary Search**	**Genetic Search**	**Greedy Stepwise**	**Harmony Search**
Feature Count	**32**	**20**	**110**	**106**	**20**	**33**
13-15 Tested on 2016/17	Detection: 92.3%(82.9%, 97.4%) FPR: 7.3%(4.5%, 11.0%) Precision: 0.926(0.893, 0.951) Recall: 0.923(0.829, 0.974)	Detection: 78.4%(66.5%, 87.7%) FPR: 2.7%(1.5%, 6.0%) Precision: 0.966(0.939, 0.982) Recall: 0.784(0.665, 0.877)	Detection: 86.1%(75.3%, 93.4%) FPR: 1.4%(0.3%, 3.7%) Precision: 0.968(0.942, 0.984) Recall: 0.861(0.753, 0.934)	Detection: 73.8%(61.4%, 84.0%) FPR: 4.4%(2.0%, 7.2%) Precision: 0.943(0.910, 0.963) Recall: 0.738(0.614, 0.840)	Detection: 78.4%(66.5%, 87.6%) FPR: 2.4%(1.0%, 5.0%) Precision: 0.970(0.943, 0.984) Recall: 0.784(0.665, 0.876)	Detection: 83.0%(71.7%, 91.2%) FPR: 1.4%(0.3%, 3.0%) Precision: 0.961(0.934, 0.979) Recall: 0.830(0.717, 0.912)
13-17 Tested on 2018 Data	Detection: 100%(94.4%, 100%) FPR: 3.4%(1.5%, 6.0%) Precision: 0.967(0.942, 0.984) Recall: 1.0(0.944, 1.0)	Detection: 98.4%(91.7%, 99.9%) FPR: 2.22%(1.0%, 5.2%) Precision: 0.979(0.958, 0.999) Recall: 0.984(0.917, 0.999)	Detection: 100%(94.4%, 100%) FPR: 5.5%(3%, 8.8%) Precision: 0.943(0.914, 0.966) Recall: 1.0(0.944, 1.0)	Detection: 100%(94.4%, 100%) FPR: 1.4%(0.3%, 3.0%) Precision: 0.989(0.974, 0.998) Recall: 1.0(0.944, 1.0)	Detection: 100%(94.4%, 100%) FPR: 3.4%(1.5%, 6.0%) Precision: 0.967(0.942, 0.984) Recall: 1.0(0.944, 1.0)	Detection: 53.8%(41.0%, 66.3%) FPR: 1.4%(0.3%, 3.7%) Precision: 0.922(0.889, 0.949) Recall: 0.540(0.410, 0.663)
13-18 Tested on 2019 Data	Detection: 93.8%(85.0%, 98.3%) FPR: 3.4%(1.5%, 6.1%) Precision: 0.965(0.939, 0.982) Recall: 0.938(0.850, 0.983)	Detection: 87.6%(77.2%, 94.5%) FPR: 11.9%(8.4%, 16.4%) Precision: 0.880(0.843, 0.915) Recall: 0.871(0.772, 0.945)	Detection: 90.7%(80.9%, 96.5%) FPR: 1.4%(1.2%, 5.7%) Precision: 0.979(0.958, 0.992) Recall: 0.907(0.809, 0.965)	Detection: 87.6%(77.2%, 94.5%) FPR: 2.9%(1.2%, 5.7%) Precision: 0.968(0.943, 0.984) Recall: 0.876(0.772, 0.945)	Detection: 87.6%(77.2%, 94.5%) FPR: 3.1%(1.2%, 5.7%) Precision: 0.963(0.943, 0.984) Recall: 0.876(0.772, 0.945)	Detection: 83.0%(71.7%, 91.2%) FPR: 1.7%(0.5%, 4.2%) Precision: 0.969(0.943, 0.984) Recall: 0.830(0.717, 0.912)

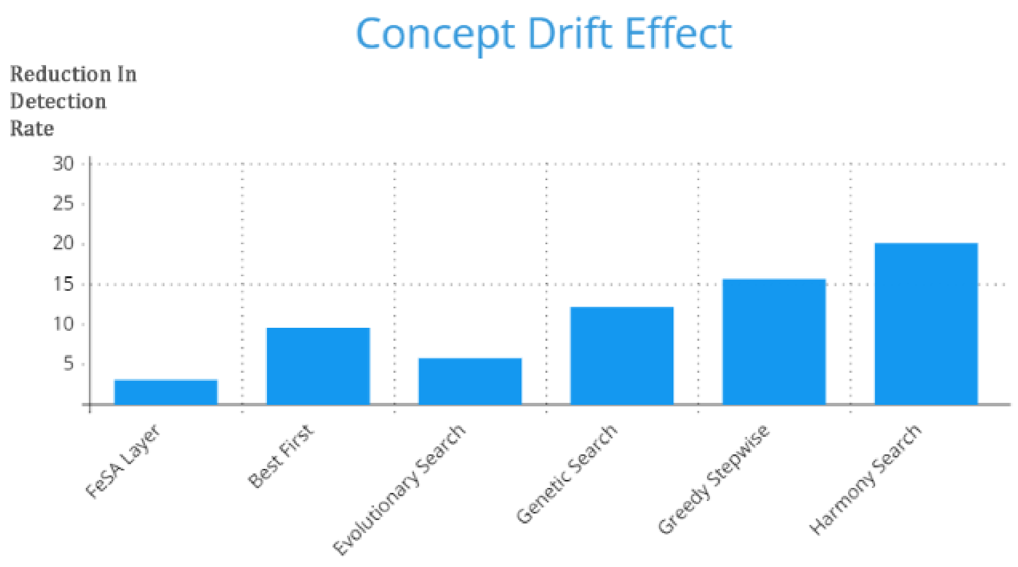

Figure 4.7: EldeRan Dataset Detection Reduction

The robustness of the feature sets generated in the FeSA Layer is tested in concept drift scenarios; concept drift is introduced by testing on prominent ransomware families from a time period after the ransomware from the training data; the results of these experiments are shown in Table 4.7. The structure of the experiments would mean training on data from 2012 to 2015 and testing on data from 2016 or 2017 and onwards from these time periods. As shown in Figure 4.7, the average reduction in detection for the FeSA system is 3%, which is lower than the reduction in detection rate for the feature sets produced by the compared feature selection algorithms. The genetic and evolutionary search algorithms produce the nearest comparable feature sets, which supports the initial assumption that a genetic algorithm can produce feature sets effective in concept drift situations. The experimental structure follows the most realistic real-world structure where the zero-day ransomware is ransomware from a time period after the training set data. The FeSA Layer is proven to produce feature sets that show low degradation under concept drift; however, false positives can be improved under concept drift. The FeSA Layer's ability to produce high-quality feature sets effective for ransomware detection is shown through this phase of experiments; however, it was necessary to ensure its effectiveness on an external dataset besides the EldeRan dataset. There is little statistical significance in the difference in FeSA performance from the other algorithms in the 2018 test; in the 2016/17 and 2019 tests, the FeSA algorithm shows comparable performance with all the algorithms; the confidence intervals show a lack of statistical significance in the difference despite FeSA achieving more robust detection rates. A ransomware dataset from Navarra University (Berrueta et al., 2020), specifically designed to model the evolution of ransomware, is used to assess the FeSA Layer. The performance of the FeSA Layer on the Navarra dataset is presented in Table 4.8. The FeSA Layer can produce a feature set that achieves a detection rate above 99% when trained using cross-validation; however, all the feature sets produced by the alternative feature selection algorithms also achieve similar results. The advantage of FeSA is made more evident in the zero-day detection statistics. The feature set produced by FeSA maintains a detection rate of 85.1%, only behind the feature set produced by the best first algorithm. The FeSA layer has shown a clear and consistent ability to build feature sets resilient to concept drift, demonstrated across multiple datasets. The test data, which is the zero-day set, contains 114 ransomware behaviour trace samples, and the FeSA layer feature set shows a wide confidence interval similar to the other algorithms, so there is a lack of statistical significance in the performance difference between FeSA and the other algorithms. The changes in detection rate for each algorithm from test batch to test batch are not statistically significant except in the case of the difference in Harmony search performance from 13-17 to 13-18.

Table 4.8 Navarra University Dataset with Concept Drift

	FeSA	**Best First**	**Evolutionary Search**	**Genetic Search**	**Greedy Stepwise**	**Harmony Search**
Feature Count	**32**	**97**	**114**	**109**	**106**	**111**
Tested on Training Distribution	Detection: 99.9%(99.8%, 99.9%) FPR: 0.4%(0.2%, 0.5%) Precision: 0.998(0.997, 0.998) Recall: 0.999(0.998, 0.999)	Detection: 99.4%(99.1%, 99.6%) FPR: 0.4%(0.2%, 0.5%) Precision: 0.998(0.997, 0.998) Recall: 0.994(0.991, 0.996)	Detection: 99.8%(99.6%, 99.9%) FPR: 0.4%(0.2%, 0.5%) Precision: 0.998(0.997, 0.998) Recall: 0.998(0.996, 0.999)	Detection: 99.9%(99.8%, 99.9%) FPR: 0.4%(0.2%, 0.5%) Precision: 0.998(0.997, 0.998) Recall: 0.999(0.998, 0.999)	Detection: 99.6%(99.3%, 99.7%) FPR: 0.4%(0.2%, 0.5%) Precision: 0.996(0.995, 0.997) Recall: 0.996(0.993, 0.997)	Detection: 99.8%(99.6%, 99.9%) FPR: 0.4%(0.2%, 0.5%) Precision: 0.998(0.997, 0.998) Recall: 0.998(0.996, 0.999)
Tested on Zero-Day Distribution	Detection: 85.1%(77.2%, 91.1%) FPR: 14.9%(14.3%, 15.5%) Precision: 0.851(0.844, 0.857) Recall: 0.851(0.772, 0.911)	Detection: 78.9%(70.3%, 86.0%) FPR: 21.1%(20.3%, 21.7%) Precision: 0.789(0.782, 9.796) Recall: 0.789(0.703, 0.860) +	Detection: 78.1%(69.4%, 85.3%) FPR: 21.9%(21.2%, 22.6%) Precision: 0.781(0.774, 0.788) Recall: 0.781(0.694, 0.853)	Detection: 78.8%(73.2%, 88.2%) FPR: 21.2%(20.5%, 21.9%) Precision: 0.788(0.781, 0.794) Recall: 0.788(0.732, 0.882)	Detection : 87.7%(83.4%, 95.1%) FPR: 12.3%(11.7%, 12.9%) Precision: 0.877(0.873, 0.885) Recall: 0.877(0.834, 0.951)	Detection: 78.1%(69.4%, 85.3%) FPR: 21.9%(21.2%, 22.6%) Precision: 0.781(0.777, 0.784) Recall: 0.781(0.694, 0.853)

4.5 FeSAD Experiments

The FeSAD system was reviewed as a whole in the next stage of experiments; this stage exposed the FeSAD system to different concept drift scenarios. We initially review FeSAD using the HEOM metric as this was identified as a strong candidate for detecting drift in malware. The FeSAD system is evaluated using the three highest-performing algorithms from Table 4.4: the Random Forest, Bayesian Network and the Multi-Layer Perceptron with five hidden layers. The Random Forest, Bayesian Network and the Multi-Layer Perceptron showed the most natural resistance to concept drift in the initial experiments and are logical choices to use as the underlying algorithms for the FeSAD system. The deep neural network setup used in Weka was not chosen due to the complexity of the DeepLearning4J configuration and the overall time required for using it. The other algorithms used in Table 4.4 are different configurations of Tree type algorithms or neural networks; therefore, they were viewed as inadequate candidates for the FeSAD system. The results of the experiments using the Random Forests are shown in Table 4.9, with the algorithms using the configurations from Table 4.2. The experiments shown in Tables 4.9, 4.10 and 4.11 were split into test batches. The test batches consist of a training set, a test set and a calibration set; each data set would contain ransomware and benign files; training data has ransomware and benign files from one time period, likewise with the calibration set and test data. The detection drop represents the drop in the detection rate of the classifier when exposed to the new test data in each test batch. The general concept would be to train a classifier on the training set and understand the levels of ”drift” present in the data. The calibration set would help calibrate these drift values to account for concept drift over time, and the test set is then used to test the framework's performance when exposed to ransomware from a different time period likely to show concept drift.

Table 4.9 Random Forests with HEOM Performance

Test Batch Number	**Setup**	**Detection Without Feature Reduction**	**Detection Without Drift Calibration**	**Detection with FeSAD**
1	**Train:** 2013-15 **Test:** 2016 **Calibration:** 2017-18	**Detection:** 73.8%(61.5%, 83.9%) **Detection Drop:**22.7% **FPR:** 19.7%(14.5%, 24.0%) **Precision:** 0.788(0.741, 0.830) **Recall:** 0.736(0.615, 0.839)	**Detection:**92.3%(86.1%, 99.3%) **FPR:** 7.4%(4.5%, 11.0%) **Precision:** 0.926(0.876, 0.976) **Recall:** 0.932(, 0.893, 0.951)	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 2.1%(0.8%, 4.7%) **Precision:** 0.979(0.950, 0.987) **Recall:** 0.973(0.893, 0.996) **Unexpected Drift Reduction:** 50%
2	**Train:** 2013-15 **Test:** 2018 **Calibration:** 2016-17	**Detection:** 90.6%(80.9%, 96.5%) **Detection Drop:**5.7% **FPR:** 8.0%(5.1%, 11.9%) **Precision:** 0.941(0.911, 0.964) **Recall:** 0.906(0.809, 0.965)	**Detection:** 98.4%(97.3%, 99.5%) **FPR:** 0.6%(0.5%, 0.7%) **Precision:** 0.993(0.992, 0.994) **Recall:** 0.984(0.911, 0.964)	**Detection:** 100%(94.5%, 100%) **FPR:** 0%(0%, 1.3%) **Precision:** 1.0(0.989, 1.0) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 42%
3	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2017-18	**Detection:** 84.6%(73.5%, 92.4%) **Detection Drop:** 11.1% **FPR:** 12.0%(8.4%, 16.4%) **Precision:** 0.876(0.836, 0.910) **Recall:** 0.852(0.735, 0.924)	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 8.3%(5.4%, 12.3%) **Precision:** 0.914(0.854, 0.974) **Recall:** 0.883(0.879, 0.942)	**Detection:**92.3%(82.9%, 97.5%) **FPR:** 3.4%(1.3%, 5.5%) **Precision:** 0.967(0.943, 0.984) **Recall:** 0.923(0.829, 0.975) **Unexpected Drift Reduction:** 52%
4	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2018	**Detection:** 83.0%(71.7%, 91.2%) **Detection Drop:** 13.6% **FPR:** 8.0%(5.0%, 11.9%) **Precision:** 0.895(0.856, 0.925) **Recall:** 0.830(0.717, 0.912)	**Detection:** 87.8%(77.2%, 94.5%) **FPR:** 10.2%(6.9%, 14.4%) **Precision:** 0.895(0.856, 0.925) **Recall:** 0.878(0.772, 0.945)	**Detection:** 90.7%(80.9%, 96.5%) **FPR:** 5.0%(2.8%, 8.4%) **Precision:** 0.947(0.918, 0.969) **Recall:** 0.900(0.809, 0.965) **Unexpected Drift Reduction:** 62%
5	**Train:** 2013-18 **Test:** 2019 **Calibration:** 2020-21	**Detection:** 86.2%(75.3%, 93.4%) **Detection Drop:** 9.8% **FPR:** 12.0%(8.4%, 16.4%) **Precision:** 0.898(0.859, 0.927) **Recall:** 0.866(0.753, 0.934)	**Detection:** 89.2%(79.0%, 95.5%) **FPR:** 9.2%(5.9%, 13.1%) **Precision:** 0.907(0.873, 0.937) **Recall:** 0.899(0.790, 0.955)	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 7.1%(4.2%, 10.6%) **Precision:** 0.931(0.90, 0.957) **Recall:** 0.969(0.893, 0.996) **Unexpected Drift Reduction:** 50%
6	**Train:** 2013-18 **Test:** 2020-21 **Calibration:** 2019	**Detection:** 90.8%(80.9%, 96.5%) **Detection Drop:** 5.8% **FPR:** 5.0%(2.8%, 8.4%) **Precision:** 0.947(0.918, 0.968) **Recall:** 0.904(0.809, 0.965)	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 7.1%(4.2%, 10.6%) **Precision:** 0.944(0.914, 0.966) **Recall:** 0.923(0.830, 0.975)	**Detection:** 94.1%(85.4%, 98.3%) **FPR:** 5.1%(2.8%, 8.4%) **Precision:** 0.927(0.893, 0.952) **Recall:** 0.941(0.854, 0.983) **Unexpected Drift Reduction:** 48%
7	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2016-17	**Detection:** 84.2%(73.5%, 92.4%) **Detection Drop:** 11.1% **FPR:** 12.0%(8.4%, 16.4%) **Precision:** 0.884(0.847, 0.917) **Recall:** 0.852(0.753, 0.924)	**Detection:** 88.3%(80.1%, 96.5%) **FPR:** 8.3%(5.4%, 12.3%) **Precision:** 0.914(0.880, 0.942) **Recall:** 0.883(0.801, 0.965)	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 6.3%(3.6%, 9.7%) **Precision:** 0.936(0.904, 0.959) **Recall:** 0.926(0.830, 0.975) **Unexpected Drift Reduction:** 52%
8	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2016-17	**Detection:** 82.8%(69.9%, 90.0%) **Detection Drop:** 13.6% **FPR:** 8.0%(5.0%, 11.9%) **Precision:** 0.859(0.817, 0.894) **Recall:** 0.828(0.699, 0.900)	**Detection:** 87.8%(77.2%, 94.5%) **FPR:** 10.2%(6.9%, 14.4%) **Precision:** 0.895(0.856, 0.924) **Recall:** 0.878(0.772, 0.945)	**Detection:** 90.7%(81.0%, 96.5%) **FPR:** 7.1%(4.5%, 11.0%) **Precision:** 0.927(0.897, 0.954) **Recall:** 0.901(0.810, 0.965) **Unexpected Drift Reduction:** 49%

The experimental setup used to obtain the results shown in Table 4.9 involves exposing the FeSAD framework to various concept drift scenarios. The years of origin for each test and training batch are not critical factors in the experiments; the idea is to expose the model to test data from a different distribution. A new set of ransomware families with potentially different behavioural patterns to the training set ransomware is a different distribution. In a realistic scenario, it would be impossible to train on data from the future and test or calibrate on data from the present or past. Suppose we view the training as from time t. In that case, the calibration from time $t+1$ and the test data from time $t+2$, the year of the data is assigned to these variables, is irrelevant as the data assigned to each variable is from different families and distributions, which in theory can have appeared in different orders and not necessarily the year they appeared; the more important thing is that there is clear drift between the samples in each distribution. This approach is used to test the versatility of FeSAD and its ability to work with different train, test and calibration combinations. The results without FeSAD use only the feature sets generated by the FeSA layer with no intervention from the drift calibration or decision layer. The first experimental batch shows that the FeSAD framework improves detection from 93.2% to 97.3%. The FeSAD framework uses the drift recorded in the training set and the drift recorded in the calibration set to update the drift thresholds; with the drift updated thresholds from the calibration set, the level of samples showing unexpected drift reduces by 50%. The false positive rate reduced from 7.4 to 2.1% in test batch 1; however, the false positives were low to begin with. It is essential that the FeSAD framework can reduce the misclassification of benign and ransomware samples and ensure it does not cause an increased amount of false positives. Test batch 2 sees a 100% detection rate with the FeSAD framework; however, with a reduced unexpected drift rate than the first test batch. The false positive rate reduces by 0.6% which is negligible; however, in this batch, the false positive rate is at 0 with the FeSAD framework. The samples being classed as ransomware while being benign could be due to benign samples behaving like ransomware or displaying API call-like behaviour that matches the ransomware behavioural profile. The unexpected drift level reduced less than the first test batch can be down to various factors introduced by the ransomware from 2018 and their behavioural patterns being harder to distinguish from benign files statistically. Test batches 3 and 4 show an improvement in detection with the FeSAD framework, with both batches maintaining detection rates above 90%. Test batches 3 and 4 demonstrate the longevity that can be achieved by a model if calibrated according to the expected drift. Test batches 3 and 4 show a higher level of false positives with 3.4% and 5.0%, which indicates a shift in ransomware and benign files; however, the maintenance of the high detection rate without a very high false positive rate is still positive. FeSAD significantly improves the false positive rate in test batches 3 and 4 compared to the results achieved by the FeSA layer without drift metrics. The base model was trained on ransomware and benign files from 2013 to 2015, whereas the test data in batch 3 is from 2019, and the test data for batch 4 is from 2020-21. Test batch 5 shows a significant increase in detection rate from 89.9% to 96.8%; however, the false positive rate remains higher than desired, only reducing to 7.1%. Test batches 7 & 8 show the potential longevity of the FeSAD framework and its potential to be effective across many different distributions as the framework maintains a high detection rate despite being trained with 2013-15 and, calibrated with 2016-17, and tested on 2019 and 2020-21 data. The expected drift and unacceptable drift levels are initially calibrated according to the training set and calibrated according to changes in drift observed in the calibration set. If the threshold for samples showing unexpected or unacceptable drift is exceeded, the system must be retrained. The unexpected drift reduction in Table 4.10 shows the reduction of samples showing unexpected levels of drift using the calibrated drift values using the calibration and training sets. The importance of drift calibration is shown in Table 4.9 as it helps define drift thresholds which better equips the framework to anticipate drift and operate reliably for longer. The FeSAD framework can consistently reduce false positives; however, test batches 7 and 8 show a false positive rate closer to test batch 5 which may be down to benign samples that show behaviour close to ransomware files. There is a significant reduction in unexpected drifts across all test batches using the drift calibration in the FeSAD framework. In addition to an improved detection rate, the FeSAD framework reduces false positives and maintains longevity for a classifier. Taking confidence intervals into account the performance difference between using the FeSAD and not using FeSAD requires higher detection rates to achieve statistical significance. It is also noted that the difference in detection rate from batch to batch is not statistically significant.

