DOI: https://doi.org/10.30837/IST-2024.P1.144

Enhancing Intrusion Detection Systems through Advanced Feature Engineering and Machine Learning Techniques

Aditya Patel
pateladij1201@gmail.com

Abstract*—In this study, I explore advanced methodologies for improving intrusion detection systems (IDS) by leveraging machine learning algorithms and sophisticated feature engineering. I initially assess the performance of various classification algorithms, including k-nearest neighbors (KNN) and support vector machines (SVM), on the ADFA-LD 12 dataset. My analysis reveals that SVMs outperform other methods, particularly when using two-sequence feature spaces instead of traditional frequency-based approaches. I conduct a detailed evaluation of SVM performance with linear and sigmoid kernels, revealing that the two-sequence feature space significantly enhances detection accuracy. Recursive feature elimination demonstrates that optimal performance can be achieved with fewer than 240 features, underscoring the importance of effective feature selection. In contrast, one-class SVMs, used for outlier detection, show comparatively poor performance, indicating that traditional outlier detection methods may not be as effective in this context. My findings highlight the efficacy of SVM classifiers in both two-sequence and frequency feature spaces, though the latter does not offer substantial improvement. The study also emphasizes the need for further research into scalable algorithms and improved kernel methods for unsupervised outlier detection. Future work should focus on integrating domain-specific knowledge for feature engineering and exploring modern datasets to validate and enhance these findings.

I. Introduction

In the industry I are currently experiencing what many people call the fourth industrial revolution. The main characteristics of this disruptive process are:

- The ubiquitous presence of connected devices, collectively called Internet of Everything.
- The ability of cyber system to interact with and affect the physical world creating the so called cyber-physical systems (CPS).
- The growth of distributed computing and the capabilities that it allows.
- The evolution of Machine Learning enabling cyber-physical systems with increased autonomy.[1]

One common denominator of the aspects of the fourth industrial revolution is increased connectivity of computing devices. This brings the negative side effect that more facets of my life and society are exposed online making cyber-security even more important. During the last couple of years I have seen many examples of this danger.

[1]A good example of an autonomous cyber-physical system are Tesla's self-driving cars.

Further expanding on the example of cars I mention that on the summer of 2015 two american security researchers demonstrated that a contemporary model Jeep Cherokee could be remotely accessed maliciously over mobile telephony network.[2] As demonstrated in the afforementioned article[2] this has resulted in the academia, the industry and regulators considering policy decisions for regulation and standards adoption in order for car safety to keep up with technological innovation. Another significant event is the attack on a regional electricity distribution company, Ukrainian Kyivoblenergo, on 23 December 2015. The attack compromised the company's computer systems and their supervisory control and data acquisition systems (SCADA). It resulted in power outages for around 225,000 customers for a period of hours. The diligence of the company employees and the quick transition on manual mode allowed the company to restore its service with little delay. The attack is thought to have been carried out by a state actor and could have inflicted more damage had the perpetrators more time before making their pressence known. Again this attack has attracted a lot of attention among security researchers and the industry in order to formulate appropriate policies to protect core infrastructure[3].

Now that I have seen examples of the increasing importance of cyber - security let's look deeper at what it actually is. Cyber security generally consists of two parts. Intrusion Prevention Systems (IPS) and Intrusion Detection Systems (IDS). Put it simply an IPS prevents the attacker from getting in and an IDS detects him once he is in. With the emergence of cloud computing the boundaries between the two are getting blurred[4]. Nowadays modern IDS have a wide range of features including the detection of an occuring attack, automated responses and detection of malicious behavior within the system[4].

II. Foundations of the project

In this project I are drawing knowledge from two related but distinct fields. [2] The first field is outlier and anomaly detection techniques from Machine Learning. The second field is Intrusion Detection techniques within deployed cyber-security solutions. On the next sections I will describe the foundations of this project. They have been studied to a big extend during the Project preparation course.

[2]With info from:
Project Preparation: Outlier Detection in Cybersecurity Application, 2016, submitted by Nikolaos Perrakis.

III. Outlier and Anomaly Detection

The concept of bad data has been quite old in the field of statistics. One of the first systematic approaches to dealing with them, dating back to the 19^{th} century, has been Chauvenet's criterion on whether one data point of an experimental data set was spurious or not. Moving on to modern machine learning techniques on outlier and anomaly detection I see that there are many approaches. They are based on different statistical methods but also on different characteristics on the domain from which the dataset comes. Below are most relevant methods with regards to my work.

The first method has been put forward by Roberts et al.[5]. They use a Gaussian Mixture model to map the normal behavior of the system their dataset describes. This creates a set of normal system states their trained model recognizes. In their method, they minimize the set of heuristically chosen parameters used in such techniques by using an evolving threshold on how much a data point can differ from a learned normal state before it gets classified as belonging to another state. When I am training our model with normal operation data, this results in a newly learned normal state. Afterward, the same threshold is used to identify an anomaly in the test data. They then test their method in a set of electroencephalograms (EEG) obtained from patient data. They prove the robustness of their method by successfully identifying epileptic seizures when they occur. They do not quantify the success of their results by mentioning accuracy, precision or fall out rates. However they include results from particular examples of EEG activity and explain their performance metrics result comparing them with input from domain related knowledge.

A pioneering approach in unsupervised approaches for anomaly detection came from Schölkopf et al.[6]. They propose an algorithm that estimates the region where the probability density of some given data lives. New data can be compared to this density for anomaly detection purposes. They describe their method in detail and describe it's conceptual limitations. Then they demonstrate it's characteristics in an artificial dataset and it's effectiveness in a real dataset. Their algorithm has come to be known as One class Support Vector Machine algorithm and it has been a very influential algorithm in the field of outlier detection.

Hayton et al.[7] have written another influential paper. They studied Support Vector machine-based anomaly detection in Jet engine vibration spectra. Their approach also adds the feature of combining a second dataset in order to train their model more accurately. Moreover, during the discussion of their results, Hayton et al. gave a very insightful discussion on whether it is preferable to address novelty detection as a 2-class classification problem or not. They argue that when the "abnormal" data points are representative of the abnormal class, then training a model as a binary classification problem is optimum. On the other hand, they point out that the novel data points may be artificial and that their nature may be non-stationary. In domains with those characteristics, it is better to treat only the normal behavior data points as representative of their class. This is also the case in the Information Technology domain, which is why deployed Intrusion Detection Systems use the latter approach.

Clifton et al.[8] are a team of researchers that use class Support Vector Machines for anomaly detection. study luminosity measurements of a Typhoon G30 combustor engine and create an SVM model for anomaly detection. They use wavelet analysis on the multi-channel combustion data to create their feature space and subsequently perform supervised learning to train their SVM to recognize anomalous behavior. Clifton et al. also compare the results of the SVM approach to the GMM approach on the same dataset. They found that SVM-based anomaly detection performed better and demonstrated that with instances of multi-channel combustion data where the SVM model identified the novelty at earlier times compared to the GMM model.

Another interesting approach on the subject touches on the issue of the nature of the novel data points being non-stationary. Farran et al.[9] study the KDD 1999 Cup dataset[3]. Their technique uses two steps. The first uses a method called Voted Spheres, which runs over the dataset once and performs non-parametric classification. It results in partitioning the feature space into a series of overlapping spheres. The second step is to take into account the possibility that the test set may not come from the same distribution as the training set. They use two algorithms, important weighting, and kernel mean matching, to account for that difference. It is useful to mention the author's note that kernel mean matching does not scale very well as the dataset increases due to its reliance on quadratic programming, something that may negate the low computation load of the Voted Spheres method.

IV. Intrusion Detection Systems

Bhuyan et al.[10] give the following description: "Intrusion is a set of actions aimed to compromise the security of computer and network components in terms of confidentiality, integrity, and availability". Intrusion Detection Systems are my answer to that threat. The basic assumption I make is that my system will behave differently during an intrusion than during normal operation. And this is why it is appropriate to use novelty detection in IDS applications.