Table 4.10 MLP with HEOM Performance

Test Batch Number	**Setup**	**Detection Without Feature Reduction**	**Detection Without Drift Calibration**	**Detection with FeSAD**
1	**Train:** 2013-15 **Test:** 2016 **Calibration:** 2017-18	**Detection:** 70.7%(56.6%, 80.0%) **Detection Drop:** 24.7% **FPR:** 21.7%(17.1%, 27.2%) **Precision:** 0.767(0.719, 0.812) **Recall:** 0.717(0.566, 0.800)	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 2.5%(1.0%, 5.2%) **Precision:** 0.974(0.950, 0.989) **Recall:** 0.969(0.893, 0.996)	**Detection:** 100.0%(94.5%, 100%) **FPR:** 1.5%(0.3%, 3.0%) **Precision:** 0.985(0.966, 0.995) **Recall:** 1.00(0.945, 1.0) **Unexpected Drift Reduction:** 55%
2	**Train:** 2013-15 **Test:** 2018 **Calibration:** 2016-17	**Detection:** 92.3%(82.9%, 97.5%) **Detection Drop:**3.7% **FPR:** 7.7%(5.1%, 11.9%) **Precision:** 0.961(0.936, 0.975) **Recall:** 0.923(0.829, 0.975)	**Detection:** 98.4%(91.7%, 99.9%) **FPR:** 4.1%(2.3%, 7.5%) **Precision:** 0.960(0.936, 0.979) **Recall:** 0.984(0.917, 0.999)	**Detection:** 100.0%(94.5%, 100%) **FPR:** 3.9%(2.0%, 7.0%) **Precision:** 0.962(0.934, 0.979) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 47%
3	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2017-18	**Detection:** 53.8%(41.0%, 66.3%) **Detection Drop:** 41.6% **FPR:** 38.8%(33.1%, 44.9%) **Precision:** 0.585(0.530, 0.638) **Recall:** 0.538(0.410, 0.663)	**Detection:** 80.6%(68.2%, 88.9%) **FPR:** 17.4%(13.1%, 22.4%) **Precision:** 0.822(0.779, 0.863) **Recall:** 0.806(0.682, 0.889)	**Detection:** 92.3%(82.9%, 97.5%) **FPR:** 7.4%(4.5%, 11.0%) **Precision:** 0.923 **Recall:** 0.923(0.829, 0.975) **Unexpected Drift Reduction:** 52%
4	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2018	**Detection:** 41.5%(29.4%, 54.4%) **Detection Drop:** 55.0% **FPR:** 50.2%(44.1%, 56.2%) **Precision:** 0.429(0.376, 0.484) **Recall:** 0.414(0.294, 0.544)	**Detection:** 83.1%(71.7%, 91.2%) **FPR:** 15.6%(11.5%, 20.5%) **Precision:** 0.841(0.798, 0.878) **Recall:** 0.831(0.717, 0.912)	**Detection:** 87.0%(77.2%, 94.5%) **FPR:** 3.8%(2.1%, 7.0%) **Precision:** 0.958(0.925, 0.973) **Recall:** 0.870(0.772, 0.945) **Unexpected Drift Reduction:** 47%
5	**Train:** 2013-18 **Test:** 2019 **Calibration:** 2020-21	**Detection:** 72.4%(59.8%, 82.7%) **Detection Drop:** 22.2% **FPR:** 22.5%(17.7%, 27.9%) **Precision:** 0.770(0.722, 0.814) **Recall:** 0.724(0.598, 0.827)	**Detection:** 90.7%(80.9%, 96.5%) **FPR:** 9.3%(6.4%, 13.5%) **Precision:** 0.906 **Recall:** 0.907(0.809, 0.965)	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 3.5%(1.8%, 6.6%) **Precision:** 0.965(0.939, 0.981) **Recall:** 0.969(0.893, 0.996) **Unexpected Drift Reduction:** 40%
6	**Train:** 2013-18 **Test:** 2020-21 **Calibration:** 2019	**Detection:** 72.4%(59.8%, 82.7%) **Detection Drop:** 24.0% **FPR:** 23.9%(19.1%, 29.5%) **Precision:** 0.751(0.700, 0.795) **Recall:** 0.724(0.598, 0.827)	**Detection:** 83.1%(71.7%, 91.2%) **FPR:** 6.3%(3.9%, 10.1%) **Precision:** 0.929(0.897, 0.954) **Recall:** 0.831(0.717, 0.912)	**Detection:** 92.3%(82.9%, 97.5%) **FPR:** 7.8%(4.9%, 11.4%) **Precision:** 0.922 **Recall:** 0.923(0.829, 0.975) **Unexpected Drift Reduction:** 54%
7	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2016-17	**Detection:** 54.8%(41.0%, 66.3%) **Detection Drop:** 41.6% **FPR:** 38.8%(33.1%, 44.9%) **Precision:** 0.585(0.531, 0.638) **Recall:** 0.548(0.410, 0.663)	**Detection:** 80.0%(68.2%, 88.9%) **FPR:** 17.4%(13.2%, 22.5%)% **Precision:** 0.822(0.779, 0.863) **Recall:** 0.806(0.682, 0.889)	**Detection:** 91.0%(80.9%, 96.5%) **FPR:** 8.2%(5.4%, 12.2%) **Precision:** 0.918(0.887, 0.947) **Recall:** 0.916(0.809, 0.965) **Unexpected Drift Reduction:** 48%
8	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2016-17	**Detection:** 41.5%(29.4%, 54.4%) **Detection Drop:** 55.0% **FPR:** 50.0%(44.1%, 56.2%) **Precision:** 0.429(0.376, 0.484) **Recall:** 0.415(0.294, 0.544)	**Detection:** 83.1(71.7%, 91.2%) **FPR:** 15.6%(11.6%, 20.5%) **Precision:** 0.841(0.798, 0.879) **Recall:** 0.831(0.717, 0.912)	**Detection:** 87.7%(77.2%, 94.5%) **FPR:** 11.7%(7.4%, 15.9%) **Precision:** 0.882(0.843, 0.915) **Recall:** 0.877(0.772, 0.945) **Unexpected Drift Reduction:** 49%

Table 4.11 shows the results of the experiments carried out with a Multi-Layer Perceptron as an underlying algorithm. The experimental results shown in table 4.10 follow the same format as the results shown in Table 4.9; performance statistics without FeSAD are the performance results of the feature set produced by the FeSA Layer with no intervention from the drift calibration or decision layers. The Multi-Layer Perceptron shows an unpredictable and less uniform degradation when exposed to concept drift; this reinforces that the evolution of ransomware behaviour will not necessarily uniformly degrade a machine learning classifier. Table 4.10 shows that the Multi-Layer Perceptron suffers from a higher rate of false positives than the Random Forest; however, the FeSAD framework can compensate for this shortcoming through its adaptability to concept drift. The FeSAD framework is relied upon more heavily when using the multi-layer perceptron as the results are surprisingly different to the initial tests in Table 4.4. The multi-layer perceptron shows a higher vulnerability to concept drift than the Random Forest, which allows the FeSAD framework to show its capabilities. Test Batch 1 and 2 achieved a perfect detection rate with the FeSAD framework; the FeSA feature set helps significantly increase detection and accuracy for both test batches, and the FeSAD framework further improves on this by detecting and correctly classifying all drifting samples. The false positive rate in test batch 1 and 2 are low, and the FeSAD framework helps reduce it further. Test batches 3 and 4 are more affected by concept drift, which shows in the detection rates without feature reduction and FeSAD; with detection rates dropping by 33% in test batch 3 and 47% in test batch 4. The FeSAD framework can lift detection rates between the high 80s and low 90s in both test batches, significantly improving performance without FeSAD. The false positive rates for both test batches are higher than in batches 1 and 2; however, the FeSAD framework can reduce the false positives by 10% in test batch 3 and 3.9% in test batch 4. Test batches 5 and 6 show that the FeSAD framework achieves detection rates in the 90% range. Test batch 5 shows a 6.5% increase in detection, and test batch 6 shows a 9.7% increase in detection rate by using the FeSAD adaption. Test batch 5 shows a reduction of false positives on 5.8%; however, for the first time test batch 6 sees an increase in false positives from 6.4% to 7.8%; this could be down to a high volume of dubious benign samples in the test batch and shortcomings in the MLP which shows a generally higher level of false positives than the Random Forest. Test batches 7 and 8 fail to achieve detection rates as high as with the corresponding test batches using the random forest; however, the detection drop-off for these test batches without the FeSAD framework is significantly higher for the MLP algorithm than the random forest. The FeSAD framework has to compensate for significantly lower detection rates and can achieve detection rates in the high 80% range for both test batches 7 and 8. Table 4.10 shows that the reduction in unexpected drift remains competitive with the results in Table 4.9. The reduction in unexpected drift shows that the drift calibration used with the MLP algorithm helps the framework to anticipate better drift in the samples allowing it to stay accurate and reliable for longer. Taking confidence intervals into account, the FeSAD framework does show a statistically significant improvement in performance in all test batches besides Test Batch 2, 3 ,4 and 8. It must be noted the difference in detection between test batches are not statistically significant.

Table 4.11 Bayesian Network with HEOM Performance

Test Batch Number	**Setup**	**Detection Without Feature Reduction**	**Detection Without Drift calibration**	**Detection with FeSAD**
1	**Train:** 2013-15 **Test:** 2016 **Calibration:** 2017-18	**Detection:** 75.3%(63.1%, 85.2%) **Detection Drop:** 17.0% **FPR:** 23.7%(19.1%, 29.1%) **Precision:** 0.762(0.713, 0.806) **Recall:** 0.753(0.631, 0.852)	**Detection:** 87.7%(77.2%, 94.5%) **FPR:** 12.9%(9.3%, 17.7%) **Precision:** 0.871(0.833, 0.906) **Recall:** 0.877(0.772, 0.945)	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 3.2%(1.5%, 6.1%) **Precision:** 0.968(0.947, 0.986) **Recall:** 0.969(0.893, 0.996) **Unexpected Drift Reduction:** 58%
2	**Train:** 2013-15 **Test:** 2018 **Calibration:** 2016-17	**Detection:** 90.8%(80.9%, 96.5%) **Detection Drop:**2.6% **FPR:** 11.7%(8.2%, 16.0%) **Precision:** 0.885(0.847, 0.917) **Recall:** 0.908(0.809, 0.965)	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 8.3%(5.1%, 11.9%) **Precision:** 0.917(0.883, 0.945) **Recall:** 0.923(0.830, 0.975)	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 2.9%(1.2%, 5.7%) **Precision:** 0.971(0.947, 0.986) **Recall:** 0.969(0.969, 0.893, 0.996) **Unexpected Drift Reduction:** 39%
3	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2017-18	**Detection:** 87.7%(77.1%, 94.5%) **Detection Drop:** 5.8% **FPR:** 14.2%(10.2%, 18.9%) **Precision:** 0.859(0.820, 0.897) **Recall:** 0.877(0.771, 9.945)	**Detection:** 89.3%(79.1%, 95.6%) **FPR:** 11.4%(8.10%, 16.1%) **Precision:** 0.886(0.847, 0.917) **Recall:** 0.893(0.791, 0.956)	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 9.1%(5.9%, 13.1%) **Precision:** 0.909(0.873, 0.937) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 47%
4	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2018	**Detection:** 75.3%(63.1%, 85.2%) **Detection Drop:** 17.0% **FPR:** 23.7%(18.7%, 29.1%) **Precision:** 0.817(0.772, 0.857) **Recall:** 0.753(0.631, 0.852)	**Detection:** 87.7%(77.2%, 94.5%) **FPR:** 13.0%(9.3%, 17.7%) **Precision:** 0.871(0.830, 0.904) **Recall:** 0.877(0.772, 0.945)	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 9.2%(5.9%, 13.1%) **Precision:** 0.907(0.869, 0.935) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 42%
5	**Train:** 2013-18 **Test:** 2019 **Calibration:** 2020-21	**Detection:** 87.7%(77.2%, 94.5%) **Detection Drop:** 5.8% **FPR:** 14.1%(10.3%, 18.9%) **Precision:** 0.856(0.814, 0.891) **Recall:** 0.877(0.772, 0.945)	**Detection:** 93.8%(85.0%, 98.3%) **FPR:** 5.3%(3.1%, 8.8%) **Precision:** 0.947(0.918, 0.968) **Recall:** 0.938(0.850, 0.983)	**Detection:** 98.5%(91.7%, 99.9%) **FPR:** 1.5%(0.4%, 3.7%) **Precision:** 0.985(0.966, 0.995) **Recall:** 0.985(0.917, 0.999) **Unexpected Drift Reduction:** 46%
6	**Train:** 2013-18 **Test:** 2020-21 **Calibration:** 2019	**Detection:** 86.2%(75.3%, 93.5%) **Detection Drop:** 6.7% **FPR:** 14.8%(10.9%, 19.7%) **Precision:** 0.853(0.811, 0.889) **Recall:** 0.862(0.735, 0.935)	**Detection:** 90.7%(81.0%, 96.5%) **FPR:** 9.3%(6.3%, 13.5%)% **Precision:** 0.907(0.870, 0.935) **Recall:** 0.907(0.810, 0.965)	**Detection:** 94.0%(89.3%, 99.6%) **FPR:** 5.5%(3.6%, 9.7%) **Precision:** 0.946(0.918, 0.968) **Recall:** 0.940(0.893, 0.996) **Unexpected Drift Reduction:** 45%
7	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2016-17	**Detection:** 87.7%(77.1%, 94.5%) **Detection Drop:** 5.8% **FPR:** 14.2%(10.2%, 18.9%) **Precision:** 0.865(0.823, 0.899) **Recall:** 0.871(0.771, 0.945)	**Detection:** 89.3%(79.1%, 95.6%) **FPR:** 11.4%(8.1%, 16.0%) **Precision:** 0.881(0.843, 0.915) **Recall:** 0.893(0.791, 0.956)	**Detection:** 90.7%(81.0%, 98.3%) **FPR:** 9.6%(6.6%, 13.9%) **Precision:** 0.932 **Recall:** 0.907(0.810, 0.983) **Unexpected Drift Reduction:** 52%
8	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2016-17	**Detection:** 75.3%(63.1%, 82.5%) **Detection Drop:** 17.0% **FPR:** 23.7%(19.1%, 29.1%) **Precision:** 0.762(713, 0.806) **Recall:** 0.753(0.631, 0.852)	**Detection:** 87.7%(77.2%, 94.5%) **FPR:** 13.0%(9.3%, 11.9%) **Precision:** 0.871(0.830, 0.904) **Recall:** 0.871(0.772, 0.945)	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 8.3%(5.1%, 11.9%) **Precision:** 0.917(0.883, 0.945) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 54%

The results in Table 4.11 with a Bayesian Network follow a similar pattern to the results of the Random Forest; however, the Bayesian Network does not suffer the degradation that the MLP suffers. The Bayesian network suffers most from false positives without the intervention of the FeSAD framework. The FeSAD framework limits the false positive rate to an average of 6.2% across all 8 test batches, which is a significant reduction from an average false positive rate of 17.1% without using the FeSAD framework. Table 4.11 uses the identical experimental setups that Tables 4.9 and 4.10 used and demonstrates the FeSAD framework's ability to maintain an accurate and reliable classifier. The reduction in unexpected drift ranges from the low 40 to mid 50 per cent, a similar average to the other two configurations. As with the Random Forest and the MLP, the FeSAD framework maintains a high detection rate and compensates for the algorithm's degradation under concept drift. Test batches 7 and 8 show the FeSAD framework's ability to maintain longevity and adaptability across ransomware from different sample distributions. The results in test batch 3 stand out as anomalous compared to the other 6 test batches as the lowest reduction in unexpected drift; this appears consistent across the results using Random Forests, the MLP and the Bayesian Network. In terms of confidence intervals, the most significant result is shown in test Batch 1 and 8 where the biggest improvement in performance is shown when using the Bayesian Network.

4.6 Comparative Analysis

The results in Tables 4.9, 4.10 and 4.11 represent the FeSAD framework working with the three most promising algorithms from the initial experiments shown in Table 4.4; the Random Forest, the Multi-Layer Perceptron and the Bayesian Network combined with the HEOM metric. The focus of these experiments was to test the performance of the FeSAD framework in different concept drift scenarios. The tests determine whether the FeSAD framework can increase the effectiveness of a machine-learning ransomware detection system while increasing its longevity and reliability under concept drift. The first observation is the drop-off in detection rate when a base machine learning classifier experiences concept drift, which is presented in Figure 4.7. The base classifier is the same for Test Batch 1, 2, 3, 4, 7, and 8, the classifier trained on data from 2013-2015. Test Batches 5 and 6 use a base classifier trained on data from 2013-2018. Test batches 1-4 are tested on data from 2016-17, 2018, 2019, and 2020-2021 while using a base classifier trained on data from 2013-2015. Test batches 5 and 6 use a base classifier trained on 2013-2018 data and tested on data from 2019 and 2020-21.

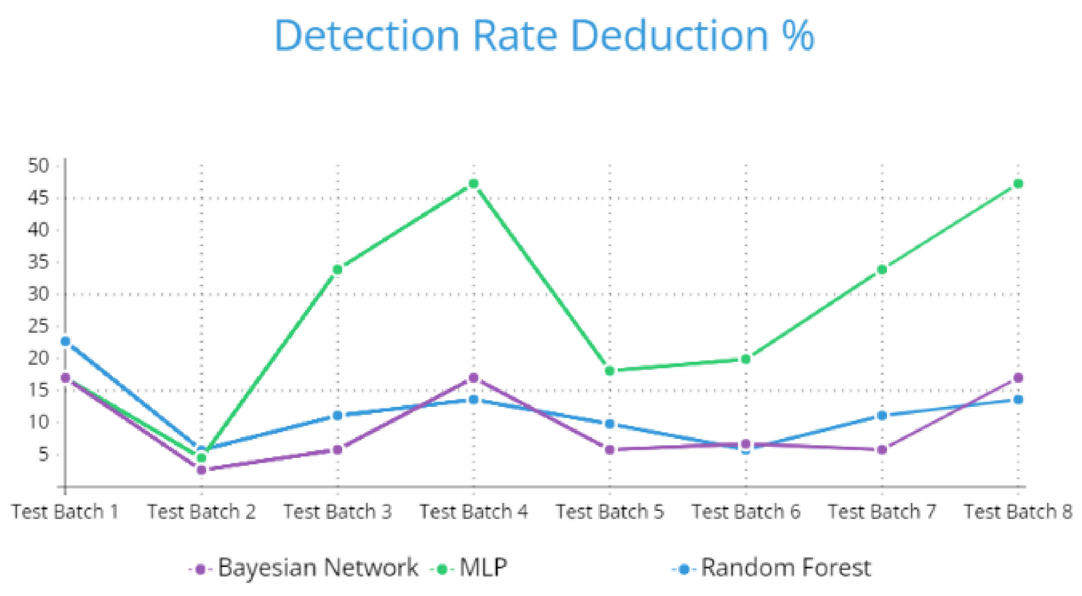

Figure 4.8: Detection Rate Deduction

The classifier that experiences the most volatile performance reduction is the Multi-Layer Perceptron, where the base classifier experiences an average detection rate reduction of 31.8% when exposed to ransomware files that are outside the distribution it has been trained on. The Bayesian Network and Random Forest suffer a similar drop-off in performance, with the Random Forest showing an average reduction in the detection rate of 10.95% and the Bayesian Network showing a reduction of 9.7% on average. All three algorithms show a detection drop-off, as has been proven by testing various algorithms in Table 4.4. The detection drop-off for the MLP is more extreme than the initial experiments showed; however, this worked well because it allowed the FeSAD framework to be tested with algorithms that show moderate performance degradation and more extreme performance degradation.

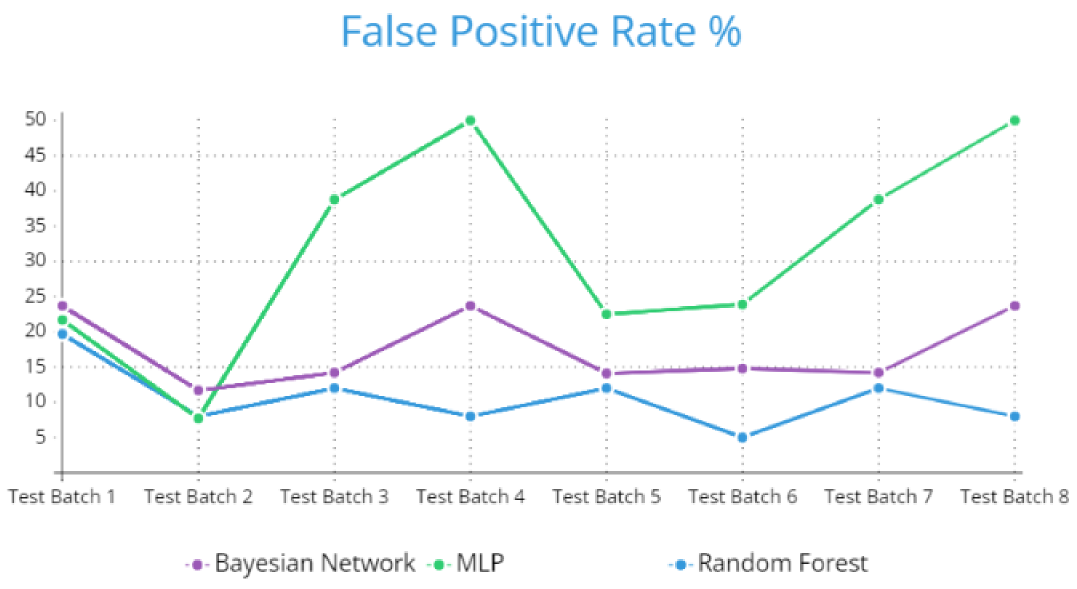

Figure 4.9: False Positive Rate without FeSAD

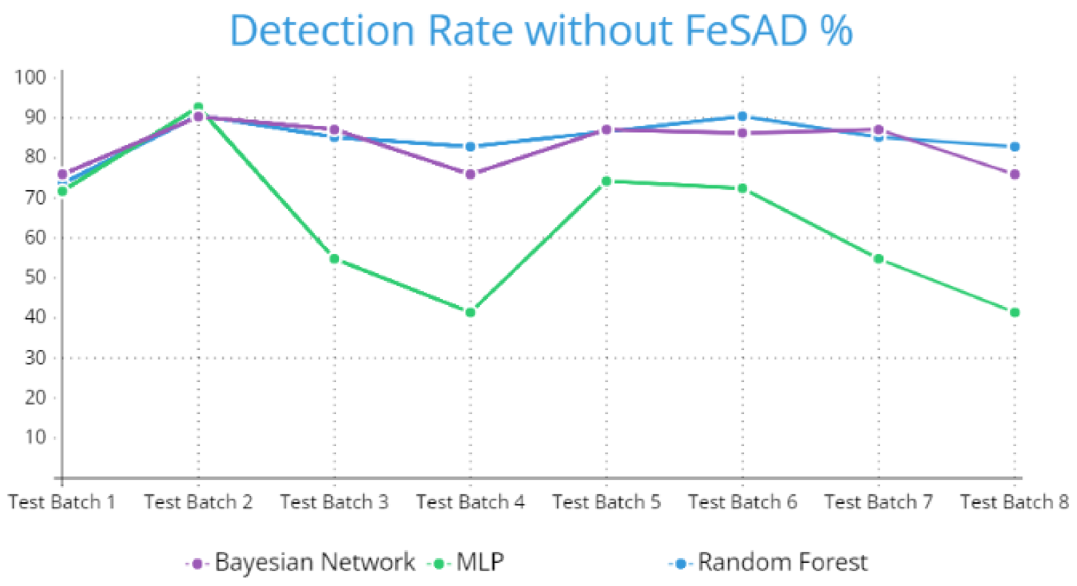

Figure 4.10: Detection Rate Without FeSAD

The false-positive rate shows a similar pattern to the detection rate drop-off; however, the Bayesian Network consistently has more false positives than the Random Forest. The Multi-Layer Perceptron struggles more than the other two algorithms, almost consistently suffering with a higher false positive rate when exposed to concept drift. Figures 4.8 and 4.9 show that it is vital for these algorithms need additional help to be useful in concept drift scenarios. The false-positive rate recorded is the overall false-positive rate of the algorithm across benign and ransomware classes.

Figure 4.9 shows the detection rate for each base classifier and its respective test data. The degradation in all three algorithms appears uniform except for Test Batch 2; however, there is evidence that concept drift in ransomware behaviour may not behave uniformly due to the irregular drop-off in detection rates for all three classifiers. The Multi-Layer Perceptron shows a sharper decline in detection rate than the other two algorithms. The degradation rate reduces in test batches 5 and 6 as they are trained with a combination of newer and older ransomware data. The evident non-uniform degradation in performance shown in Figures 4.8 and 4.9 shows that ransomware's behaviour does not change uniformly over time; the non-uniformity of ransomware change supports the experimental structure of using a combination of uniform and non-uniform calibration and training data sets.

Based on the initial results and degradation of the three algorithms, the Multi-Layer perceptron suffers higher performance degradation than the Bayesian Network and the Random Forest. The degradation patterns appear to be similar despite the algorithms showing varying levels of performance degradation. Observing figures 4.8, 4.9 and 4.10, the performance degradation affects each classifier in similar ways in every test batch despite the degradation being present in differing levels of severity.

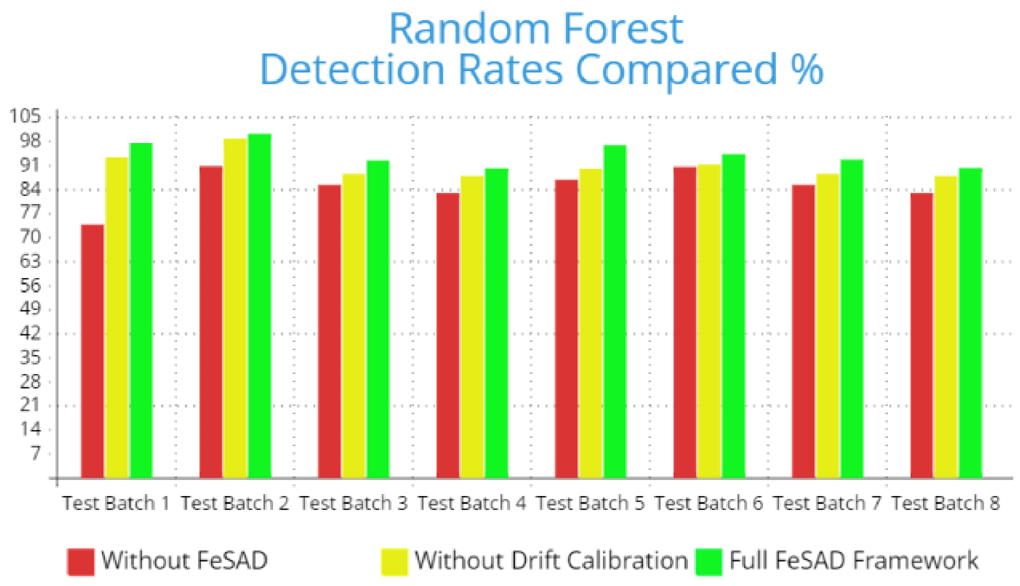

Figure 4.11: Random Forest Detection Rates Compared

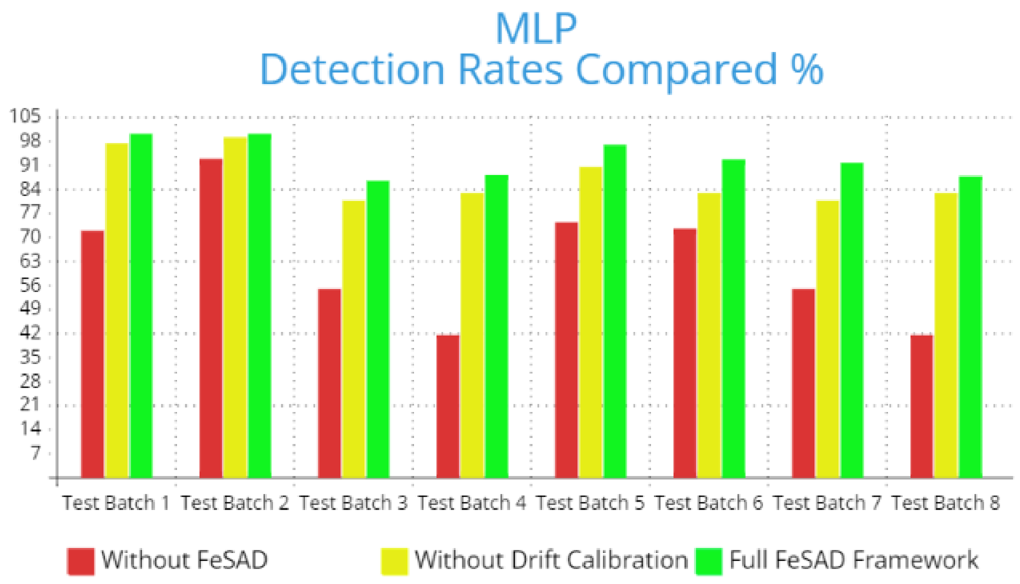

Figure 4.12: MLP Detection Rates Compared

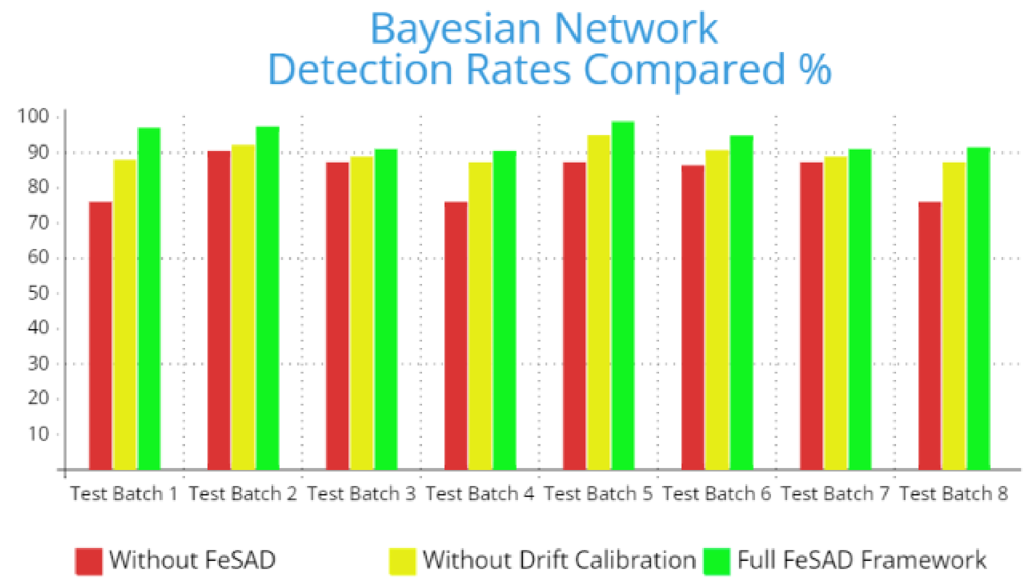

Figure 4.13: Bayesian Network Detection Rates Compared

The data presented in Figure 4.10 shows the detection rates of each test batch without FeSAD, without drift calibration and with the complete FeSAD framework when using the Random Forest as an underlying algorithm. The FeSAD framework can help maintain a healthy detection rate while improving the classifier's performance under concept drift. The FeSAD framework shows it can perform in various concept drift situations, with the FeSAD framework increasing performance in every scenario the framework is tested under. Figures 4.11 and 4.12 show similar results with the FeSAD framework improving the classifier's performance in both cases. Figure 4.12 shows the performance of the FeSAD framework with the Multi-Layer Perceptron; the main difference between the MLP and the other two algorithms is that the MLP shows far steeper performance degradation than the other two algorithms. The FeSAD framework can significantly improve the performance of an algorithm degrading under concept drift while also stabilising the performance of an algorithm showing low to moderate degradation due to concept drift. The average detection rate consistently rises with the FeSAD framework applied. Test Batch 1 shows a low detection rate across the three algorithms with an average detection rate of 73.8%; however, with the FeSAD framework, the average detection rate across the three algorithms is 98.1%. The 1st test batch shows a significant detection degradation, most likely due to a change in ransomware behaviour between 2015 and 2016. The FeSAD framework compensates for the degradation in test batch one and significantly increases detection rates across the three test algorithms. The underlying algorithms perform consistently well on Test Batch 2 with an average detection rate of 91.2%; this would be due to the ransomware's behaviour shifting more towards the behaviour defined in the training set from 2013-2015. The FeSAD framework uses data from 2017-18 to calibrate the framework for the drift values it should expect. The FeSAD framework lifts the detection rate to 99.1% on average on Test Batch 2, which is still a significant improvement from the baseline detection rate of the algorithms. Test Batch 3 shows another shift in the detection rate of the baseline classifiers to 75.7% on average; the lower average detection rate is significantly influenced by the degradation experienced by the MLP classifier that achieves a 54.8% detection rate; however, the Random Forest and Bayesian Network do also show a higher level of degradation than that seen in Test batch 2. The FeSAD framework achieves an average of 91.8% on Test Batch 3. Test batch 3 uses data from 2017-18 and is tested on data from 2019. Test batch 4 sees an average drop in detection rates to 66.7%; the steep reduction is influenced heavily by the detection in the MLP at 41.4%; however, the Random Forest and Bayesian Network show the lowest detection rates see out of the first four test batches. The average detection rate achieved by FeSAD on Test Batch 4 is 89.4% which is a significant improvement from the baseline detection rate achieved by the underlying classifiers. The reduction in performance from test batch 3 to 4 is likely owed to another shift in ransomware behaviour; in this case, it is away from the baseline established from the 2013-15 data. Test Batch 4 uses data from 2018 for the FeSAD framework to calibrate its drift thresholds. Test batches 5 and 6 use a new baseline classifier trained on 2013-18 data and calibration data from 2020-21 and 2019. The baseline classifier achieves an average detection rate of 82.6% on Test Batch 5; the MLP contributes negatively to this figure as it only achieves a detection rate f 74.2%. The FeSAD framework achieves a positive result with Test Batch 5 on all three detection rates with an average detection rate of 97.4%. The baseline classifiers achieve an average detection rate of 83% on Test Batch 6; the FeSAD framework achieves a detection rate of 93.7% on average on Test Batch 6. The baseline classifiers yield higher average performance on Test Batch 5 and 6 than the previous test batches, using newer data and a greater variety of ransomware and benign data. The FeSAD framework achieves, on average, a 93.7% detection rate on Test Batch 6. Test Batches 7 and 8 return to the baseline models created using data from 2013-15; however, in these test batches, the FeSAD framework uses calibration data from 2016-17. The calibration data in Test Batch 7 and 8 is older than the calibration data used in Test Batch 3 and 4. FeSAD achieves a 91.7% detection rate on average on Test Batch 7 and an 89.7% detection rate on average on Test Batch 8. The detection rate for Test Batch 8 is below 90% due to the detection rate with the MLP being at 87.7%. In every test case containing concept drift, as presented in Figures 4.11, 4.12 and 4.13, the FeSAD framework increases the detection rates achieved by the baseline classifier.

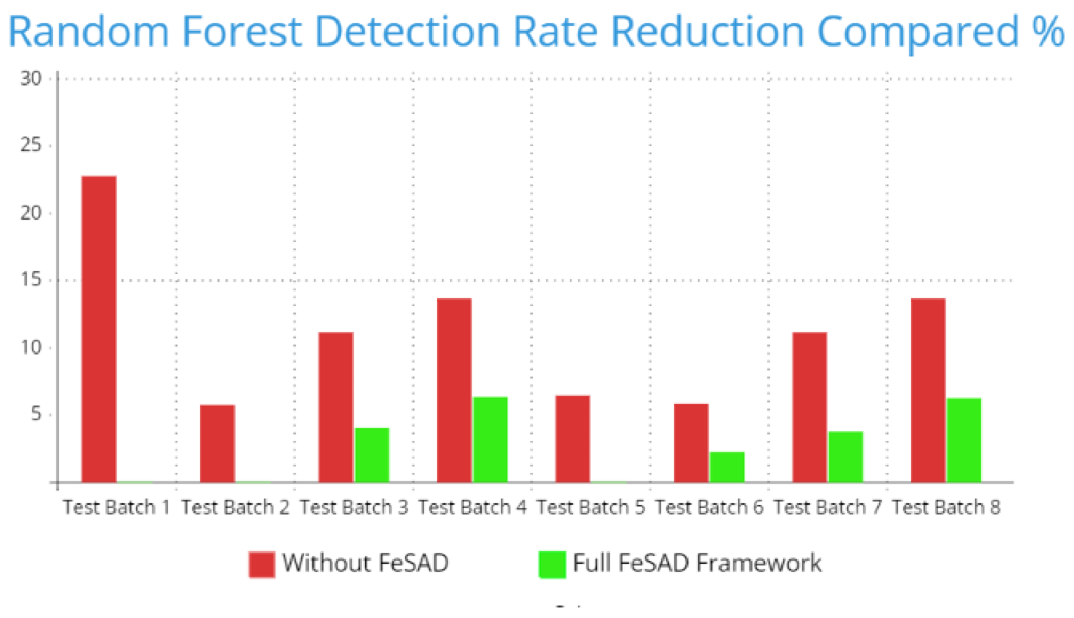

Figure 4.14: Concept Drift Data Random Forest Detection Reduction

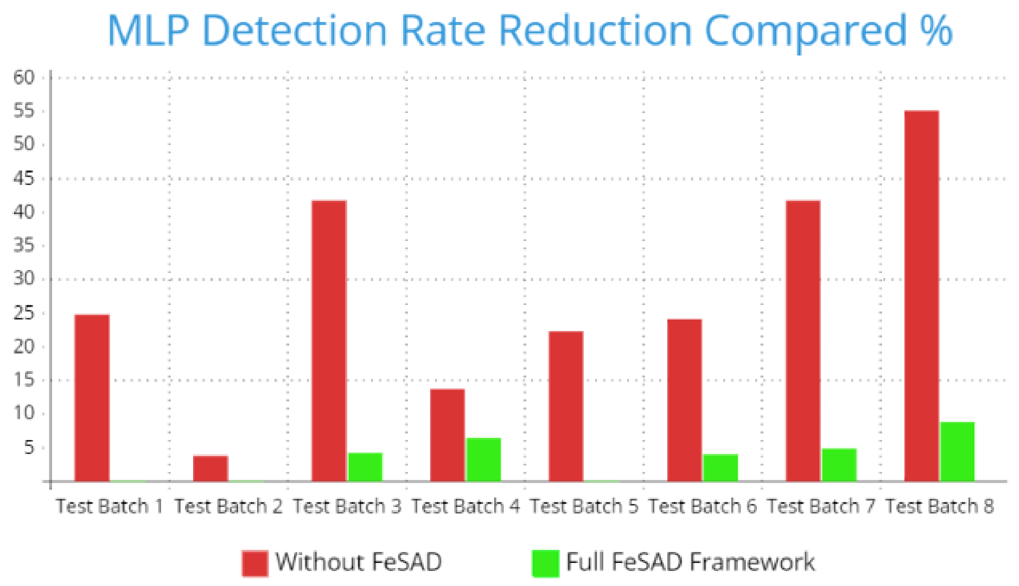

Figure 4.15: Concept Drift Data MLP Detection Rate Reduction

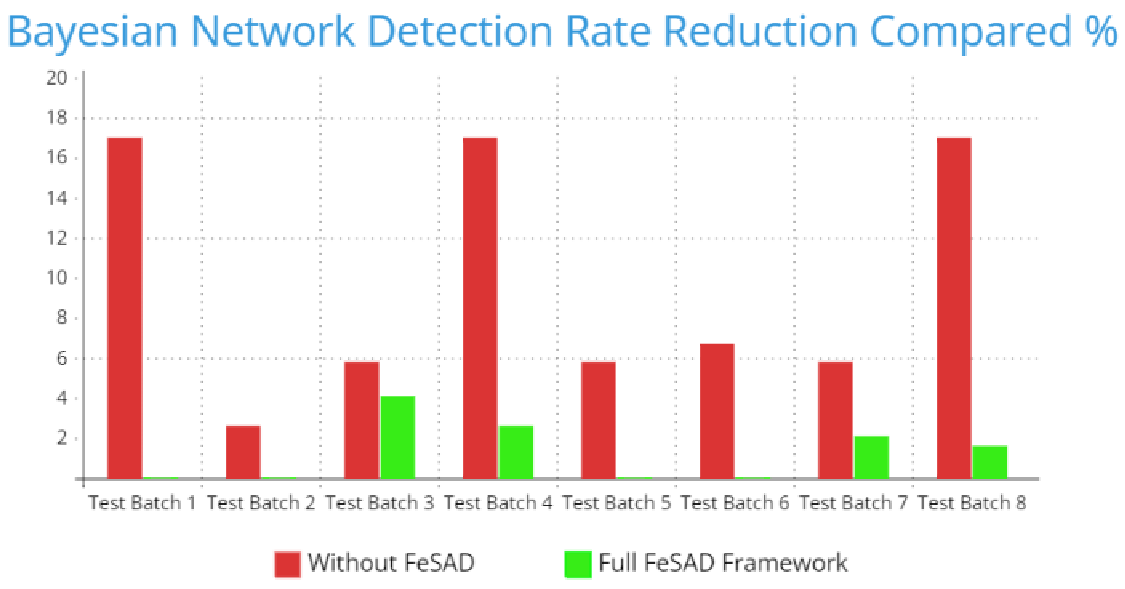

Figure 4.16: Concept Drift Data Bayesian Network Detection Reduction

Figures 4.14, 4.15, and 4.16 show the reduction in detection rate each classifier experiences when exposed to each Test Batch with and without using the FeSAD framework. The reduction in detection rate is compared to the initial detection rate when the classifier is first trained using 10-fold cross-validation. The detection rate reduction for all three algorithms is minimal when using FeSAD, especially compared to the results obtained when FeSAD is not being used. Using FeSAD with Test Batch 1 and 2 shows no reduction in detection rate across all three algorithms. The FeSAD framework allows a higher level of degradation for Test Batch 3 and 4, and the MLP seems to suffer the least relative performance drop while using FeSAD as opposed to the Random Forest and the Bayesian Network; however, the base classifier for the two latter classifiers was significantly more robust than the MLP in general. Figure 4.15 shows the capability of the FeSAD framework with initial degradation rates above 40% being brought down to below 5%. The FeSAD framework performs consistently with significant degradation reduction for each algorithm; however, the Bayesian network shows the least overall degradation when using the FeSAD framework. Overall, the Bayesian network experiences an average degradation of 1.3% across all 8 Test Batch scenarios, whereas the Random Forest and MLP do not achieve as low a degradation level with FeSAD. The MLP benefits the most from FeSAD, according to figure 4.15, as its levels of detection degradation reduce most of the three algorithms. The critical takeaway from Figures 4.14, 4.15 and 4.16 is that the FeSAD framework significantly reduces the impact of concept drift on a classifier and allows for reliable classification under concept drift.

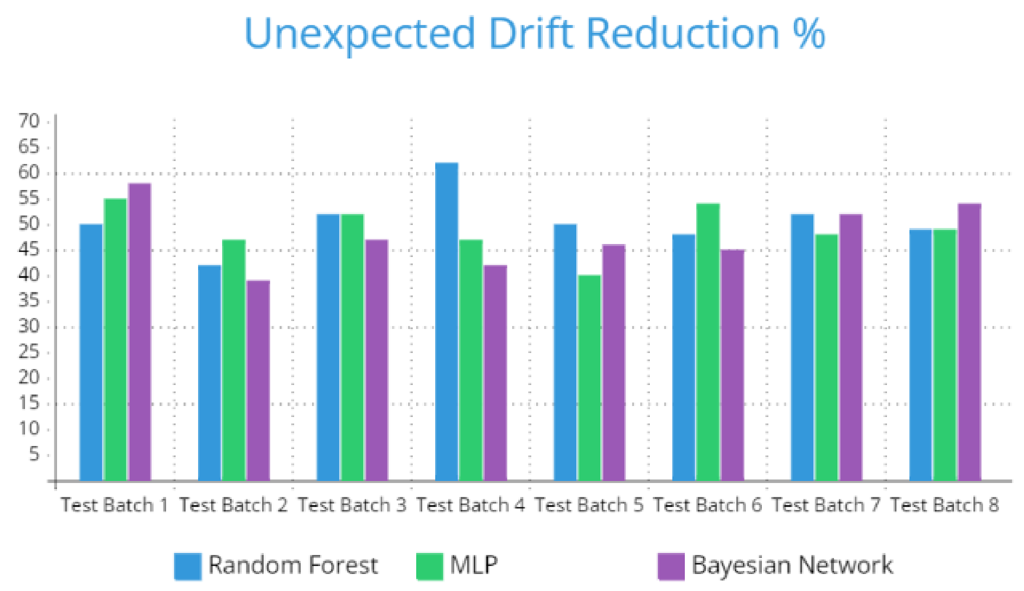

Figure 4.17: Unexpected Drift Reduction Using FeSAD

Figure 4.17 shows that the reduction rate is unexpected drift when the FeSAD framework applies the drift calibration and decision layers. The unexpected drift represents samples in each test batch that show a level of drift that the framework does not expect; the reduction rate of the value shows how the rate of samples showing unexpected drift reduces when the FeSAD framework's drift calibration layer is used. In each test batch, the initial drift threshold values are calculated using only the training set, and the number of samples showing unexpected drift is calculated using the test set. The FeSAD framework then calibrates the expected drift values in the model using the calibration dataset, and the reduction in samples showing unexpected drift is calculated. The average unexpected drift reduction rate using the Random Forest algorithm is 50.6%. The reduction rate of samples showing unexpected drift when using the MLP is 49%, and the average reduction rate of samples showing unexpected drift when using the Bayesian Network is 47.9%. The reduction rate of samples showing unexpected drift is consistently similar across all three algorithms showing that the FeSAD framework can achieve consistent results with different algorithms. We note irregularities in test batch 4, likely due to the calibration being done with 2018 data. There was a clear difference in drift between the 2015 and 2018 data when implemented with the Random Forest; therefore, the thresholds for future drift were better accounted for leading to a higher reduction in abnormal samples. The consistent decrease in samples showing drift the framework does not expect shows that the FeSAD framework can effectively classify samples that show concept drift and reduce the effects of concept drift on model uncertainty. Using calibration data sets containing data from new ransomware strains and benign files to tune drift values accounts for the reduction in unexpected drift by close to 50%.

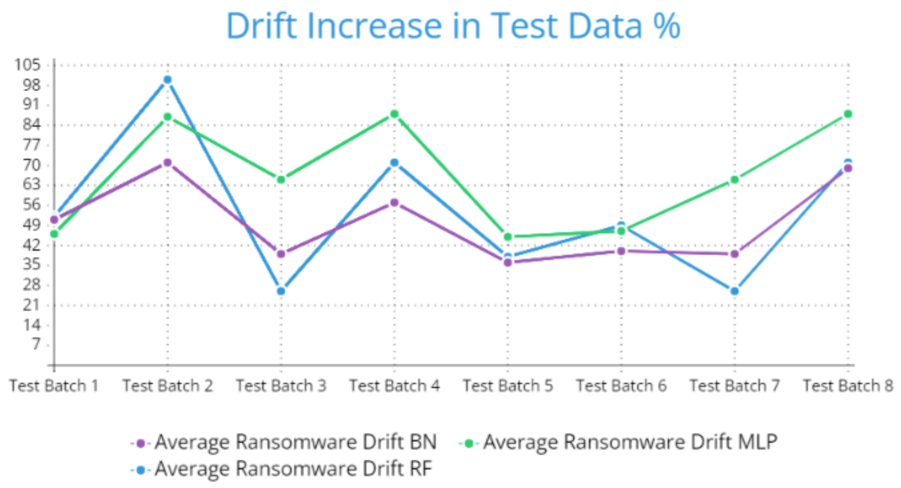

Figure 4.18: Drift Increase in Test Data

Figure 4.18 compares the rate of increase drift calculated by the FeSAD framework of the data in each test batch concerning the data from the base classifier in each test batch. The shifts in drift are not uniform; this reinforces the idea that drift in ransomware behaviour is not uniform. The drift patterns are not drastically different from classifier to classifier, indicating consistent performance with different classifiers. The FeSAD framework incorporates drift from calibration datasets to prepare for these drift changes over time. The primary anomaly in terms of drift is Test Batch 2, where drift levels are high, but the base classifiers can perform well. The drift shown by Test Batch 2 is explainable by a significant reduction in prediction probability for the test samples; however, the classifier still maintains a strong classification late without the intervention of the FeSAD framework. The shifts in drift allowances per test batch are shown in Figure 4.19; this gives a visual indication of how using calibration datasets can help prepare a classifier for concept drift. The drift boundary is significantly increased in all cases because of the equations used to calculate the expected drift. The expected drift takes the drift in the calibration set into account and the rate of increase in drift between the calibration set and training data set.

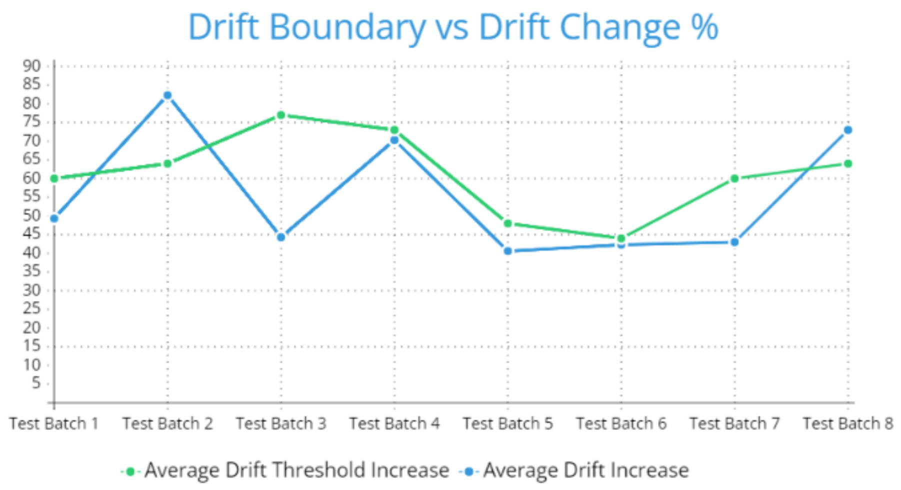

Figure 4.19: Drift Boundary vs Drift Change

The percentage increase in the drift thresholds increases at a higher rate than the actual drift increase because FeSAD takes the drift of the training set, calibration set and the rate of drift increase into account when tuning the drift boundaries. The relationship between the two metrics is relatively consistent except in Test Batch 2, where the test data set shows a significant increase in drift relative to the training set. The classifier's overall performance will not always suffer because of drift; however, the health of the classifier will reflect the level of drift it experiences. Test Batch 2 indicates high levels of drift but the lowest drop-off in detection rates across all three algorithms. The Random Forest experiences extraordinarily low levels of prediction probability and, by extension, drift but still maintains a robust overall performance against Test Batch 2. The data presented in Figure 4.19 shows that the calibration setups used in each test set differ based on the differences between the calibration data and the training data; however, the FeSAD framework can significantly impact detection rates despite the differences between the calibrations it makes per test batch.

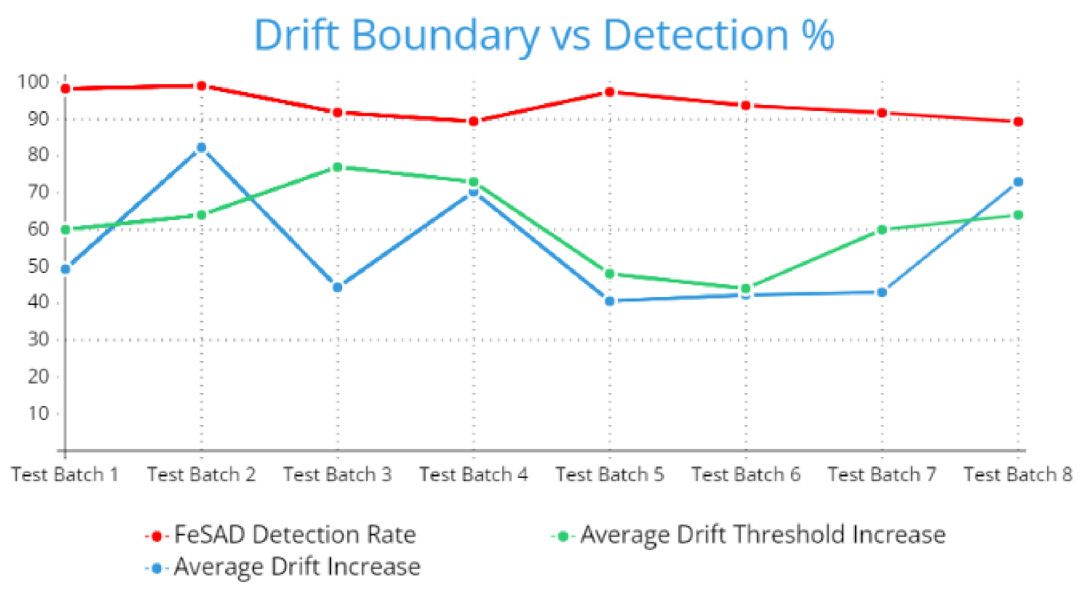

Figure 4.20: Drift Boundary vs Detection

The average detection rates achieved by the FeSAD framework in relation to the shifts in drift and drift boundaries are shown in Figure 4.20, and the detection rate does not suffer adversely when the shifts in the drift boundary are not on the higher end of the scale. The increase in the drift boundaries when drift is high does not necessarily help or harm detection rates, and it appears that the use of calibration sets can effectively create a banding for which ransomware behaviour with API calls and registry activity can fall into.

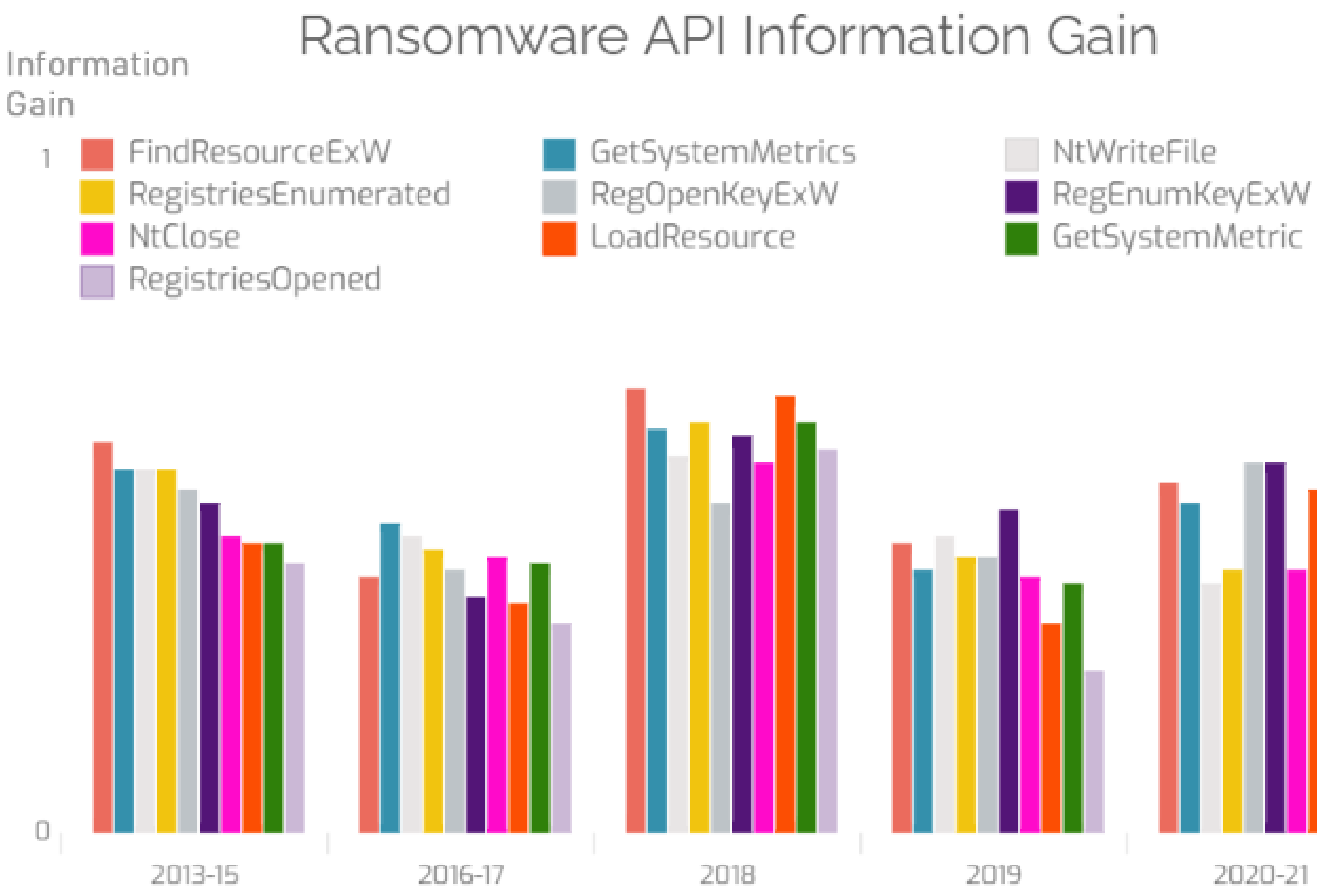

Figure 4.21: Ransomware API Information Gain

Figure 4.21 shows the shifts in information gain for the most prominent API calls used by ransomware as identified by the FeSAD framework. The information gained from each API call shifts over time, although these API calls remain important somewhat consistently in each distribution. The API activity in each ransomware differs slightly, and judging by the differences in information gain for each feature over time shows why a classifier may degrade over time as features it identifies as critical may become less important, and features that were not important have now become critical identifiers between ransomware and benign files. The features identified as most important in the 13-15 data never become entirely irrelevant; however, it is clear that the features experience a shift in importance over time, becoming more and less prominent with new ransomware variants emerging. The experiments conducted took the API call behaviour and registry activity from the first two minutes of execution which would explain the lack of cryptographic or encryption API calls, as most of the samples had not started encrypting by the time the execution period had finished. The statistical aspect of the FeSAD framework compensates for the changes in these features over time as long as the behavioural model does not entirely shift extremely all at once. Based on the performance of the FeSAD framework over different distributions of ransomware over eight years, it can be concluded that the API call behaviour of ransomware does shift; however, it does not shift enough for the FeSAD framework not to be able to maintain high performance rates.

Table 4.12 Navarra University Dataset with Concept Drift

	FeSAD with Random Forest	**FeSAD with Baysian Networks**	**Random Forest**	**Logistic Regression**	**SVM**	**J48 Decision Trees**	**GTB**	**Bayesian Networks**
Tested on Training Distribution	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.4% (0.2%, 0.5%) **Precision:** 0.998 (0.997, 0.999) **Recall:** 0.999 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.1% (0.05%, 0.2%) **Precision:** 0.998 (0.975, 0.984) **Recall:** 0.999 (0.998, 0.999)	**Detection:** 99.5 (99.1%, 99.7%) **FPR:** 0.4% (0.2%, 0.5%) **Precision:** 0.998 (0.975, 0.984) **Recall:** 0.9995 (0.991, 0.997)	**Detection:** 87.0% (85.9%, 88.1%) **FPR:** 10.4% (9.5%, 10.6%) **Precision:** 0.893 (0.888, 0.898) **Recall:** 0.870 (0.859, 0.881)	**Detection:** 86.6% (85.5%, 87.7%) **FPR:** 10.8% (9.3%, 11.1%) **Precision:** 0.889 (0.884, 0.894) **Recall:** 0.866 (0.855, 0.877)	**Detection:** 96.8% (96.2%, 97.3%) **FPR:** 2.8% (2.4%, 2.9%) **Precision:** 0.985 (0.955, 0.961) **Recall:** 0.986 (0.962, 0.973)	**Detection:** 95.3% (94.6%, 95.9%) **FPR:** 3.8% (3.5%, 4.2%) **Precision:** 0.984 (0.981, 0.986) **Recall:** 0.953 (0.946, 0.959)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.2% (0.1%, 0.3%) **Precision:** 0.996 (0.994, 0.998) **Recall:** 0.999 (0.998, 0.999)
Tested on Zero-Day Distribution	**Detection:** 94.3% (88.9%, 98.0%) **FPR:** 0.8% (0.7% − /9%) **Precision:** 0.991 (0.990, 0.993) **Recall:** 0.943 (0.889, 0.980)	**Detection:** 96.7% (92.5%, 99.5%) **FPR:** 0.2% (0.1%, 0.3%) **Precision:** 0.999 (0.998, 0.999) **Recall:** 0.967 (0.925, 0.995)	**Detection:** 89.5% (82.3, 94.4%) **FPR:** 10.5% (9.9%, 11.1%) **Precision:** 0.895 (0.890, 0.900) **Recall:** 0.895 (0.823, 0.944)	**Detection:** 45.6 (36.2%, 55.2%) **FPR:** 54.3% (53.5%, 55.1%) **Precision:** 0.456 (0.448, 0.465) **Recall:** 0.456 (0.362, 0.552)	**Detection:** 50.9% (41.3%, 60.4%) **FPR:** 49.1% (48.3%, 49.9%) **Precision:** 0.509 (0.504, 0.521) **Recall:** 0.509 (0.431, 0.604)	**Detection:** 78.1% (69.4%, 85.3%) **FPR:** 21.9% (21.2%, 22.6%) **Precision:** 0.781 (0.775, 0.788) **Recall:** 0.781 (0.694, 0.853)	**Detection:** 86.0% (78.2%, 91.8%) **FPR:** 0.6% (0.2%, 1%) **Precision:** 0.994 (0.991, 0.995) **Recall:** 0.860 (0.782, 0.981)	**Detection:** 92.1% (85.5%, 96.3%) **FPR:** 0.9% (0.1%, 1.5%) **Precision:** 0.994 (0.993, 0.995) **Recall:** 0.921 (0.855, 0.963)

The results presented in table 4.12 are from a dataset built by researchers from the University of Navarra. Berrueta et al. constructed a ransomware dataset from 70 different ransomware strains from 2015 to 2019 [90]. The features in the dataset are a combination of I/O features and network features, and the dataset is split into training and zero-day sets. The data is obtained by extracting the behavioural data of a file-sharing network under attack from a ransomware infection. The zero-day data represents ransomware behaviour that could imply concept drift. We have used the algorithms in prominent ransomware research to compare with FeSAD and how the detection metrics compare when the classifiers are presented with the zero-day data most likely to contain concept drift. We observe that most of the tested algorithms perform well through 10-fold cross-validation except the Multi-Layer Perceptron, which seems unable to classify ransomware based on the I/O and network features; therefore, we did not include these results. The models are built using the training data and evaluated using 10-fold cross-validation; these models are tested on the zero-day distribution with ransomware and benign data not in the training data to test their performance under concept drift detection. Based on the initial training and test, the highest performers are the Random Forest, J48 Decision Tree, Gradient Tree Boosting and the Bayesian Network; however, every algorithm faces a significant performance drop-off. The Logistic

Regression, and SVM approaches experience a significant drop-off, with detection rates falling to around 50%. The Bayesian Network and Random Forest show the least degradation out of the tested algorithms besides FeSAD. The Random Forest without FeSAD achieves a detection rate of 89.5% on the zero-day data without FeSAD, and FeSAD combined with the Random Forest achieves a detection rate of 94.3%, which is an improvement from 89.5% without FeSAD. We do not include the MLP statistics in table 4.12 because the algorithm appeared unable to distinguish ransomware from benign data. The FeSAD framework achieves a 96.7% detection rate with the Bayesian Network on the zero-day distribution improving on an initial detection rate of 92.1%. The FeSAD framework can improve detection rates in both cases while maintaining low false positive rates. The feature set presented by Barrueta et al. appears to help classify benign data, as the false positives are very low and even easier to maintain with FeSAD. Observing the confidence intervals, the FeSAD framework shows statistically significant results over the Logistic Regression, SVM, and J48 algorithms and would need a higher level of performance to have a statistically significant advantage over the other algorithms; however, given its results with smaller datasets and the larger Navarra dataset, its scalability is plausible.

Table 4.13 Imperial University Dataset with Concept Drift

	FeSAD with Random Forest	**FeSAD with Baysian Networks**	**FeSAD with MLP**	**Random Forest**	**Bayesian Network**	**MLP**
Tested on 2012-13 Data	**Detection:** 92.4%(87.9%, 95.7%)	**Detection:** 89.6%(84.5%, 93.4%)	**Detection:** 90.6%(85.6%, 94.1%)	**Detection:** 88.0%(82.7%, 92.2%)	**Detection:** 85.9%(80.4%, 90.4%)	**Detection:** 85.0%(79.2%, 89.6%)
	FPR: 7.8%(5.3%, 10.8%)	**FPR:** 6.4%(4.3%, 9.4%)	**FPR:** 4.9%(3.1%, 7.6%)	**FPR:** 9.6%(6.8%, 12.8%)	**FPR:** 12.0%(8.9%, 15.6%)	**FPR:** 14.0%(10.8%, 17.8%)
	Precision: 0.942(0.918, 0.959)	**Precision:** 0.943(0.927, 0.964)	**Precision:** 0.950(0.929, 0.966)	**Precision:** 0.943(0.922, 0.960)	**Precision:** 0.877(0.848, 0.904)	**Precision:** 0.878(0.849, 0.903)
	Recall: 0.924(0.879, 0.957)	**Recall:** 0.896(0.845, 0.934)	**Recall:** 0.906(0.856, 0.941)	**Recall:** 0.880(0.827, 0.922)	**Recall:** 0.859(0.804, 0.904)	**Recall:** 0.850(0.792, 0.896)
Tested on 2014 Data	**Detection:** 89.6(84.5%, 94.1%)	**Detection:** 87.5%(82.1%, 91.7%)	**Detection:** 89.5%(84.5%, 93.4%)	**Detection:** 82.5%(76.5%, 87.5%)	**Detection:** 77.5%(71.1%, 83.1%)	**Detection:** 82.5%(76.5%, 87.5%)
	FPR: 6.9%(4.7%, 10.0%)	**FPR:** 9.4%(6.8%, 12.8%)	**FPR:** 10.5%(7.7%, 13.9%)%	**FPR:** 12.4%(9.4%, 12.1%)	**FPR:** 11.4%(8.5%, 15.0%)	**FPR:** 17.1%(13.4%, 21.1%)
	Precision: 0.960(0.941, 0.974)	**Precision:** 0.953(0.929, 0.967)	**Precision:** 0.899(0.871, 0.921)	**Precision:** 0.911(0.888, 0.934)	**Precision:** 0.837(0.805, 0.866)	**Precision:** 0.837(0.805, 0.866)
	Recall: 0.896(0.845, 0.941)	**Recall:** 0.875(0.821, 0.917)	**Recall:** 0.895(0.845, 0.934)	**Recall:** 0.825(0.765, 0.875)	**Recall:** 0.775(0.711, 0.831)	**Recall:** 0.825(0.765, 0.875)
Tested On 2015 Data	**Detection:** 85.3%(79.1%, 89.9%)	**Detection:** 89.8%(84.1%, 93.6%)	**Detection:** 81.0%(72.5%, 86.2%)	**Detection:** 70.4%(63.1%, 76.9%)	**Detection:** 70.4%(63.1%, 76.9%)	**Detection:** 64.8%(57.4%, 71.8%)
	FPR: 1.5%(0.5%, 1.5%)	**FPR:** 3.3(1.7%, 5.4%)	**FPR:** 6.7%(4.5%, 9.7%)	**FPR:** 17.6%(13.5%, 21.0%)	**FPR:** 17.6%(13.5%, 21.0%)	**FPR:** 24.0%(19.9%, 28.4%)
	Precision: 0.982(0.966, 0.999)	**Precision:** 0.976(0.959, 0.987)	**Precision:** 0.923(0.899, 0.945)	**Precision:** 0.800(0.766, 0.832)	**Precision:** 0.800(0.766, 0.832)	**Precision:** 0.729(0.692, 0.766)
	Recall: 0.853(0.791, 0.899)	**Recall:** 0.898(0.898, 0.841)	**Recall:** 0.810(0.725, 0.862)	**Recall:** 0.704(0.631, 0.769)	**Recall:** 0.704(0.631, 0.769)	**Recall:** 0.648(0.574, 0.718)

The results in Table 4.13 were obtained using the FeSAD framework with a dataset produced by Sgandurra et al. The imperial dataset contained data from 480 ransomware samples and over

100 benign files. The ransomware and benign data in this dataset were captured from ransomware and benign files between 2012 to 2015; therefore, we decided to segregate the data to test the capabilities of FeSAD and its concept drift adaption methods. The dataset was a dump of raw data of cuckoo sandbox executions and therefore contained many data which needed to be cleaned and organised to be considered a viable dataset. The features were significantly reduced into API call types, including network-based API calls and registry activity. The classifiers are trained on 2012 and 2013 data and tested on 2014 and 2015 data. The classifiers show an average reduction in the detection rate of 5.2% when tested on 2014 data; however, the FeSAD framework can reduce this detection loss to 2.1%. The classifier shows an average reduction rate in the detection rate of 19.6% when exposed to the 2015 data; however, the FeSAD framework reduces this detection loss to 8.2% when tested on the 2015 data. This dataset appeared to pose a challenge for the classifiers to achieve detection rates above the 90% mark; therefore, the FeSAD framework could not necessarily achieve the higher standards it achieved in the two previous datasets. The conversion of over 30,000 nominal features into continuous features could result in some accuracy loss; however, the dataset was not usable in its provided form. The FeSAD framework still shows the ability to compensate for the loss in performance because of concept drift, and this is presented by aiding to preserve the detection rate while significantly reducing the number of false positives. Considering confidence intervals, the FeSAD framework shows statistically significant results when tested on the 2015 data; however, it does not show enough performance gain on the 2014 data to achieve this.

4.6.1 FeSAD Experiments with Alternate Similarity Metrics

We observe the performance of the FeSAD framework with the HEOM metric combined with three prominent machine learning algorithms is promising. As the FeSAD framework is designed to use interchangeable components, we decided to test its flexibility and performance when working with different similarity metrics. We evaluate the performance of each algorithm combined with the respective similarity metric and observe the degradation the classifier experiences when used in the FeSAD framework. These experiments also track the reduction in observed abnormal samples when calibrated to expect drift. For the FeSAD framework to be truly flexible, it must work with different similarity metrics, as other metrics may provide better solutions for various data types.

Table 4.14 FeSAD Performance with Cosine Similarity

Test Batch Number	**Setup**	**Performance with Random Forest**	**Performance with Bayesian Network**	**Performance With MLP**
1	**Train:** 2013-15 **Test:** 2016 **Calibration:** 2017-18	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 3.9%(2.3%, 6.4%) **Precision:** 0.960(0.934, 0.979) **Recall:** 0.969(0.893, 0.996) **Unexpected Drift Reduction:** 52%	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 3.2%(1.6%, 5.9%) **Precision:** 0.968(0.895, 0.997) **Recall:** 0.969(0.893, 0.996) **Unexpected Drift Reduction:** 53%	**Detection:** 100.0%(94.5%, 100%) **FPR:** 2.3%(1.0%, 5.2%) **Precision:** 0.977(0.954, 0.989) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 55%
2	**Train:** 2013-15 **Test:** 2018 **Calibration:** 2016-17	**Detection:** 100.0%(94.5%, 100%) **FPR:** 2.2%(0.9%, 5.1%) **Precision:** 0.975(0.952, 0.987) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 44%	**Detection:** 100.0%(94.5%, 100%) **FPR:** 2.2%(0.9%, 5.1%) **Precision:** 0.975(0.952, 0.987) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 41%	**Detection:** 100.0%(94.5%, 100%) **FPR:** 0%(0%, 0.1%) **Precision:** 1.0(0.989, 1.0) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 42%
3	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2017-18	**Detection:** 93.8%(89.3%, 99.6%) **FPR:** 2.4%(0.9%, 5.1%) **Precision:** 0.975(0.941, 0.990) **Recall:** 0.938(0.893, 0.996) **Unexpected Drift Reduction:** 51%	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 4.6%(2.5%, 7.9%) **Precision:** 0.953(0.924, 0.972) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 50%	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 3.4%(1.9%, 6.3%) **Precision:** 0.964(0.939, 0.981) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 46%
4	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2018	**Detection:** 93.8%(89.3%, 99.6%) **FPR:** 7.7%(4.8%, 11.4%) **Precision:** 0.924(0.889, 0.950) **Recall:** 0.938(0.893, 0.996) **Unexpected Drift Reduction:** 60%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 6.2%(3.6%, 9.7%) **Precision:** 0.936(0.903, 0.959) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 46%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 5.0%(2.8%, 8.4%) **Precision:** 0.947(0.917, 0.968) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 45%
5	**Train:** 2013-18 **Test:** 2019 **Calibration:** 2020-21	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 4.6%(2.5%, 5.1%) **Precision:** 0.952(0.924, 0.972) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 50%	**Detection:** 96.8%(89.3%, 99.6%) **FPR:** 6.4%(3.9%, 10.1%) **Precision:** 0.937(0.907, 0.961) **Recall:** 0.968(0.893, 0.996) **Unexpected Drift Reduction:** 43%	**Detection:** 96.8%(89.3%, 99.6%) **FPR:** 6.9%(4.3%, 10.5%) **Precision:** 0.933(0.903, 0.959) **Recall:** 0.968(0.893, 0.996) **Unexpected Drift Reduction:** 41%
6	**Train:** 2013-18 **Test:** 2020-21 **Calibration:** 2019	**Detection:** 98.5%(91.7%, 99.9%) **FPR:** 3.8%(2.1%, 7.0%) **Precision:** 0.962(0.934, 0.979) **Recall:** 0.985(0.917, 0.999) **Unexpected Drift Reduction:** 48%	**Detection:** 90.8%(80.9%, 97.5%) **FPR:** 6.9%(4.3%, 10.5%) **Precision:** 0.928(0.897, 0.954) **Recall:** 0.908(0.809, 0.975) **Unexpected Drift Reduction:** 45%	**Detection:** 92.3%(83.0%, 97.5%)% **FPR:** 5.1%(2.7%, 8.2%) **Precision:** 0.947(0.917, 0.967) **Recall:** 0.925(0.830, 0.975) **Unexpected Drift Reduction:** 52%
7	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2016-17	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 6.3%(3.9%, 10.1%) **Precision:** 0.934() **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 51%	**Detection:** 87.7%(77.2%, 94.5%) **FPR:** 8.3%(5.1%, 11.9%) **Precision:** 0.914(0.879, 0.942) **Recall:** 0.887(0.772, 0.945) **Unexpected Drift Reduction:** 48%	**Detection:** 92.3(83.0%, 97.5%) **FPR:** 6.3%(3.7%, 9.9%) **Precision:** 0.936(0.903, 0.959) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 45%
8	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2016-17	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 6.9%(4.2%, 10.7%) **Precision:** 0.907(0.873, 0.937) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 50%	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 7.5(4.8%, 11.4%) **Precision:** 0.920(0.887, 0.947) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 49%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 6.9%(4.3%, 10.5%) **Precision:** 0.928(0.897, 0.954) **Recall:** 0.901(0.809, 0.965) **Unexpected Drift Reduction:** 47%

Table 4.14 shows the performance of the FeSAD framework using the Cosine Similarity instead of the HEOM metric, and the results are promising. The first train batch maintains an average detection rate of 98.1% across the three classifiers compared with the 98.0% average detection with the HEOM metric used. The average false positive rate with the Cosine similarity is 3.13% compared to 2.67% for the HEOM metric. The average detection rate for test batch 2 is 100% with an average false positive rate of 1.47%, whereas the HEOM metric achieves an average detection rate of 99.1% with an average false positive rate of 1.47%. Test batch 3 achieves an average detection rate of 92.7% and a false positive rate, and an average false positive rate of 3.47% compared to HEOM, achieving a detection rate of 91.8% and a false positive rate of 6.63%. Test batch 3 shows a drop-off of performance in HEOM in terms of false positives, whereas both similarity metrics appear similar in terms of detection. Test batch 4 sees the detection rate average 91.53% with a false positive rate of 6.3%, whereas, with HEOM, the false positive rate is 8.63% and detection rate is 89.43%. Test batch 5 sees an average detection rate of 95.57% and a false positive rate of 6%, whereas with HEOM, the average false-positive rate is 4.03% and a detection average of 96.8%. Test batch 6 achieves an average detection rate of 93.8% and a false positive rate of 5.27%, whereas the HEOM achieves an average detection rate of 93.73% and a false positive rate of 6.13%. Test batch 7 sees an average detection rate of 90.33% and a false positive rate of 6.97%, whereas, with HEOM, the average detection rate is 91.67% and a false positive rate of 8.03%. Test batch 8 sees an average detection rate of 89.76% and 7.1%, whereas, with HEOM, the average detection rate is 87.7% and an average false positive rate of 9.03%. Considering confidence intervals there is no statistical significant in the performance difference between the algorithms when used with FeSAD and Cosine similarity as the three algorithms perform very closely.

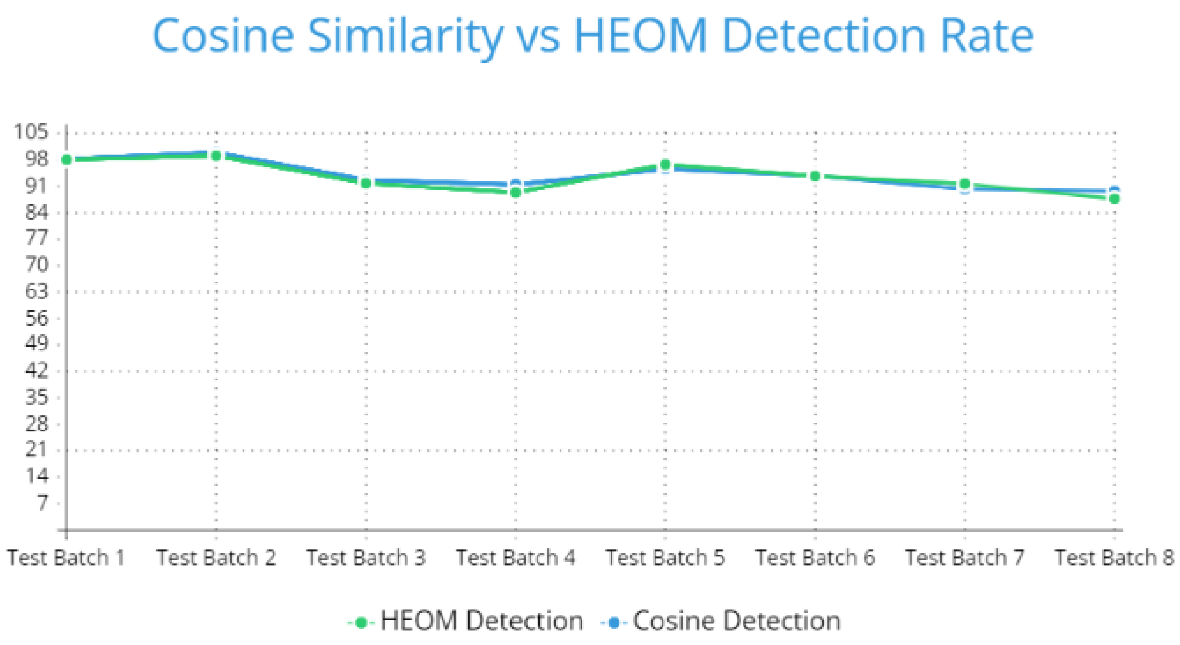

Figure 4.22: Cosine vs HEOM Detectionk

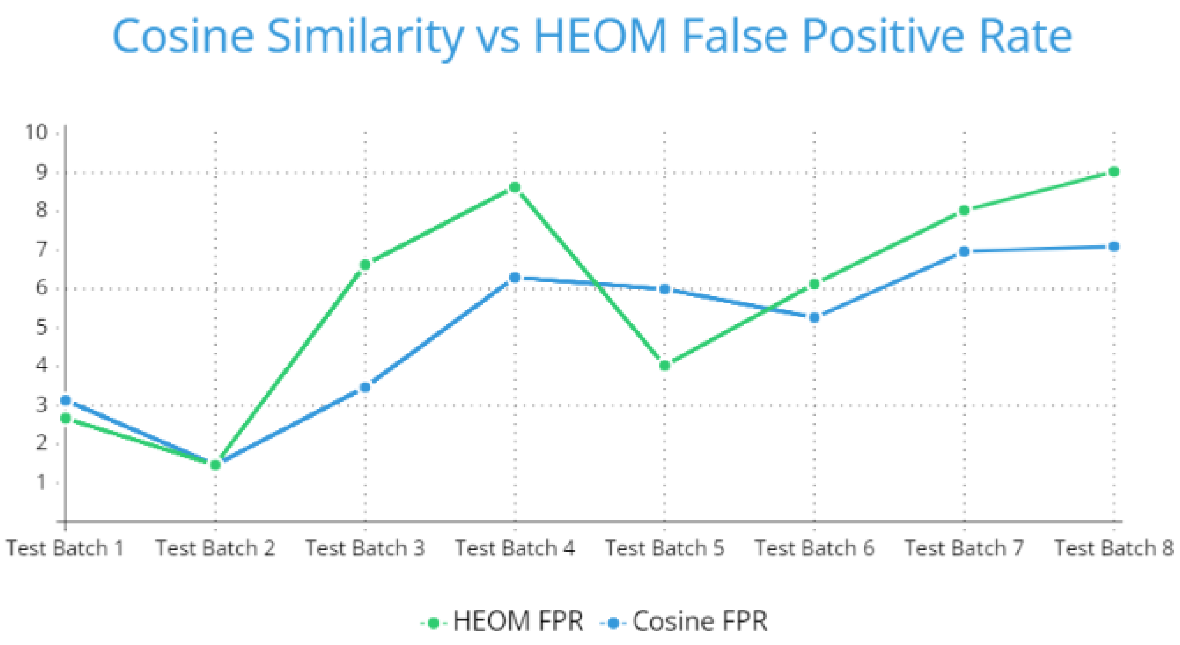

Figure 4.23: Cosine vs HEOM False Positive Rate

Analysing Figures 4.22 and 4.23, both similarity metrics are not far apart in terms of performance, with the HEOM metric displaying equal or better performance when the calibration set is not far from the test data. In terms of detection, there is little to split the two similarity metrics; however, it can be argued that the cosine similarity may provide greater longevity as it can achieve better detection rates in the Test Batches that stretch an old classifier which would be test batches 3, 4 and 8 whereas the HEOM metric provides higher detection rates when the training and calibration sets are close together, and the test set is not far from either training or calibration set.

In terms of false positives, it is clear that the cosine similarity has the upper hand as it achieves a lower false positive rate in every test batch except test batch 1 and test batch 5. The false positive rates being lower with cosine similarity than the HEOM metric is potentially due to the way a vector angle can identify the difference between samples as opposed to the modified Euclidean distance used by HEOM. Regarding detection, the HEOM metric and Cosine similarity are on par. The HEOM metric provides better performance in situations likelier to happen in a real-world setting where the training and calibration sets won't be years behind test data.

Table 4.15 FeSAD Performance with Canberra Metric

Test Batch Number	**Setup**	**Performance with Random Forest**	**Performance with Bayesian Network**	**Performance With MLP**
1	**Train:** 2013-15 **Test:** 2016 **Calibration:** 2017-18	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 5.0%(2.8%, 8.4%) **Precision:** 0.947(0.918, 0.968) **Recall:** 0.897(0.790, 0.956) **Unexpected Drift Reduction:** 48%	**Detection:** 90.8(80.9%, 96.5%) **FPR:** 3.2%(1.6%, 5.9%) **Precision:** 0.966(0.939, 0.981) **Recall:** 0.932(0.809, 0.965) **Unexpected Drift Reduction:** 48%	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 3.2%(1.6%, 5.9%) **Precision:** 0.966(0.939, 0.981) **Recall:** 0.892() **Unexpected Drift Reduction:** 50%
2	**Train:** 2013-15 **Test:** 2018 **Calibration:** 2016-17	**Detection:** 98.5%(91.7%, 99.9%)% **FPR:** 3.6%(1.7%, 6.1%) **Precision:** 0.955(0.929, 0.975) **Recall:** 0.985(0.917, 0.999) **Unexpected Drift Reduction:** 40%	**Detection:** 99.0%(91.8%, 99.9%) **FPR:** 6.5%(5.1%, 11.9%) **Precision:** 0.938(0.908, 0.969) **Recall:** 0.991(0.918, 0.999) **Unexpected Drift Reduction:** 40%	**Detection:** 99.0%(91.8%, 99.9%) **FPR:** 4.5%(2.4%, 7.9%) **Precision:** 0.962(0.936, 0.979) **Recall:** 0.991(0.918, 0.999) **Unexpected Drift Reduction:** 42%
3	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2017-18	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 2.4%(1.2%, 5.4%) **Precision:** 0.973(0.950, 0.987) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 50%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 5.7%(3.4%, 9.2%) **Precision:** 0.941(0.912, 0.965) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 48%	**Detection:** 90.8%(80.90%, 96.5%) **FPR:** 7.2%(3.4%, 9.3%) **Precision:** 0.926(0.901, 0.949) **Recall:** 0.903(0.809, 0.965) **Unexpected Drift Reduction:** 46%
4	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2018	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 9.2%(5.9%, 13.1%) **Precision:** 0.909(0.873, 0.937) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 57%	**Detection:** 90.8(80.9%, 96.5%)% **FPR:** 7.9%(5.1%, 11.9%) **Precision:** 0.920(0.887, 0.947) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 46%	**Detection:** 89.6%(79.0%, 95.6%) **FPR:** 7.2%(4.2%, 10.6%) **Precision:** 0.926(0.893, 0.952) **Recall:** 0.896(0.790, 0.956) **Unexpected Drift Reduction:** 44%
5	**Train:** 2013-18 **Test:** 2019 **Calibration:** 2020-21	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 4.7%(2.3%, 5.0%) **Precision:** 0.950 **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 48%	**Detection:** 93.5%(89.0%, 99.3%) **FPR:** 8.4%(5.4%, 12.3%) **Precision:** 0.917(0.883, 0.945) **Recall:** 0.935(0.890, 0.993) **Unexpected Drift Reduction:** 40%	**Detection:** 93.5%(89.0%, 99.3%) **FPR:** 6.9%(4.3%, 10.5%) **Precision:** 0.931(0.902, 0.957) **Recall:** 0.935(0.890, 0.993) **Unexpected Drift Reduction:** 42%
6	**Train:** 2013-18 **Test:** 2020-21 **Calibration:** 2019	**Detection:** 93.5%(89.0%, 99.3%) **FPR:** 3.8%(1.9%, 6.4%) **Precision:** 0.961(0.955, 0.979) **Recall:** 0.935(0.890, 0.993) **Unexpected Drift Reduction:** 45%	**Detection:** 93.5%(89.0%, 99.3%) **FPR:** 9.2%(6.6%, 14.0%) **Precision:** 0.910(0.876, 0.937) **Recall:** 0.935(0.890, 0.993) **Unexpected Drift Reduction:** 42%	**Detection:** 93.5%(89.0%, 99.3%) **FPR:**4.2%(2.2%, 7.5%) **Precision:** 0.956(0.928, 0.975) **Recall:** 0.935(0.890, 0.993) **Unexpected Drift Reduction:** 42%
7	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2016-17	**Detection:** 86.2%(75.3%, 93.5%) **FPR:** 5.2%(2.9%, 8.7%) **Precision:** 0.943(0.914, 0.966) **Recall:** 0.862(0.753, 0.935) **Unexpected Drift Reduction:** 49%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 8.3%(5.4%, 12.3%) **Precision:** 0.916(0.879, 0.942) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 45%	**Detection:** 86.2%(75.3%, 93.5%) **FPR:** 8.9%(5.9%, 12.8%) **Precision:** 0.906(0.870, 0.935) **Recall:** 0.862(0.753, 0.935) **Unexpected Drift Reduction:** 42%
8	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2016-17	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 6.9%(4.3%, 10.5%) **Precision:** 0.907(0.870, 0.935) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 46%	**Detection:** 87.7%(77.2%, 94.5%) **FPR:** 7.5%(4.8%, 11.4%) **Precision:** 0.925 **Recall:** 0.877(0.772, 0.945) **Unexpected Drift Reduction:** 46%	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 7.5%(4.8%, 11.4%) **Precision:** 0.923(0.890, 0.950) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 45%

Table 4.15 shows the performance of the FeSAD framework using the Canberra distance metric instead of the HEOM metric. The first train batch maintains an average detection rate of 89.87% across the three classifiers compared with the 98.0% average detection with the HEOM metric used. The average false positive rate with the Canberra metric is 3.8% compared to 2.67% false positive rate achieved by the HEOM metric. The average detection rate for test batch 2 is 98.9% with an average false positive rate of 4.87%, whereas the HEOM metric achieves an average detection rate of 99.1% with an average false positive rate of 1.47%. Test batch 3, with the Canberra metric achieves an average detection rate of 90.07% and a false positive rate, and an average false positive rate of 5.1% compared to HEOM, which achieve a detection rate of 91.8% and a false positive rate of 6.63%. Test batch 3 shows a drop-off of performance in HEOM in terms of false positives, whereas both similarity metrics appear similar in terms of detection despite HEOM achieving a higher detection rate. Test batch 4 sees the detection rate average 90.73% with a false positive rate of 8.10%, whereas, with HEOM, the false positive rate is 8.63% and detection rate is 89.43%. Test batch 5 sees an average detection rate of 92.43% and a false positive rate of 6.67%, whereas with the HEOM metric, the average false-positive rate is 4.03% and a detection average of 96.8%. Test batch 6 achieves an average detection rate of 93.2% and a false positive rate of 7.0%, whereas the HEOM achieves an average detection rate of 93.73% and a false positive rate of 6.13%. Test batch 7 sees an average detection rate of 87.5% and a false positive rate of 7.47%, whereas, the HEOM metric achieves an average detection rate of 91.67% and a false positive rate of 8.03%. Test batch 8 sees an average detection rate of 88.35% and false positive of 7.3%, whereas, with HEOM, the average detection rate is 87.7% and an average false positive rate of 9.03%. Considering the confidence intervals, the performance differences between the three algorithms using the Canberra Metric are not statistically significant. The FeSAD framework is able to achieve similar results with all three algorithms using the Canberra Metric.

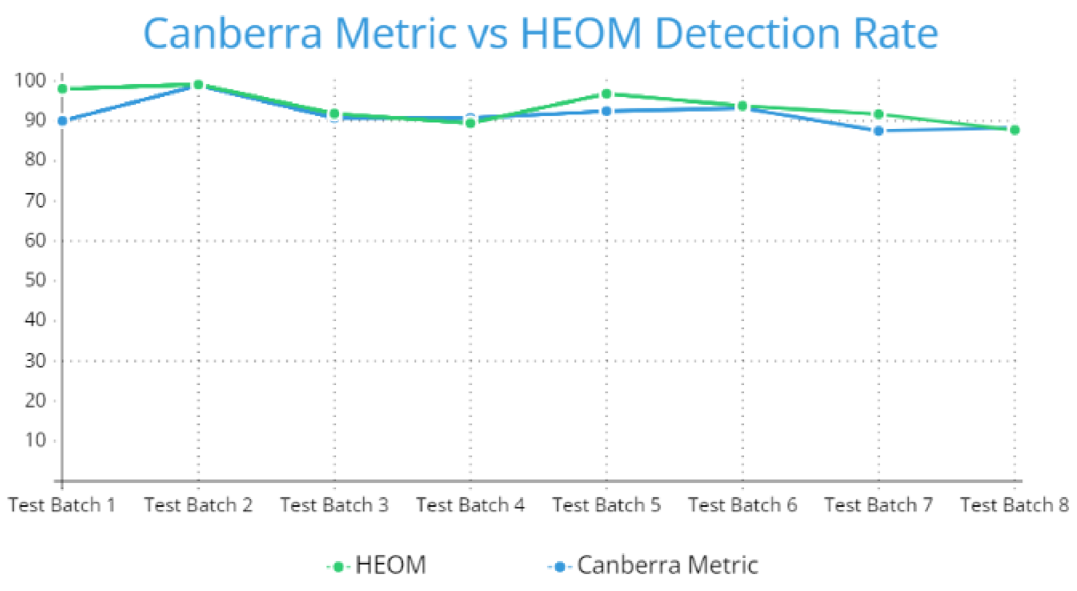

Figure 4.24: Canberra Metric vs HEOM Detection

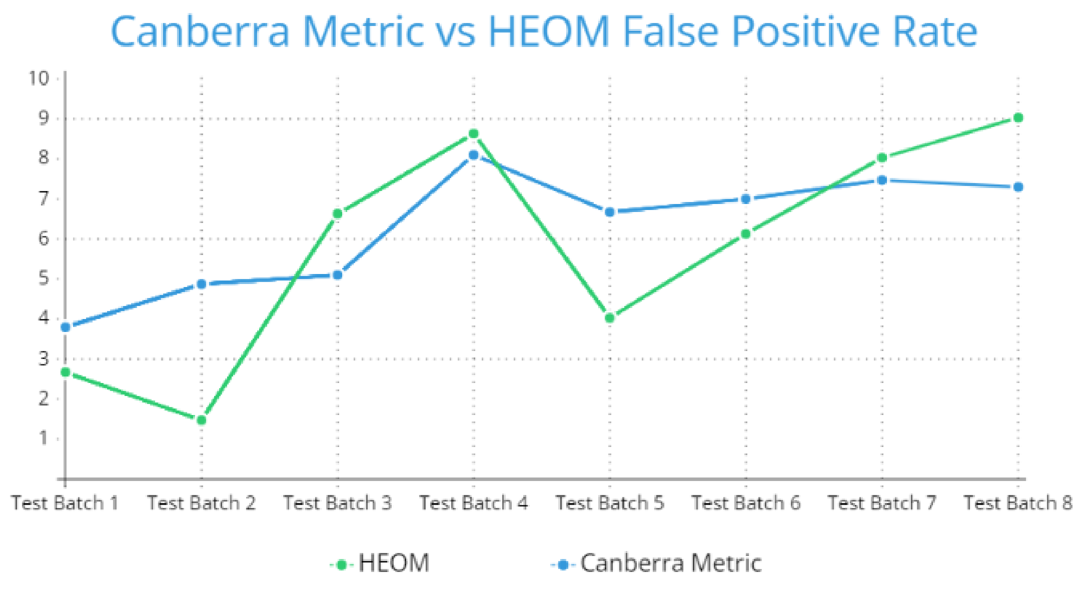

Figure 4.25: Canberra Metric vs HEOM False Positive Rate

Figure 4.24 and Figure 4.25 compare the performance of the Canberra metric with the HEOM metric. The Canberra metric tends to achieve slightly reduced detection rates compared to the HEOM metric. The false positives with the Canberra metric are comparable to HEOM false positive rates, with the Canberra metric achieving lower false positives than the HEOM metric in test batches 3, 4, 6 and 7. The HEOM metric achieves generally higher detection rates than the Canberra metric, and this may be due to the HEOM metric doing a better job of normalising each feature, whereas the Canberra metric doesn't equalise the difference in numerical values for features in the same way the HEOM metric does. Overall, the Canberra metric achieves comparable performance to the HEOM metric, reducing misclassifications. The patterns of reduction in detection rate and increase in false positives appear very similar between the Canberra metric and the HEOM metric, which suggests the performance difference is down to how well the metric can differentiate different samples from different classes, granted that the classifier performance remains the same. The Canberra metric could likely achieve the same detection levels as the HEOM metric with modifications that would address the lack of normalisation from feature to feature.

Table 4.16 FeSAD Performance with Pearson Correlation

Test Batch Number	Setup	Performance with Random Forest	Performance with Bayesian Network	Performance With MLP
1	**Train:** 2013-15 **Test:** 2016 **Calibration:** 2017-18	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 3.9%(2.0%, 7.0%) **Precision:** 0.959(0.932, 0.977) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 54%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 2.2%(0.8%, 4.0%) **Precision:** 0.976(0.954, 0.990) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 53%	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 2.2%(1.1%, 4.2%) **Precision:** 0.978(0.958, 0.992) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 54%
2	**Train:** 2013-15 **Test:** 2018 **Calibration:** 2016-17	**Detection:** 100.0%(94.5%, 100%) **FPR:** 1.0%(0.8%, 2%) **Precision:** 0.990(0.974, 0.998) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 48%	**Detection:** 100.0%(94.5%, 100%) **FPR:** 2.2%(1.1%, 4.2%) **Precision:** 0.978(0.958, 0.991) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 44%	**Detection:** 100.0%(94.5%, 100%) **FPR:** 3.0%(1.2%, 5.7%) **Precision:** 0.970(0.947, 0.986) **Recall:** 1.0(0.945, 1.0) **Unexpected Drift Reduction:** 44%
3	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2017-18	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 5.4%(3.1%, 9.0%) **Precision:** 0.943(0.914, 0.966) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 52%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 2.2%(1.1%, 4.2%) **Precision:** 0.975(0.954, 0.990) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 51%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 1.2%(0.2%, 3.2%) **Precision:** 0.987(0.970, 0.997) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 50%
4	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2018	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 3.2%(1.2%, 5.7%)% **Precision:** 0.967(0.943, 0.984) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 63%	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 4.1%(2.0%, 7.0%) **Precision:** 0.977(0.954, 0.990) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 49%	**Detection:** 92.3%(83.0%, 97.5%) **FPR:** 5.0%(2.8%, 8.7%) **Precision:** 0.949(0.921, 0.970) **Recall:** 0.923(0.830, 0.975) **Unexpected Drift Reduction:** 55%
5	**Train:** 2013-18 **Test:** 2019 **Calibration:** 2020-21	**Detection:** 93.5%(89.0%, 99.3%) **FPR:** 1.4%(0.4%, 3.6%) **Precision:** 0.985(0.966, 0.995) **Recall:** 0.935(0.89, 0.993) **Unexpected Drift Reduction:** 52%	**Detection:** 96.9%(89.3%, 99.6%) **FPR:** 3.2%(1.6%, 5.9%) **Precision:** 0.966(0.892, 0.994) **Recall:** 0.969(0.893, 0.996) **Unexpected Drift Reduction:** 47%	**Detection:** 93.5%(89.0%, 99.3%) **FPR:** 2.5%(1.2%, 5.5%) **Precision:** 0.974(0.950, 0.988) **Recall:** 0.935(0.89, 0.993) **Unexpected Drift Reduction:** 45%
6	**Train:** 2013-18 **Test:** 2020-21 **Calibration:** 2019	**Detection:** 98.5%(91.7%, 99.9%) **FPR:** 3.8%(2.2%, 6.3%) **Precision:** 0.962(0.938, 0.980) **Recall:** 0.987(0.917, 0.999) **Unexpected Drift Reduction:** 51%	**Detection:** 96.8%(89.3%, 99.6%)% **FPR:** 2.4%(0.9%, 5.1%) **Precision:** 0.974(0.930, 0.990) **Recall:** 0.968(0.893, 0.996) **Unexpected Drift Reduction:** 50%	**Detection:** 93.5%(89.0%, 99.3%) **FPR:** 1.6%(0.5%, 4.1%) **Precision:** 0.983(0.962, 0.993) **Recall:** 0.935(0.890, 0.993) **Unexpected Drift Reduction:** 51%
7	**Train:** 2013-15 **Test:** 2019 **Calibration:** 2016-17	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 3.3%(1.4%, 5.7%) **Precision:** 0.965(0.948, 0.986) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 53%	**Detection:** 90.8%(80.9%, 96.5%) **FPR:** 2.5%(1.2%, 5.5%) **Precision:** 0.973(0.950, 0.988) **Recall:** 0.908(0.809, 0.965) **Unexpected Drift Reduction:** 50%	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 3.0%(1.4%, 5.7%) **Precision:** 0.968(0.942, 0.984) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 49%
8	**Train:** 2013-15 **Test:** 2020-21 **Calibration:** 2016-17	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 8.9%(5.9%, 13.1%) **Precision:** 0.928(0.897, 0.954) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 52%	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 4.5%(2.4%, 7.9%) **Precision:** 0.973(0.950, 0.988) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 54%	**Detection:** 89.2%(79.0%, 95.6%) **FPR:** 5.6%(3.4%, 9.3%) **Precision:** 0.982(0.962, 0.993) **Recall:** 0.892(0.790, 0.956) **Unexpected Drift Reduction:** 49%

The results covered in table 4.16 show the performance of the FeSAD framework with the Pearson correlation used instead of the HEOM metric and the observations are promising. The pearson correlation achieves competitive results with the HEOM metric and in some test batches surpasses the performance of the HEOM metric. The first train batch maintains an average detection rate of 91.6% across the three classifiers compared with the 98.0% average detection with the HEOM metric used. The average false positive rate with the Pearson correlation is 2.7% compared to 2.67% false positive rate achieved by the HEOM metric. The average detection rate for test batch 2 is 100% with an average false positive rate of 2.07%, whereas the HEOM metric achieves an average detection rate of 99.1% with an average false positive rate of 1.47%. Test batch 3, with the Pearson correlation achieves an average detection rate of 90.3% and an average false positive rate of 2.93% compared to HEOM, which achieve a detection rate of 91.8% and a false positive rate of 6.63%. Test batch 4 sees the detection rate average 94.23% with a false positive rate of 4.10%, whereas, with HEOM, the false positive rate is 8.63% and detection rate is 89.43%. Test batch 5 sees an average detection rate of 94.23% and a false positive rate of 2.43%, whereas with the HEOM metric, the average false-positive rate is 4.03% and a detection average of 96.8%. Test batch 6 achieves an average detection rate of 96.3% and a false positive rate of 7.0%, whereas the HEOM achieves an average detection rate of 93.73% and a false positive rate of 6.13%. Test batch 7 sees an average detection rate of 90.1% and a false positive rate of 2.93%, whereas, the HEOM metric achieves an average detection rate of 91.67% and a false positive rate of 8.03%. Test batch 8 sees an average detection rate of 89.67% and false positive of 7.3%, whereas, with HEOM, the average detection rate is 87.7% and an average false positive rate of 6.33%. Considering the confidence intervals, the performance differences between the three algorithms while using the Pearson Correlation are not statistically significant.

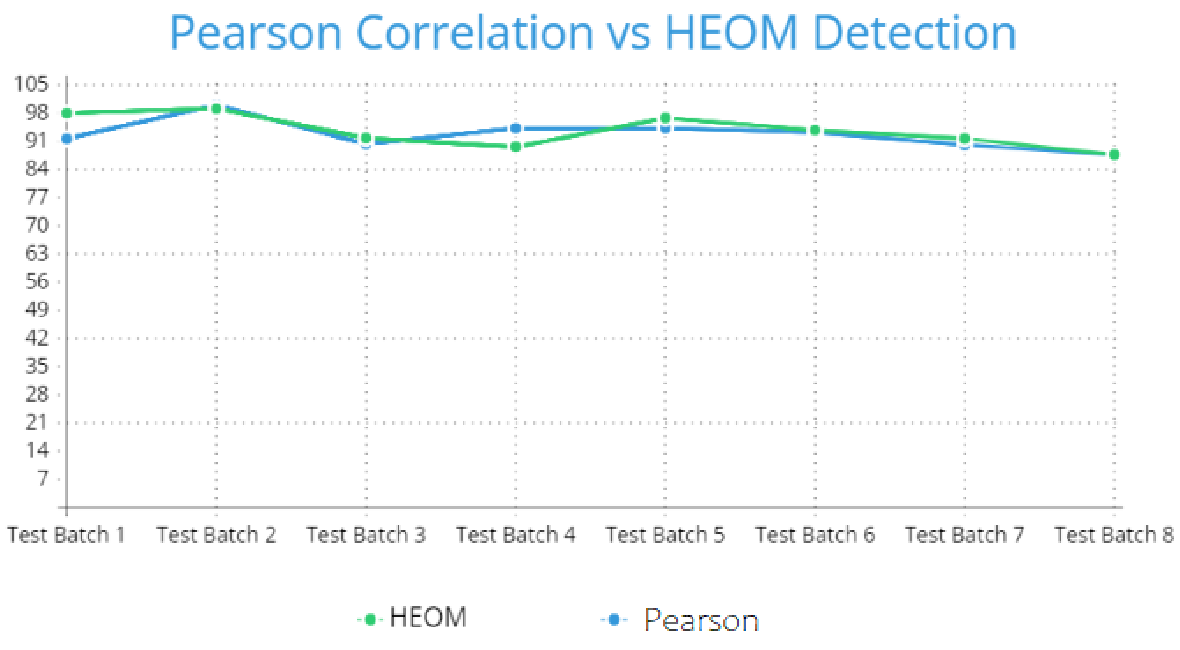

Figure 4.26: Pearson Correlation vs HEOM Detection Rate

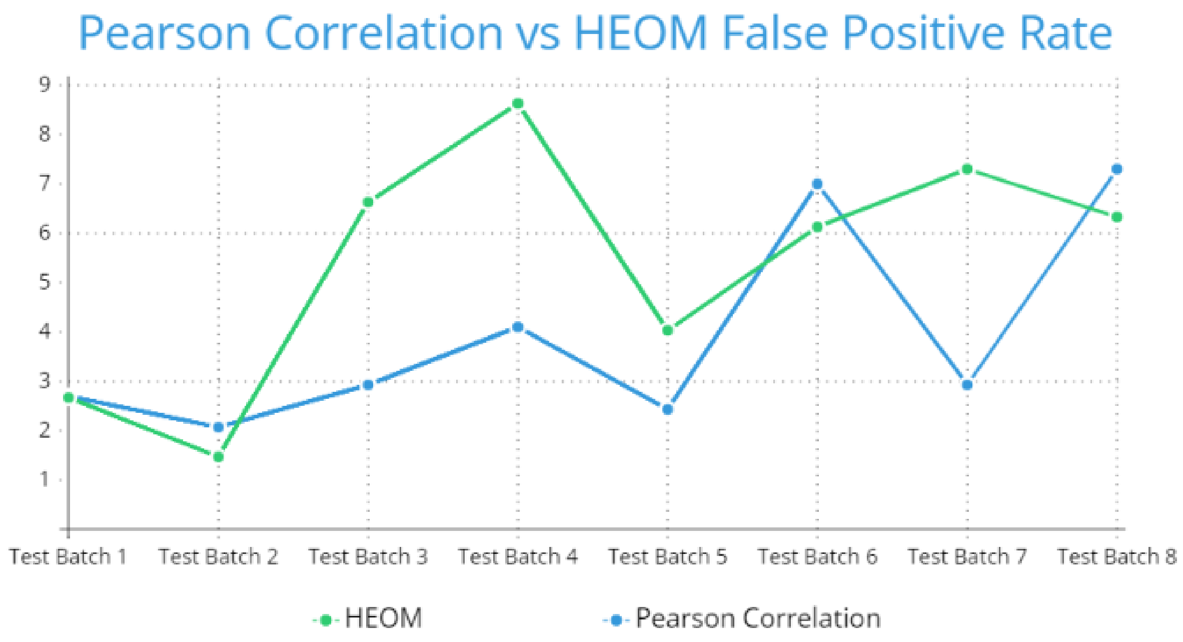

Figure 4.27: Pearson Correlation vs HEOM False Positive Rate

The performance of the Pearson Correlation compared to the HEOM metric is strong. Figure 4.26 shows the FeSAD framework with Pearson Correlation provides better detection performance than the HEOM metric in Test batch 4, which suggests it may have greater longevity when recalibrated with newer data. The HEOM metric generally achieves marginally higher detection rates; however, the difference is marginal, and the FeSAD framework achieves more consistent results with the Pearson Correlation. The false positive rates achieve using the Pearson correlation are less erratic than with the HEOM metric; however, in terms of percentages, they are on par with each other with the patterns of increase in false positive rates over time. The Pearson correlation can consistently improve the detection rate in the same way HEOM does, with its effectiveness particularly clear in the Test Batches, which involve long periods without retraining.

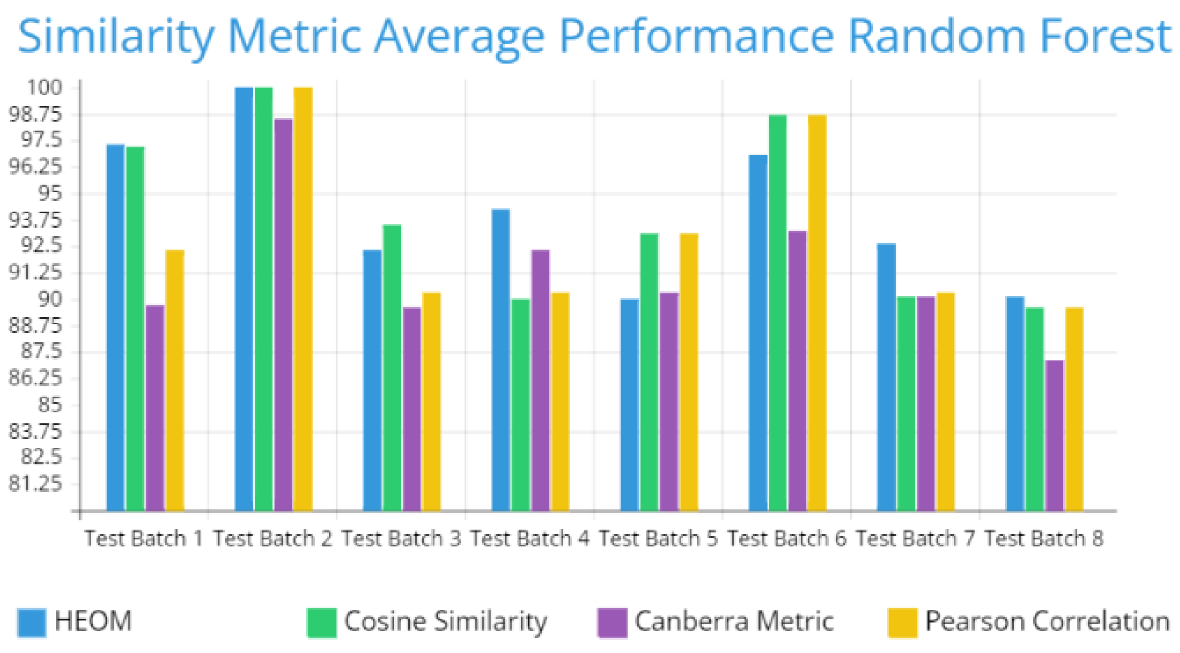

Figure 4.28: Random Forest Performance per Similarity Measure

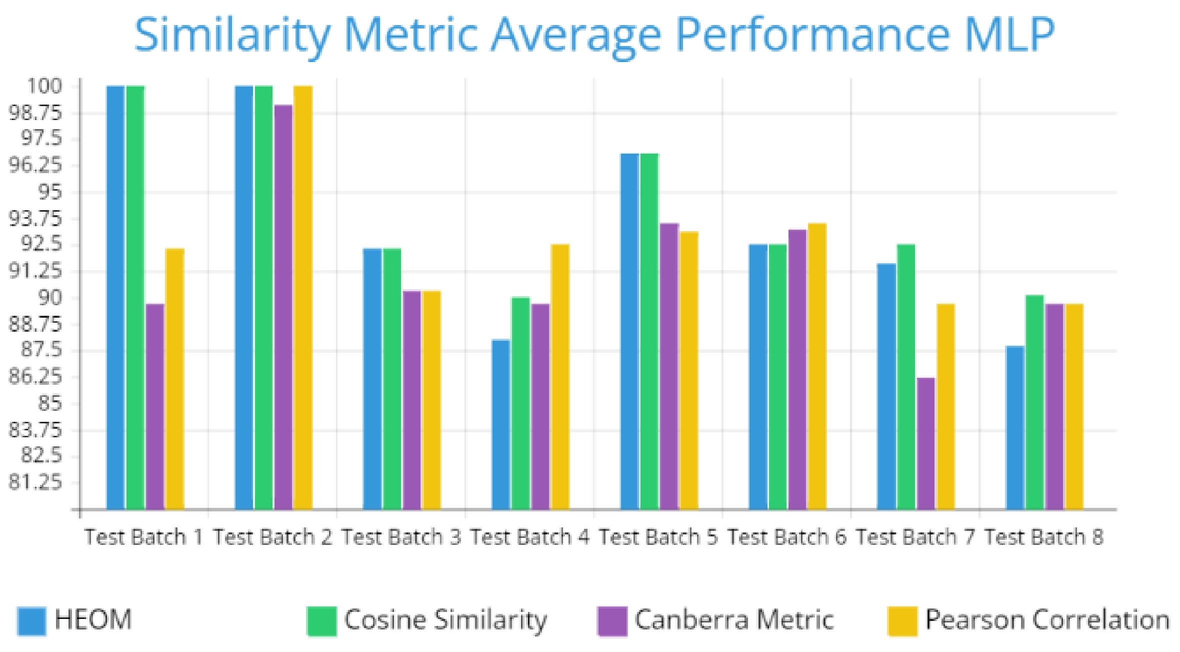

Figure 4.29: MLP Performance per Similarity Measure

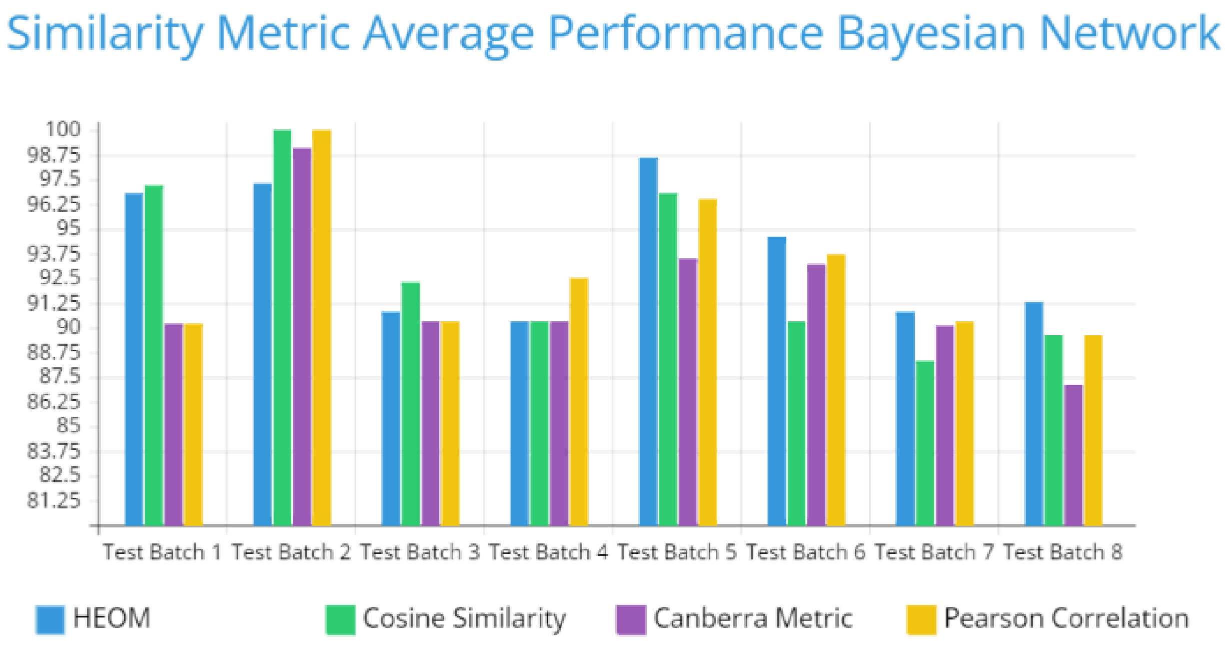

Figure 4.30: Bayesian Network Performance per Similarity Measure

Figures 4.28, 4.29 and 4.30 observes the detection patterns for each classifier used by the FeSAD framework per similarity metric. Observing figure 4.28, which displays the performance for the random forest configuration, we see that the HEOM metric and Cosine similarity metric perform the best in random forest. The HEOM metric achieves marginally higher detection rates in test batches 1, 4, 7 and 8, with equal performance in test batch 2. The Cosine similarity metric achieves higher detection rates in test batches 3, 5, and 6. When used, the Canberra metric is the weakest overall of the four similarity metrics, with the Random Forest displaying lower detection rates than its counterparts. The Pearson correlation displays near-equal performance to the HEOM and cosine similarity metrics in test batches 2, 3, 4, 5, 6, 7 and 8. All four similarity metrics show strong detection rates with the Random Forest and, in most cases, show marginal differences in detection rates, which implies the approach taken by the FeSAD framework is consistent across more than one similarity metric. The Canberra metric being the weakest, may require further modifications to match the other three metrics in terms of detection consistently. The slight differences in detection rates between the four metrics come down to how well each metric separates the numeric data and how the metrics react to being combined with the prediction weights we designed to create maximum differentiation between benign and ransomware samples. Figure 4.29 displays the differences in detection rates achieved by the FeSAD framework when used with the MLP and different similarity metrics. Test batch 1 sees a clear distinction between the results achieved by the HEOM and Cosine similarity metrics and the Canberra metric and the Pearson correlation; however, batches 2 to 7 see consistent detection rates across all four similarity metrics. Figure 4.30 shows the performance of the FeSAD framework when using the Bayesian network combined with the four similarity metrics, and the results follow a similar pattern to the previous two algorithms. The Canberra metric achieves strong results in test batch 2 with a higher detection rate than the HEOM metric, which has shown consistently strong detection rates across all tests thus far. Test batch 3 shows a similar pattern to the previous tests, with the cosine similarity achieving a slightly higher detection rate than its counterparts. Test batch 4 sees the Pearson correlation achieve a slightly higher detection rate than its counterparts which are matched on detection rate. Test batches 5 to 8 see mixed results; however, the HEOM metric performs consistently the best. All four similarity metrics achieve relatively consistent performance, with the Bayesian network showing increases and decreases consistently with the patterns observed in the other algorithms per test batch. Overall in terms of detection rate, the approach taken by the FeSAD framework can perform interchangeably with different similarity metrics to improve the detection rates of a classifier affected by concept drift. There are observable differences in performance between underlying algorithms, and this comes down to the feature sets produced by the FeSAD framework using different algorithms and the subsequent prediction probabilities produced by each of these algorithms during their classification phase.

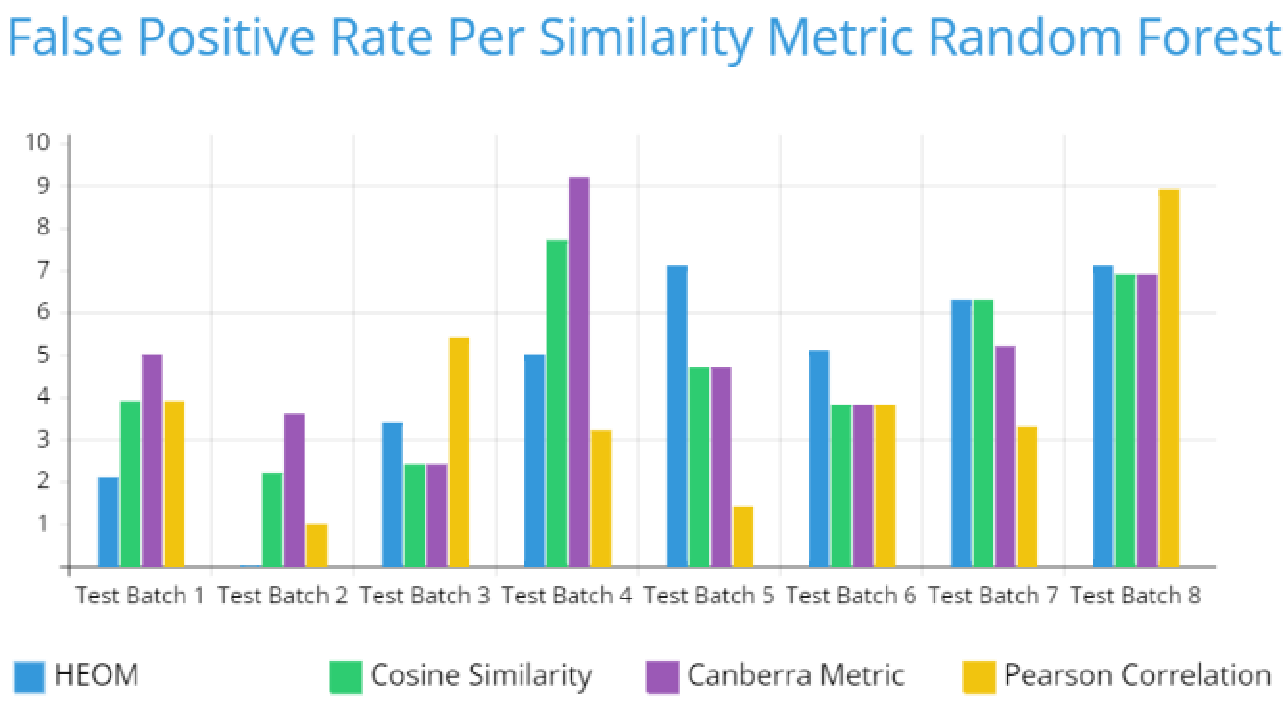

Figure 4.31: Random Forest FPR per Similarity Measure

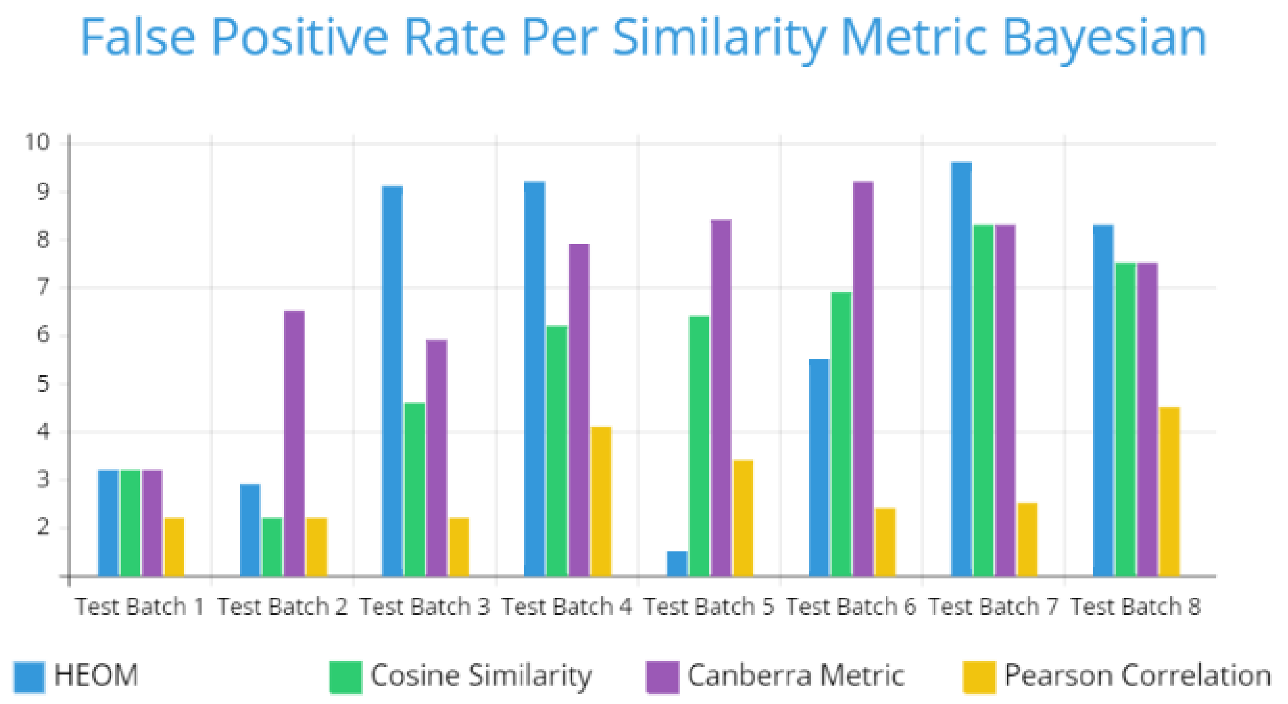

Figure 4.32: Bayesian Network FPR per Similarity Measure

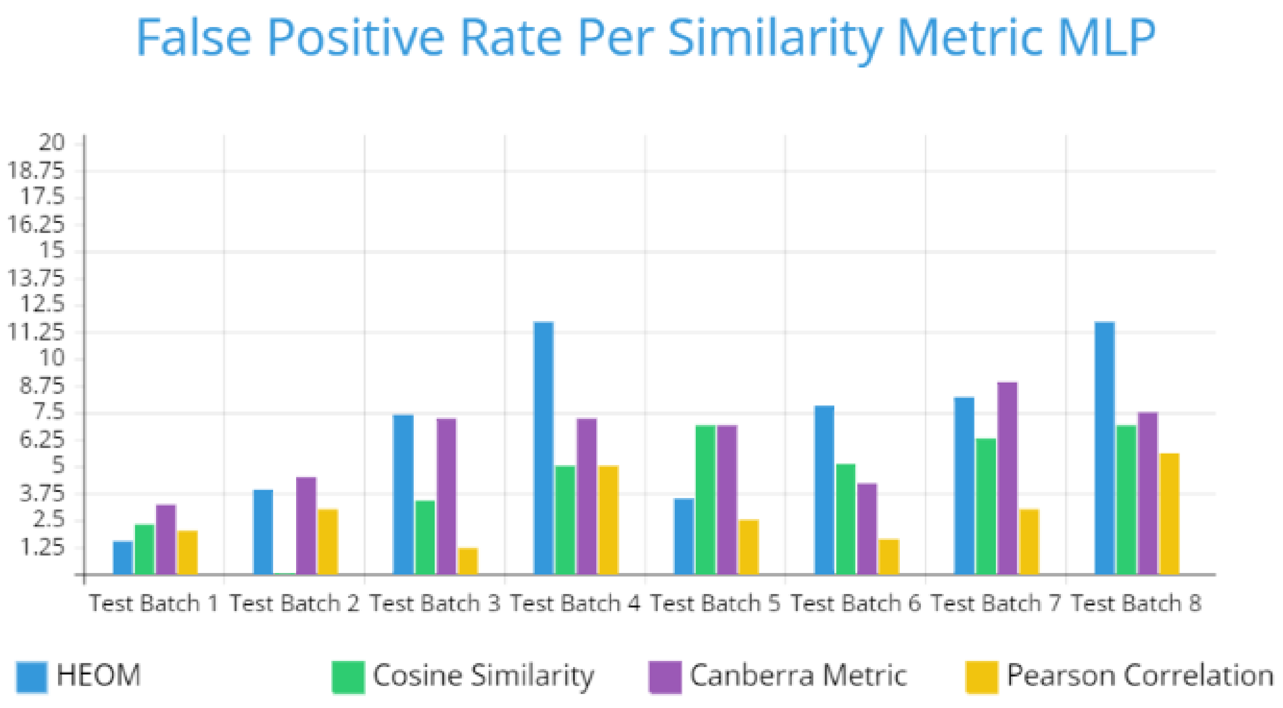

Figure 4.33: MLP FPR per Similarity Measure

Observing Figures 4.31, 4.32 and 4.33, the false positive rate across all algorithms and similarity measures stay controlled under 10% in even extreme testing situations such as test batches 3, 4, 7 and 8. The HEOM metric, which was observed to be the most stable detection rate, appears to degrade faster than the other similarity measures when taking false positives into account. The Pearson correlation shows the lowest increase in false positives over time, especially when combined with the MLP and the Bayesian Network. The false positive rate, in many instances increase, would be due to the data and the introduction of benign files that behave like ransomware; however, it is clear the Pearson Correlation is not as affected. The Pearson correlation showing the resistance to false positives may be due to it not being distance or vector based and it quantifying the linear relationship between samples; this may be able to capture the subtle difference between some benign samples and ransomware better than distance or vector based measure. The measures being taken in relation to ransomware may also explain the more consistent detection statistics, whereas some configurations of algorithms with similarity metrics appear to show a higher false positive rate.

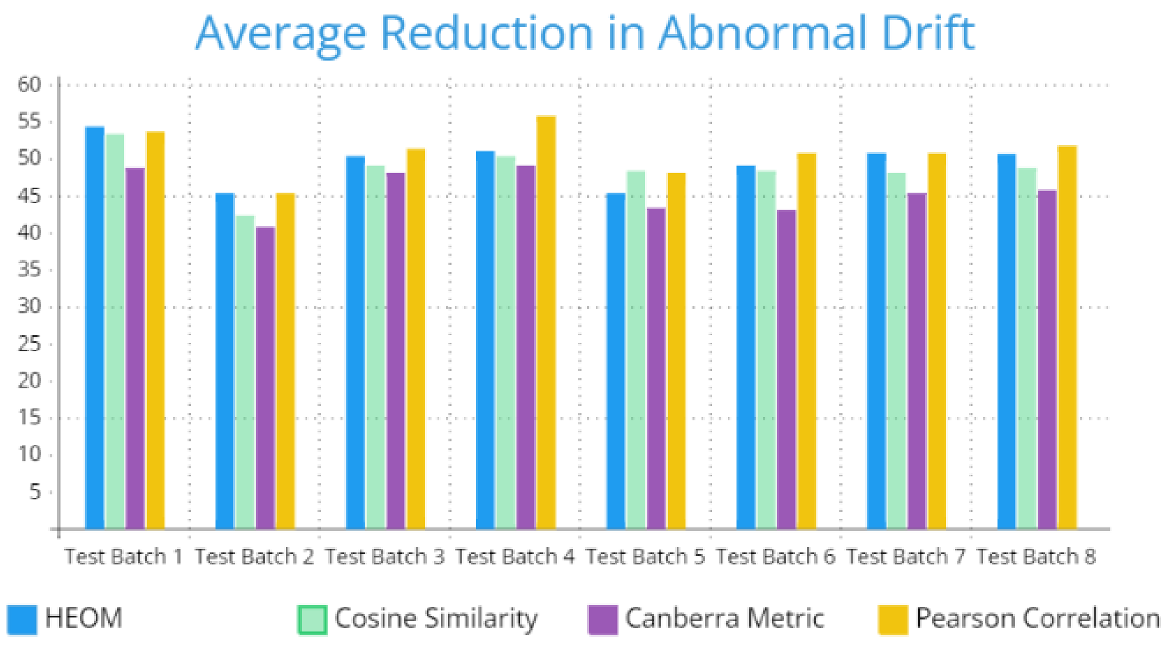

Figure 4.34: Average Abnormal Reduction

Figure 4.34 summarises how well the FeSAD framework captures future drift based on the drift observed between the training and calibration data sets. Figure 4.17 assessed the abnormal drift reduction per classifier when using HEOM, and figure 4.34 shows the abnormal drift reduction per similarity measure. The abnormal drift reduction represents the reduction in samples showing a level of drift that the framework does not expect in the test data. The drift boundaries are calculated using the calibration set, and retraining is recommended when the abnormal samples exceed the threshold. The reduction in abnormal samples suggests the extension of the lifespan of a classifier, as the classifier does not see the same level of uncertainty within a period. The reduction percentage in abnormal drift is quite consistent between the HEOM, Cosine similarity and Pearson Correlation, with the Canberra metric proving slightly less effective. The HEOM significantly reduces test batches 1, 2, and 7, whereas the Pearson correlation shows the highest reduction in abnormal samples in test batches 3, 4, 6 and 8. The reduction levels are fairly consistent between all similarity metrics and provide evidence the calibration data captures a significant amount of the drift observed in future test data. The different reduction levels often come down to the drift between the calibration data and the training data compared to the test data drift; it might be advisable to use multiple calibration datasets to capture the maximum variation between ransomware samples over the years. The FeSAD framework can maintain a consistent level of abnormal sample reduction across all test batches when using different similarity metrics.

Table 4.17 Navarra University Dataset with FeSAD Similarity Configurations

	Random Forest with HEOM	**Bayesian Network with HEOM**	**Random Forest with Cosine**	**Bayesian Network with Cosine**	**Random Forest with Canberra**	**Bayesian Network with Canberra**	**Random Forest with Pearson**	**Bayesian Network with Pearson**
Tested on Training Distribution	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.4% (0.2%, 0.5%) **Precision:** 0.998 (0.997, 0.998%) **Recall:** 0.998 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.3% (0.2%, 0.5%) **Precision:** 0.997 (0.996, 0.998) **Recall:** 0.99 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.2% (0.1%, 0.2%) **Precision:** 0.998 (0.996, 0.997) **Recall:** 0.999 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.3% (0.2%, 0.5%) **Precision:** 0.997 (0.996, 0.998) **Recall:** 0.999 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.2% (0.1%, 0.4%) **Precision:** 0.998 (0.996, 0.997) **Recall:** 0.999 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.2% (0.1%, 0.4%) **Precision:** 0.998 (0.996, 0.997) **Recall:** 0.999 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.2% (0.1%, 0.4%) **Precision:** 0.998 (0.996, 0.997) **Recall:** 0.999 (0.998, 0.999)	**Detection:** 99.9% (99.8%, 99.9%) **FPR:** 0.3% (0.2%, 0.5%) **Precision:** 0.997 (0.996, 0.998) **Recall:** 0.999 (0.998, 0.999)
Tested on Zero-Day Distribution	**Detection:** 94.7% (88.9%, 98.0%) **FPR:** 0.8% (0.6%, 0.9%) **Precision:** 0.991 (0.989, 0.992) **Recall:** 0.943 (0.889, 0.980)	**Detection:** 96.7% (91.3%, 99.0%) **FPR:** 0.2% (0.1%, 0.3%) **Precision:** 0.967 (0.964, 0.969) **Recall:** 0.967 (0.913, 0.990)	**Detection:** 95.1% (88.9%, 98.0%) **FPR:** 0.5% (0.3%, 0.6%) **Precision:** 0.994 (0.993, 0.995) **Recall:** 0.951 (0.889, 0.980)	**Detection:** 96.8% (91.2%, 99.0%) **FPR:** 0.5% (0.4%, 0.6%) **Precision:** 0.995 (0.994, 0.996) **Recall:** 0.968 (0.912, 0.990)	**Detection:** 93.2% (86.6%, 96.9%) **FPR:** 0.5% (0.4%, 0.6%) **Precision:** 0.994 (0.993, 0.995) **Recall:** 0.932 (0.866, 0.969)	**Detection:** 93.2% (86.6%, 96.9%) **FPR:** 0.3% (0.2%, 0.4%) **Precision:** 0.996 (0.995, 0.997) **Recall:** 0.932 (0.866, 0.969)	**Detection:** 95.1% (88.9%, 98.0%) **FPR:** 0.9% (0.8%, 1.0%) **Precision:** 0.990 (0.989, 0.991) **Recall:** 0.951 (0.889, 0.980)	**Detection:** 96.4% (91.3%, 99.0%) **FPR:** 0.3% (0.2%, 0.4%) **Precision:** 0.996 (0.995, 0.997) **Recall:** 0.964 (0.913, 0.990)

Table 4.17 shows the results of using the different similarity measures on the Navarra dataset. As with the experiments presented in table 4.13 the MLP is not used due to its incompatibility with the data format of the Navarra university dataset. The base random forest algorithm achieves a detection rate of 89.5% and a false positive rate of 10.5%. The Bayesian network achieves a detection rate 92.1% detection rate and a false positive rate of 0.9%. Both algorithms combined with the FeSAD framework show improved detection and false positive rates with all three similarity metrics. The Bayesian network achieves slightly higher detection rates than the random forest when used with the FeSAD framework. The HEOM metric achieves an average detection rate of 95.5% on the zero-day distribution. The average false positive rate is at 0.5%; however, the Bayesian network achieves a lower false positive rate than the random forest. The detection rate achieved using the cosine similarity is 95.95% with an average false positive rate of 0.5%. The FeSAD framework achieves an average detection rate of 93.3% with the Canberra metric with a false positive rate of 0.4%. The FeSAD framework achieves a detection rate of 95.9% using the Pearson correlation and an average false positive rate of 0.6%. Overall, the FeSAD framework can use different similarity metrics combined with the random forest and Bayesian network to achieve strong results on the zero-day dataset of the Navarra university ransomware dataset. Overall, the difference between the algorithms is not statistically significant, and they perform at a strong and consistent level when used with the FeSAD framework. The confidence intervals are noticeably smaller here compared to other experiements due to the size of the Navarra dataset.

4.6.1.1 Summarising FeSAD with Different Similarity Metric

Overall, the FeSAD framework has proven to be capable of working with different similarity metrics to achieve strong detection results under concept drift. The first metric used, HEOM, appears to be very stable and consistent in the detection area, whereas the Pearson correlation seems to be the most stable when considering false positives. All similarity metrics help keep detection stats high and present results that are quite consistent, not just in numerical terms, but the degradation patterns appear consistent with each other. The FeSAD framework working with different similarity metrics is critical as it allows it to work with different datasets for which different similarity metrics could be better suited. The reduction in observed abnormal samples is also quite consistent, and this shows that the selected similarity metrics can capture a high proportion of the drift that can occur from training to test set, thus allowing a longer lifespan for a classifier. Notably, the approach taken FeSAD framework appears to be transferrable between different similarity metrics with minimal modification to calculations except for modifying the thresholds for each metric depending on the number they produce. We do not use the log calculations for the cosine similarity as it is already banded with no further need to shrink the drift metrics generated by it.

Table 4.18 FeSAD Train and Test Time Periods

	Random Forest	**Bayesian Network**	**MLP**
Train:2013-15 **Test:**2016-17	**Original Detection Reduction:** 22.7%(12.6%, 35%) **FeSAD Detection Reduction:** 0%(0%, 7.3%)	**Original Detection Reduction:** 17.0%(7.0%, 29.9%) **FeSAD Detection Reduction:** 0%(0%, 1.9%)	**Original Detection Reduction:** 24.7%(15.4%, 38.8%) **FeSAD Detection Reduction:** 0%(0%, 1.0%)
Train:2013-15 **Test:**2018	**Original Detection Reduction:** 5.7%(0%, 15.6%) **FeSAD Detection Reduction:** 0%(0%, 2.0%)	**Original Detection Reduction:** 2.6%(0%, 11.6%) **FeSAD Detection Reduction:** 0%(0%, 1.9%)	**Original Detection Reduction:** 3.7%(0%, 9.6%) **FeSAD Detection Reduction:** 0%(0%, 1.9%)
Train:2013-15 **Test:**2019	**Original Detection Reduction:** 11.1%(3.9%, 24.8%) **FeSAD Detection Reduction:** 4.0%(0%, 12.4%)	**Original Detection Reduction:** 5.8%(0%, 13.4%) **FeSAD Detection Reduction:** 2.1%(0%, 11.4)	**Original Detection Reduction:** 41.6%(29.1%, 54.4%) **FeSAD Detection Reduction:** 4.1%(0%, 12.5%)
Train:2013-15 **Test:** 2020-21	**Original Detection Reduction:** 13.6%(3.1%, 23.0%) **FeSAD Detection Reduction:** 6.3%(0%, 15.6%)	**Original Detection Reduction:** 17.0%(7.0%, 29.9%) **FeSAD Detection Reduction:** 2.1%(0%, 11.6%)	**Original Detection Reduction:** 55.0%(41.0%, 66.0%) **FeSAD Detection Reduction:** 8.4%(0%, 18.2%)
Train:2013-18 **Test:**2019	**Original Detection Reduction:** 9.8%(3.1%, 20.2%) **FeSAD Detection Reduction:** 0%(0%, 7.2%)	**Original Detection Reduction:** 5.8%(0%, 13.4%) **FeSAD Detection Reduction:** 0%(0%, 1.9%)	**Original Detection Reduction:** 22.0%(12.7%, 35.6%) **FeSAD Detection Reduction:** 0%(0%, 1.9%)
Train:2013-18 **Test:**2020-21	**Original Detection Reduction:** 5.8%(0%, 15.6%) **FeSAD Detection Reduction:** 2.1%(0%, 9.8%)	**Original Detection Reduction:** 6.7%(0%, 13.1%) **FeSAD Detection Reduction:** 0%(0%, 1.9%)	**Original Detection Reduction:** 22.0%(12.7%, 35.6%) **FeSAD Detection Reduction:** 3.9%(0%, 9.8%)

The results in Table 4.18 shows the reduction in detection performance when the machine learning classifiers were exposed to new data that showed concept drift. We observe that the FeSAD framework reduced the detection rate in all cases; however, this table also demonstrates that the FeSAD framework can indeed help extend the lifespan of a machine-learning ransomware detection system. The first case we observe is the training in 2013-15 and testing on 2016-17 data. The FeSAD framework maintains the detection rate the base classifier achieves or improves it when tested with the three machine learning algorithms; it can reduce degradation significantly. The case of training on 2013-15 data and testing on 2018 data shows less degradation, but the FeSAD framework can reduce degradation to either 0% or increase the performance of the classifier in this time period. The FeSAD framework, when trained on 2013-15 data and tested on 2019 and 2020-21 data was not able to reduce degradation to 0; however, was able to reduce degradation significantly. The case of training on 2013-18 data and testing on 2019 data again shows FeSAD reducing degradation to 0 or increasing performance in all three cases. In the case of testing the same model on 2019 data, the FeSAD framework achieves low degradation and 0% degradation with the Bayesian network. The FeSAD framework can remain effective in time, especially in the first two test scenarios, where it consistently maintains 0% degradation on 2016, 2017 and 2018 data despite being trained on data from 2013-2015. The degradation experienced due to ransomware from up to 6 years after the classifier was trained limited significantly, with the FeSAD framework further justifying its effectiveness in increasing the time between retraining. The classifiers trained on 2013-18 data show low degradation when tested on 2019 and 2020-21 data; therefore, theoretically, the classifier trained on the 2013-18 data could be stretched beyond 2019 to 2020 or 2021. When tested on 2020-21 data, the low degradation is also promising, with the Bayesian network experiencing no degradation, whereas the Random Forest and MLP experience low degradation. The results shown in Table 4.18 reinforce the idea that the FeSAD framework can help extend literal time periods between retraining and reduce the effect of concept drift over time on a classifier.

4.7 FeSAD Results Discussion

Overall the results obtained by the FeSAD framework have fulfilled what the framework is designed to do. The FeSAD framework significantly improves machine-learning ransomware detection systems' detection rate and performance under concept drift. The robustness of the FeSAD framework is tested using various scenarios which introduce ransomware outside the distribution of the training set. The FeSAD framework shows the ability to reduce performance degradation and extend a classifier's lifespan. The FeSAD can measure the health of a classifier by the number of samples showing unexpected drift; this characteristic is measured by the FeSAD drift decision layer, which uses the prediction probability of the classification and the statistical similarity of the sample to the class it is placed in by the underlying algorithm. The FeSAD framework tunes the tolerances for drift by measuring the changes in concept drift across different ransomware datasets. Using calibration with drifting data provides the FeSAD framework greater insight into drift patterns and how much drift should be expected in addition to the drift seen in the original training set. The use of drift calibration helps extend the lifespan of the classifier, as the FeSAD framework will naturally encounter fewer samples that show a higher level of drift than expected. The average reduction in samples showing unexpected drift is almost 50%, which would infer that the time between retraining could be stretched up to 50%. The percentage reduction in unexpected drift is not the direct increase time between retraining as this would stretch the classifier to its maximum, and ideally, this scenario should be avoided. The FeSAD framework has also provided evidence that it can work effectively with different machine-learning algorithms while combining different similarity metrics with these algorithms to calculate drift. Naturally, different algorithms and similarity metrics will be better suited for specific problems; however, FeSAD being agnostic of similarity measures and classifier allows it to adapt to different ransomware datasets.

The FeSAD framework achieves strong results with the API call features, registry activity, I/O data and the network data chosen for these experiments. We chose API calls as features for our dataset and experiments as they offer insight into the operation of ransomware in the early stages of its execution. The idea behind choosing these features is that a ransomware infection could happen and will eventually surpass End-Point protection systems' pre-execution safeguards. With the constant evolution of ransomware and delivery methods, it is highly plausible that a ransomware sample can be delivered to an end-point and will begin executing before it can be detected. The FeSAD framework assumes that zero-day ransomware will eventually breach pre-execution protection mechanisms, attempt to encrypt an end-point and begin propagating through the network. The early execution API calls used for the FeSAD framework should identify ransomware as it is in the early stages of its execution and prevent irreparable damage to an end-point and thus prevent the spread of the ransomware through the network. The ability of the FeSAD framework to identify the ransomware that may display execution behaviour different to the baseline identified by a machine-learning classifier makes it an effective solution to act as the last line of defence against ransomware infections, not considering backups of files as file backups do not prevent damage to systems and networks. The datasets provided by external entities provide a neutral test for FeSAD and assurance that the FeSAD system can operate with different features and feature sets; however, the features and data in other datasets cannot be guaranteed effective for the early detection we seek. The complexity of each part of the FeSAD algorithm is discussed in Chapter 3.4, the overall complexity of the FeSAD framework in $O(nlogn)$. The framework's complexity is adequate, and the log calculations are necessary to maintain clear distinctions between drifting ransomware and benign files.

4.7.1 Evaluating Confidence Intervals for FeSAD

This subsection addresses the confidence intervals observed when testing the FeSAD framework. The FeSAD framework used test datasets spanning from 2016 to 2021 with 65 ransomware samples and 275 benign samples, respectively. The confidence intervals are observed when the framework is stretched, especially in test batches 7 & 8 where confidence intervals above 7% are observed. It is important to understand that wider confidence intervals are inevitable with a smaller dataset unless the classifier is either flawless or very close to being flawless. The confidence intervals of the FeSAD framework are smaller than those of the underlying classifiers without FeSAD. The FeSAD framework combined with the Pearson Correlation, Cosine Similarity and HEOM metric all show instances of tight confidence intervals for both ransomware and benign files; however, for ransomware, achieving consistently low confidence intervals proves difficult due to the size of the test sets. The confidence intervals observed from testing on the Navarra University dataset are consistently smaller and not exceeding 4%; however, this is also attributed to the zero-day test set provided by Navarra University, which contains 114 ransomware trace samples. The FeSAD framework has consistently shown it can improve detection under concept drift, but the datasets used to test these scenarios could be expanded to improve confidence in the framework's performance. Observing the FeSAD framework's performance when classifying benign files, the confidence intervals observed are consistently smaller than without the FeSAD framework and this performance scales when tested with the Navarra University dataset, which contains more than 11,000 benign file trace samples with the FeSAD framework achieving a false positive rate below 1% with confidence intervals between 0.1% to 0.3%. The tests carried out with the Navarra University dataset show that when exposed to larger datasets, the FeSAD framework's performance can scale up, and the wider confidence intervals seen in our own experiments are contributed to by having relatively small test sets. Unfortunately, the test data sets are limited in size due to the difficulty in obtaining hundreds of unique ransomware samples without significant financial outlay, but the FeSAD framework's ability to scale its performance is demonstrated when exposed to a larger feature set.

4.8 Impact of FeSAD

FeSAD is impactful in multiple ways and has significance in ransomware detection. The first major impact FeSAD has is that it provides a way for ransomware detection systems to adapt to a rapidly changing and evolving ransomware horizon. The FeSAD framework enhances detection performance under concept drift; therefore can extend periods between retraining. The FeSAD framework can measure the concept drift present in ransomware, allowing users to quantify and track ransomware evolution through almost a decade. Tracking the evolution of ransomware allows users to better prepare for future ransomware. The FeSAD framework better prepares a machine learning detection system for ransomware evolution, which has proven inevitable through our research. The framework's impact in helping detection in the face of rapid evolution has other implications. The economic impact created by ransomware, specifically zero-day ransomware attacks, can be reduced through a specifically designed system to detect evolved ransomware strains. The impact of a system that can handle and adapt to concept drift scenarios is particularly impactful due to the dangers posed by ransomware and the irreversible nature of a ransomware infection. The FeSAD framework is also particularly impactful in the research space because this approach is an approach that can be applied in other malware detection fields to counteract concept drift in other malware fields. The FeSAD framework provides the ability to find a baseline to determine between malign and benign behaviour, which can be altered to adapt to the evolution in the field. Overall, the FeSAD framework is impactful on different fronts and provides a meaningful impact on the research space in what it has achieved.

4.9 Concluding Remarks

Overall, the results of the FeSAD framework show it is effective at significantly reducing the effect of concept drift on ransomware detection systems and extending the time a classifier can be reliably used. The FeSAD framework shows that combining underlying machine learning algorithms and the external drift metric helps dynamically identify and adapt to concept drift. The importance of choosing an appropriate underlying algorithm is high, as not all machine learning algorithms can perform with certain feature sets and data types, even though the FeSAD framework is designed to do so. FeSAD works assuming that the feature set and underlying algorithm are proven effective detection methods for ransomware, at least at some point. Despite its evolution, the FeSAD framework proves that it is possible to identify a baseline behaviour for a malware type like ransomware. The ability to find behavioural constants is demonstrated in Figure 4.21, where it is clear that the most important API calls used by ransomware may change in importance over time, but they do not become completely irrelevant, and some API calls maintain high importance through time and ransomware evolution which can be pivotal in detecting under concept drift. The changes in API call importance can help identify when changes in behavioural patterns are expected and unexpected, which naturally helps identify potential misclassifications. The central concept the FeSAD framework takes advantage of is that ransomware behaviour has a certain level of difference compared to benign software behaviour. The statistical and algorithmic measures we use can capture these differences despite the evolution of the malware. We have tested the FeSAD framework on our API dataset, the dataset produced by Imperial University, which contains API, registry and network data and the dataset produced by researchers at Navarra University, which uses network data and I/O data. The FeSAD framework proves to be effective on different feature types; however, we observe that the API call behaviour of ransomware is more easily distinguishable from benign files, not just statistically but through the uncertainty shown by the machine learning algorithms when misclassifying ransomware as benign.

Chapter 5

Conclusion & Future Work

This chapter concludes the work covered in this thesis, revisits the objectives and contributions, and evaluates how these have been achieved. This chapter will discuss the future possibilities for the FeSAD framework and ransomware detection.

5.1 Research Objective

· To review the most recent and cited studies for ransomware detection.

· To review relevant studies on concept drift in data, in malware detection.

· To create a dataset which will allow a framework to be tested under concept drift.

· To test feature selection algorithms to determine if they can help build resilience against concept drift.

· To propose a new framework for ransomware detection which will be able to detect drift and adapt the feature set if possible (If an algorithm used allows local replacement). This framework should reduce training and only resort to completely retraining the system as a last resort.

· To find commonalities between different machine learning algorithms which will be used in order to build an aspect of the framework which will allow the algorithm to be adapted instead of retraining.

5.2 Research Hypotheses

To achieve the research objectives, the following hypotheses have been defined.

· The shortcomings and challenges of current research in ransomware detection, concept drift and feature selection algorithms will help us improve ransomware detection.

· A genetic algorithm can improve ransomware detection feature sets and help us identify components that improve performance under concept drift.

· Machine learning algorithm metrics combined with statistical metrics accurately and reliably classify ransomware that evolves in time.

· It is possible to extend the lifespan of a machine learning classifier for ransomware by integrating concept drift into the training and evaluation process.

The first objective is accomplished through a thorough literature review which evaluates various studies that use machine learning for ransomware detection. The literature review also reviews general malware detection studies that use machine learning as a detection mechanism. The first phase of experiments shown in this thesis details experiments that test machine learning algorithms on various ransomware datasets. The ransomware datasets used in the first experimental phase are procured online or through data taken from Cuckoo Sandbox executions. The independent experiments that test various popular machine learning algorithms gave an insight into what algorithms had the best natural resistance against concept drift.

The second objective is to create a framework that detects ransomware and benign samples showing concept drift; this is achieved through the FeSAD framework. The FeSAD framework uses a statistical measure and takes data from the underlying algorithm to determine which samples behave outside the boundaries the classifier expects. The FeSAD framework has proven highly effective in detecting drift; however, it was not enough to identify drift.

The third objective was to create a framework that can extend the lifespan of an M-L ransomware detection system, and the Drift Calibration and Drift Decision Layer accomplish this. The FeSAD framework uses the statistical data and underlying M-L data from the training set and a data calibration set to define drift thresholds for samples that help identify the ransomware that has been classified as benign and benign files that have been classified as ransomware. The FeSAD framework's integration of drift allows it to significantly reduce the effects of concept drift on a machine learning classifier, slowing performance degradation and reducing ransomware misclassification. The FeSAD framework's use of drift metrics allows it to anticipate concept drift and accounts for significant amounts of the change seen in ransomware behaviour in our eight-year test period. The fulfilment of the objectives would imply that the initial research hypotheses defined at the start of the project have been proven to be true.

5.3 Threats To Validity

This section discusses aspects of the research that can be improved and adjusted to increase the validity of the work.

· The test data sets designed to simulate concept drift scenarios could increase in size. The Test datasets contain 65 ransomware samples each, and show the consistent ability to reduce the effects of concept drift on a detection system. The confidence intervals observed when using small datasets are naturally wide and do not provide enough reassurance in model performance; therefore, increasing the size of the test sets is beneficial.

· The decision boundaries for misclassification can be tuned; currently, it is derived from a midway point between the two distributions. The decision boundary can be derived from past misclassifications and correct classification as opposed to the midway point between acceptable drift for ransomware and benign samples.

· The API-call features used by FeSAD allow the assumption that a behavioural baseline will be maintained between ransomware and benign files due to API calls defining how a program interacts with the Windows OS. The use of API calls assumes that ransomware will always have to have some behavioural elements that will set it apart from benign files. These elements cannot suddenly change in a rapid drift scenario. It would be useful to use other feature types besides API-calls to establish behavioural baselines in other aspects in the event ransomware is able to mask itself through it's API-call behaviours completely.

· The 0.5 threshold value for the important features defined by the feature ranker could be derived by analysing multiple data sets. Additionally, the feature ranker could be redesigned to identify features with consistently high information gain with little reduction in information gain over different distributions.

5.4 Future Work

· The FeSAD framework achieves strong detection rates under concept drift with all the algorithms it was tested with; however, it would be desirable to achieve detection rates closer to 100%. The detection rates achieved are impressive especially considering the degradation observed without FeSAD; however, it is crucial to avoid any ransomware misclassifications.

· The FeSAD framework is effectively using API, network, and I/O features. It might be useful to research further into which features are particularly used pre-encryption to ensure detection is guaranteed before infection.

· The FeSA Layer of the FeSAD framework uses a genetic algorithm to generate strong feature sets to be used in the FeSAD framework. The FeSA Layer provides robust performance but also creates a performance overhead due to its operation mode and reliance on the underlying algorithm as a fitness function. The complexity and performance overhead introduced in the FeSA Layer can be reduced, but using a fitness function besides the underlying algorithm did not yield the best feature sets. It is an interesting challenge to maximise the performance of the FeSA Layer while reducing overheads; this is something that can be improved in the future.

· The FeSAD framework provides valuable insight into ransomware behavioural patterns and the changes that occur in these behavioural patterns. The FeSAD framework could be used differently to integrate directly into its underlying algorithms. It may be useful to integrate the drift data taken from the FeSAD directly into machine learning algorithms as features and then measure the differences between performance; it would be helpful to view how a machine learning classifier processes drift data without FeSAD being used as an independent framework that aids the classifier.

· The theory behind the FeSAD framework could be expanded to be tested on other types of malware, not just ransomware. The FeSAD framework's primary application is for ransomware, and the feature sets and calibrations work mainly to maximise ransomware detection; however, it can be changed and tuned to work for other types of malware. Other types of malware could be detected if the approach to feature selection is altered and the type of features gathered are tuned to adapt to the malware type in question.

· The FeSAD framework is designed to work with different similarity measures and underlying machine-learning algorithms and is proven to improve machine-learning algorithm resistance to concept drift. In terms of future work, existing solutions to concept drift can be looked at to pinpoint areas for the framework to be improved. The closest solution that shares design and conceptual elements with FeSAD would be the Transcend/Transcendent system, and it can be argued that the FeSAD framework builds takes concept drift detection a step further by detecting drift and reliably classifying. Samples that are identified as showing drift. The FeSAD framework does have a retrain feature once performance degradation has reached a threshold; however, currently, this threshold is user-defined, whereas, with a system like Transcend, the framework uses its calculations to determine the retraining threshold based on user standards.

· The FeSAD framework's drift boundaries could be refined and tuned beyond the midpoint between the two distributions. The use of probabilities from the underlying classifiers could also be calibrated to ensure further reliability; this is in the context of tuning prediction probabilities as opposed to using default thresholds of 0.5 for binary classification. Overall, there are aspects of the conformal evaluator, such as its retrain, and performance thresholds which could be integrated into the FeSAD framework, and values like credibility and confidence could be used in place of a similarity metric according to the FeSAD framework.

5.5 Concluding Remarks

Overall the work covered in this thesis has contributed to improving ransomware detection and provides valuable insight into the evolution of ransomware and its effects on ransomware detection systems. Ransomware is currently a prominent issue in the malware detection space, and it costs businesses and individuals a lot in terms of downtime and financial cost. The importance of addressing how ransomware evolves and finding a feasible solution is high. The Ransomware detection space in research contains much work that relies on machine learning classifiers to detect ransomware and zero-day ransomware; however, vulnerabilities of these detection systems are not addressed in a meaningful way, and often the solutions involve the detection of changes and retraining. The reliance on misclassifications to detect classifier degradation is inadequate when dealing with ransomware. The work shown in this thesis effectively addresses the issues with the current work in ransomware detection and takes a step in the right direction when dealing with ransomware evolution.

References

Abbasi, M.S., Al-Sahaf, H., Mansoori, M., and Welch, I. (2022). Behavior-based ransomware classification: A particle swarm optimization wrapper-based approach for feature selection, Applied Soft Computing, Volume 121.

Alhawi, O.M.K., Baldwin, J., and Dehghantanha, A. (2018). Leveraging Machine Learning Techniques for Windows Ransomware Network Traffic Detection. In *Cyber Threat Intelligence. Advances in Information Security*; A.Dehghantanha, M.Conti, T.Dargahi, Eds.; Springer, Cham, Germany, 2018; Volume 70.

Almashhadani, A.O., Kaiiali, M., Sezer, S., and O'Kane, P.A. (2019). Multi-Classifier Network-Based Crypto Ransomware Detection System: A Case Study of Locky Ransomware, *IEEE Access*, *7*, 47053–47067.

Almousa, M., Basavaraju, S., and Anwar, M. (2021). "API-Based Ransomware Detection Using Machine Learning-Based Threat Detection Models," 18th International Conference on Privacy, Security and Trust (PST), pp. 1-7.

Al-rimy, B.A.S., Maarof, M.A., Prasetyo, Y.A.S., Shaid, Z.M., and Ariffin, A.F.M. (2018). "Zero-day aware decision fusion-based model for crypto-ransomware early detection," International Journal of Integrated Engineering, vol. 10, no. 6.

Alvee, S.R.B., Ahn, B., Kim, T., Su, Y., Youn, Y., and Ryu, M. (2021). Ransomware Attack Modeling and Artificial Intelligence-Based Ransomware Detection for Digital Substations, 2021 6th IEEE Workshop on the Electronic Grid (eGRID), pp. 01-05

Alwashali, A.A.M.A., Rahman, N.A.A., and Ismail, N., (2021). A Survey of Ransomware as a Service (RaaS) and Methods to Mitigate the Attack,14th International Conference on Developments in eSystems Engineering (DeSE), pp. 92-96

Ayub, M.A., Continella, A., and Siraj, A. (2020). An I/O Request Packet (IRP) Driven Effective Ransomware Detection Scheme using Artificial Neural Network,IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), pp. 319-324

Baek, S., Jung, Y. Mohaisen, D., Lee, S., and Nyang, D. (2020). SSD-Assisted Ransomware Detection and Data Recovery Techniques, in IEEE Transactions on Computers, vol. 70, no. 10, pp. 1762-1776

Baena-Garcia, M., Del Campo-Avila, J., Figaldo, R., Bifet, A., Gavalda, R., and Morales-Bueno, R. (2006). Early Drift Detection Method, 2006. Fourth International Workshop on Knowledge Discovery from Data Streams.

Barbero, F., Pendlebury, F., Pierazzi, F., and Cavallaro, L. (2022) Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift, IEEE Symposium on Security & Privacy (Oakland).

Barros, R.S.M., Cabral, D.R.L., Goncalves, P.M., and Santos, S.G.T.C. (2017). RDDM: Reactive drift detection method, Expert Systems with Applications, Volume 90, Pages 344-355.

Berrueta, E., Morato, D., Magaña, E., and Izal, M. (2020). Open Repository for the Evaluation of Ransomware Detection Tools, in IEEE Access, vol. 8, pp. 65658-65669.

Brandom, R. (2017). Uk hospitals hit with massive ransomware attack, [Accessed: 10/06/2022] Available at https://www.theverge.com/2017/5/12/15630354/nhs-hospitalsransomware-hack-wannacry-bitcoin

Braue, D., (2022) Gone in 240 seconds: ransomware speeds compared. [Accessed: 30/08/2022], Available Online at: https://ia.acs.org.au/article/2022/gone-in-240-seconds–ransomware-speeds-compared.html

Chang, J.C., Wan, Y.L., and Chen, R.J. (2018). Feature-Selection-Based Ransomware Detection with Machine Learning of Data Analysis. In Proceedings of the 2018 3rd International Conference on Computer and Communication Systems (ICCCS).

Chen, L., Yang, C.Y., Paul, A., and Sahita, R. (2019) Towards resilient machine learning for ransomware detection. In Proceedings of the KDD 2019.

Cover, T.M., and Thomas, J.A. (2006). Elements of Information Theory. John Wiley & Sons, 2nd edition.

Cusack, G., Michel, O., Keller, E. (2018). Machine Learning-Based Detection of Ransomware Using SDN, 2018. In Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks &Network Function Virtualization, SDN-NFV Sec'18,pp. 1–6.

CyberPedia, What is an Exploit Kit, (2018). [Online] [Accessed 21/06/2022] Available at: https://www.paloaltonetwo
is-an-exploit-kit

Daku, H., Zavarsky, P., and Malik, M. (2018) Behavioural-Based Classification and Identification of Ransomware Variants Using Machine Learning, 2018. In Proceedings of the 2018 17th IEEE International Conference On Trust, Security And Privacy, pp. 1560–1564.

Das, S., Liu, Y., Zhang, W., and Chandramohan, M. (2015) Semantics-Based Online Malware Detection: Towards Efficient Real-Time Protection Against Malware, in IEEE Transactions on Information Forensics and Security, vol. 11, no. 2, pp. 289-302

De Groot, J. (2017) A History of Ransomware Attack: The Biggest and Worst Ransomware Attack of All Time. [Accessed 12/06/2022] Available at: https://www.digitalguardian.com/blog/history-ransomware-attacks-biggest-and-worst-ransomware-attacks-all-time

Devetyarov, D. and Nouretdinov, I. (2010) Prediction with Confidence Based on a Random Forest Classifier, AIAI 2010, IFIP AICT 339, pp. 37–44.

Emran, S.M. and Ye,N. (2002). Robustness of chi-square and Canberra distance metrics for computer intrusion detection, Quality and Reliability Engineering International. 18 (1): 19–28.

Fatima, A., Maurya, R., M.K.Dutta, Burget, R., and Masek, J. (2019). Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning, 2019 42nd International Conference on Telecommunications and Signal Processing (TSP).

Fearn, N. (2020) Double extortion ransomware attacks and how to stop them. [Accessed: 11/06/2022] Available at https://www.computerweekly.com/feature/Double-extortion-ransomware-attacks-and-how-to-stop-them?

Fearn, N. (2020) Ransomware victim Travelex forced into administration [Accessed: 11/06/2022] Available at https://www.itgovernance.co.uk/blog/ransomware-victim-travelex-forced-into-administration

Fernando, D.W., Komninos, N., and Chen, T. (2020) A Study on the Evolution of Ransomware Detection Using Machine Learning and Deep Learning Techniques. IoT 2020, 1, 551-604. https://doi.org/10.3390/iot102003

Fernando, D.W., and Komninos, N. (2022). FeSA: Feature selection architecture for ransomware detection under concept drift, Computers & Security, Volume 116, 2022, 102659, ISSN 0167-4048.

Freed, A.M. (2021) Ten of the Biggest Ransomware Attacks of 2021. [Accessed: 10/06/2022] Available at https://www.cybereason.com/blog/ten-of-the-biggest-ransomware-attacks-of-2021

Gao, J., Fan, W., Han, J., and Yu, P.S. (2007). A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions, Proceedings of the Seventh SIAM International Conference on Data Mining.

Ghomeshi, H., Gaber, M.M., and Kovalchuk, Y. (2019). EACD: evolutionary adaptation to concept drifts in data streams, Data Mining and Knowledge Discovery 33:663-694, 2019.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.

Gomez-Hernandez, J.A., Alvarez-Gonzalez, L., and Garcia-Teodoro, P. (2018) R-Locker: Thwarting ransomware action through a honeyfile-based approach. Computers & Security, 73: p. 389-398.

Guerra-Manzanares, A., Luckner, M., and Bahsi, H. (2022)Concept drift and cross-device behavior: Challenges and implications for effective android malware detection, Computers & Security, Volume 120.

Hasan, M., and Rahman, M. (2017). RansHunt: A Support Vector Machines Based Ransomware

Analysis Framework with Integrated Feature Set,2017, In 20th International Conference of Computer and Information Technology (ICCIT), pp. 1-7.

Hidalgo, J.I.G., Mariño, L.M.P., and Barros, R.S.M. (2019). Cosine Similarity Drift Detector, Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series, 2019, Volume 11730.

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R.R. (2012). Improving neural networks by preventing co-adaptation of feature detectors, arXiv preprint arXiv:1207.0580.

Hussain, S.S., Hashmani, M., Uddin, V., Ansari, T., and Jameel, M. (2021). A Novel Approach to Detect Concept Drift Using Machine Learning, 2021 International Conference on Computer & Information Sciences (ICCOINS), pp. 136-141

Hwang, J., Kim, J., and Lee, S. (2020) Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques. *Wireless Pers Commun.* 2020, *112*, 2597-2609.

Jordaney, R., Sharad, K., Dash, S.K., Wang, W., Papini, D., Nouretdinov, I., and Cavallaro, L. (2017). Transcend: Detecting Concept Drift in Malware Classification Models, Proceedings of the 26th USENIX Security Symposium.

Kantchelian, A., Afroz, A., Huang, S., Islam, A., and Miller, B. (2013) M.Tschantz, R.Greenstadt, A.D.Joseph, J.D.Tygar, Approaches to adversarial drift, AISec'13, Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located.

Kerner, S.M. (2022). Ransomware trends, statistics and facts in 2022 [Accessed: 10/06/2022] Available at https://www.techtarget.com/searchsecurity/feature/Ransomware-trends-statistics-and-facts

Khan, F., McNube, C., Lakshmana, R., Kadry, S., and Nam, Y. (2019). A Digital DNA Sequencing Engine for Ransomware Detection Using Machine Learning. *IEEE Access*, *8*, 119710–119719.

Kost, E. (2021).Ransomware [Accessed: 10/10/2023] Available at https://www.upguard.com/glossary/ransomware

LeCun, Y.,Bengio, Y., and Hinton, G. (2015). Deep learning, Nature, vol. 521, no. 7553,pp. 436-444.

Lee, K., Lee, S.Y., Yim, K. (2019). Machine Learning Based File Entropy Analysis for Ransomware Detection in Backup Systems. *IEEE Access* *7*, 110205–110215.

Liska, A., and Gallo, T. (2017). Ransomware: Defending Against Digital Extortion. O'Reilly.

Maggi, F., Robertson, W.K., Krugel, C., and Vigna, G. (2009) Protecting a moving target: Addressing web application concept drift. In Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Proceedings, pp. 21–40

Manavi, F. and Hamzeh, A. (2021) Static Detection of Ransomware Using LSTM Network and PE Header, 2021 26th International Computer Conference, Computer Society of Iran (CSICC), pp. 1-5

Maniath, S., Ahok, A., Poornach, R.P., Sujadev, V.G., Prem Sankar, A.U., and Jan, S. (2017). Deep Learning LSTM based Ransomware Detection, Proceedings of the Recent Developments in Control, Automation & Power Engineering (RDCAPE), pp. 442-446.

Molina, R.M.A., Torabi, S., Sarieddine, K., Bou-Harb, E., Bouguila, N., and Assi, C. (2021). On Ransomware Family Attribution Using Pre-Attack Paranoia Activities, IEEE Transactions on Network and Service Management, vol. 19, no. 1, pp. 19-36.

Moreno-Torres, J.G., Raeder, T., Alaiz-Rodriguez, R., Chawla, N.V., Herrera, F., (2012) A unifying view on dataset shift in classification, Pattern Recognition, Volume 45, Issue 1, pp. 521-530.

Olaimat, M.N., Aizaini Maarof, M., and Al-rimy, B.A.S. (2021). Ransomware Anti-Analysis and Evasion Techniques: A Survey and Research Directions, 2021 3rd International Cyber Resilience Conference (CRC), pp. 1-6.

Olenick, D. (2018) AI use in ransomware attacks and sextortion schemes top, Malwarebytes 2018 report, Malwarebytes, [Online] [Accessed 01/07/2022] Available at:https://www.scmagazine.com/home/security-news/malware/ai-use-in-ransomware-attacksand-sextortion-schemes-top-malwarebytes-2018-report/

Peker, M., Arslan, A., Şen, B., Çelebi, F.V., and But, A. (2015). A novel hybrid method for determining the depth of anesthesia level: Combining ReliefF feature selection and random forest algorithm (ReliefF+RF), 2015 International Symposium on Innovations in Intelligent Systems and Applications (INISTA), pp. 1-8.

Poudel, S., Subedi, P., and Dasgupta, D. (2018). A Framework for Analyzing Ransomware using Machine Learning, 2018. In Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence (SSCI).

Pudukotai Dinakarrao, S.M., Sayadi, H., Makrani, H.M., Nowzari, C., Rafatirad, C. and Homayoun, H. (2019). Lightweight Node-level Malware Detection and Network-level Malware Confinement in IoT Networks, 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 776-781

Richardson, R. and North, M. M. (2017). Ransomware: Evolution, mitigation and prevention, International Management Review, vol.13, no. 1, p. 10.

Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). Automatic Analysis of Malware Behavior Using Machine Learning. Journal of Computer Security, 19, 639-668.

Ross, G.J., Adams, N.M., Tasoulis, D.K., Hand, D.J. (2012). Exponentially weighted moving average charts for detecting concept drift, Pattern Recognition Letters, 33 (2), 191–198.

Sammut, C., Harries, M. Concept Drift. In: Sammut, C., Webb, G.I. (2010). Encyclopedia of Machine Learning. Springer, Boston, MA.

Seong, B., Gyu, L., Gyu, I. (2019). Ransomware detection using machine learning algorithms. In *Wiley Online Library, Concurrency and Computation: Practice and Experience, Volume 32, Issue 18.*.

Sgandurra, D., Munoz-Gonzalez, L., Mohsen, R., Lupu, E. (2016). Automated Dynamic Analysis of Ransomware: Benefits, Limitations and Use for Detection. arXiv, arXiv:1609.03020

Shafer, G., and Vovk, V. (2008). A tutorial on conformal prediction, The Journal of Machine Learning Research 9, 371–421, 2008.

Shaukat, S., and Ribeiro, V. (2018). RansomWall: A Layered Defence System against Cryptographic Ransomware Attacks using Machine Learning. Proceedings of the 10th International Conference on Communication Systems and Networks, (COMSNETS), pp. 356–363.

Sheen, S. and Yadav, A. (2018). Ransomware detection by mining API call usage, 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 983-987

Shirkhorshidi, A.S., Aghabozorgi, S., Wah, T.Y., (2015). A Comparison Study on Similarity and Dissimilarity Measures in Clustering Continuous Data. PLoS ONE.

Sibi Chakkaravarthy, S., Sangeetha, D., Cruz, M.V., Vaidehi, V., and Raman, B. (2020). Design of Intrusion Detection Honeypot Using Social Leopard Algorithm to Detect IoT Ransomware Attacks, 2020 in IEEE Access, vol. 8, pp. 169944-169956

Singhal, S., Chawla, U., Shorey, R. (2020). Machine Learning & Concept Drift based Approach for Malicious Website Detection, 12th International Conference on Communication Systems & Networks (COMSNETS).

SonicWall. (2018) SonicWall Cyber Threat Report.

SonicWall. (2021) SonicWall Cyber Threat Report.

Sophos Knowledge Base: (2018) Ransomware: How an attack works. [Online] [Accessed 21/11/2018] Available at: https://community.sophos.com/kb/en-us/124699

Taile, J.P., and Patel, A.D. (2017). A Comprehensive Survey: Ransomware Attacks Prevention, Monitoring and Damage Control, International Journal of Research and Scientific Innovation (IJRSI) — Volume IV, Issue VIS, June 2017 — ISSN 2321–2705

Takeuchi, Y., Sakai, K., and Fukumoto, S. (2018). Detecting Ransomware using Support Vector Machines, 2018. In Proceedings of the 47th International Conference on Parallel Processing Companion..

Tan, G., Zhang, P., Liu, Q., Liu, X., Zhu, C., and Dou, F. (2013). Adaptive Malicious URL Detection: Learning in the Presence of Concept Drifts, 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering pp. 99–110.

Tseng, A., Chen, Y., Kao, K., and Lin, T. (2016). Deep Learning for Ransomware Detection.*Aragorn 2016 DeepLF*, 2016.

Tsymbal, A., Pechenizkiy, M., Cunningham, P., and Puuronen, S. (2008). Dynamic integration of classifiers for handling concept drift, Information Fusion, Volume 9, Issue 1,Pages 56-68.

Urooj, U., Maarof, U.M.A.B., and Al-rimy, B.A.S. (2021). A proposed Adaptive Pre-Encryption Crypto-Ransomware Early Detection Model, 2021 3rd International Cyber Resilience Conference (CRC), pp. 1-6

VinayKumar, R., Soman, K.P., Senthil Velan, K.K., and Ganorkan, S. (2017). Evaluating Shallow and Deep Networks for Ransomware Detection and Classification. Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI).

Vivekanandan, P. and Nedunchezhian, R. (2011). Mining data streams with concept drifts using genetic algorithm. Artif Intell Rev 36(3):163–178.

Waddell, K. (2016). The Computer Virus that Haunted Early AIDS Researchers. [Accessed: 10/06/2022], Available Online at: www.theatlantic.com/

Wen, L., Lingdi, P., Wu, C., and Ming, J. (2010) Distributed Bayesian Network Trust Model in Virtual Network, 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing.

Winder, D. (2020). Are old operating systems putting the NHS at risk in 2020?, [Accessed: 10/06/2022] Available at https://www.digitalhealth.net/2020/09/are-old-operating-systems-putting-the-nhs-at-risk-in-2020/

Witten, H., & Frank, E. (2005) Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)

Won, D.O., Jang, Y.N. and Lee, S.W. (2022). PlausMal-GAN: Plausible Malware Training Based on Generative Adversarial Networks for Analogous Zero-day Malware Detection, IEEE Transactions on Emerging Topics in Computing.

Yeo, M., Koo, Y., Yoon, Y., Hwang, T. and Ryu, J., Song, J., and Park, C. (2018) Flow-based malware detection using convolutional neural network, 2018 International Conference on Information Networking (ICOIN), pp. 910-913Zakaria, W.Z.A., Mohd, M.F.A.O., and Ariffin, A.F.M. (2017). The Rise of Ransomware, ICSEB 2017 Proceedings of the 2017 International Conference on Software and e-Business Pages 66-70

Zhang, X., Wang, J and Zhu, S. (2021). Dual Generative Adversarial Networks Based Unknown Encryption Ransomware Attack Detection, in IEEE Access, vol. 10, pp. 900-913

Zhu, J., Jang-Jaccard, J., Singh, A., Welch, I., AI-Sahaf, H., and Camtepe, S. (2022). A few-shot meta-learning based siamese neural network using entropy features for ransomware classification, Computers & Security, Volume 117.

Zimba, A. (2017). Malware-Free Intrusion: A Novel Approach to Ransomware Infection Vectors.

Zuhair, H., Selamat, A., and Krejcar, O. (2020). A Multi-Tier Streaming Analytics Model of 0-Day Ransomware Detection Using Machine Learning. *Appl. Sci.10*, 3210