There are many attack actions against a computer system that can be classified as intrusions with the above definition. Each of them has different specific characteristics. Thus it is helpful for us to divide attacks into certain classes. As I did during Project Preparation[4] I will use the classification used in Bhuyan et al.[10] and Ahmed et al.[11]. I classify intrusions as Malware attacks, Denial of Service attacks, Network attacks, Physical attacks, Password Attacks, Information Gathering Attacks, Remote to User attacks, and Root attacks. Another important property of intrusions is that they have varying anomaly characteristics. I, therefore, classify them in

[3]DARPA IDS 1998 - 1999 Datasets: https://www.ll.mit.edu/ideval/data/ - Accessed at 9 Aug 2016

[4]Project Preparation: Outlier Detection in Cybersecurity Application, 2016, submitted by Nikolaos Perrakis.

an additional way. There are *point anomalies* when a data point is considered anomalous when it is distant in comparison to the points that model the normal behavior of the system. This type of intrusion closely fits the machine learning problem of anomaly detection, and it will be the main focus of my work. There are also *contextual anomalies* when a data point may be considered anomalous according to the context it is associated with, as well as its place in the feature space. The context may be additional features engineered in my feature space or an evaluation of circumstances associated with the data point from the IDS (or its operator). Lastly, there are *collective anomalies* for which a collection of data is considered anomalous, but each data point, taken individually, is not. In Table I[5], I describe the eight classes of attacks, give examples of them, and classify the examples with their respective type of anomalies.

An operational enterprise IDS solution consists of main parts. Two main parts are a Network Intrusion Detection System (NIDS) which analyses traffic over a company's network to detect intrusion and a Host-based Intrusion Detection System (HIDS) which is installed on the company's computers (hosts) and monitors them in order to detect intrusion.

There are many techniques used to identify anomalies on an IDS. Bhuyan et al.[10] do a good work on categorizing them, and I will follow their classification scheme in presenting them. It should be noted, however, that these methods are not completely distinct from each other, and a particular implementation may include aspects from more than one. The first class is statistical methods, which also includes Bayesian Networks. A model is trained to "learn" the normal operation of the system. A threshold of statistical distance from normal operation is determined, and data points exceeding that threshold are classified as anomalous. For example, Krueger et al.[12] used Bayesian Networks as part of an IDS system and managed to reduce the false alarm rates compared to threshold-based approaches. The second class is clustering methods. A distance or similarity method has to be defined on the feature space. It is then used to cluster the data with various algorithms, such as k-means clustering. I then measure the distance of new data points with my learned clusters And use it to classify them as anomalous or not. One example of such a method is the work of Bhuyan et al.[13], where they create a reference point clustering method to perform outlier detection that works well on large datasets. It is useful to note here that a clustering method, such as k-means clustering, conceptually is quite similar to a Gaussian Mixture model. Such similarities exist for various algorithms that, under different implementations, can be labeled as different classes in my classification scheme. The third class is classification methods. They include both supervised and unsupervised algorithms. One good example that I will use later is support vector machines (SVM). They can be trained either on pre-labeled data or on non-labeled data, depending on the SVM implementation. Another useful example of such techniques is Support Vector Data Description, a one-class classification method described and further extended by Kang et al.[14]. The fourth class is knowledge-based methods. They take advantage of what has been learned from previous attacks so that they are able to identify them when they re-appear. These methods include ontology, signature, and logic-based approaches. One example of a knowledge-based method is Xu 's[15] work. In it, he introduces a sequential anomaly detection method that takes advantage of a Markov reward process in a reinforced learning approach. The fifth class is soft computing methods. The key point behind them is that if finding the exact solution is not feasible, then I can look for approximate solutions. These methods include Artificial Neural Networks, Rough Sets, Fuzzy Sets, Ant Colony Algorithms, Artificial Immune Systems, and Genetic Algorithms. I mention as an example the work of Amini et al.[16], where they use adaptive resonance theory (ART) and self-organizing maps (SOM) unsupervised neural networks for real-time intrusion detection. The last class is called combination learner methods and basically consists of techniques that combine more than one of the previous classes in their implementation. A good example of such a method is the work of Borji[17]. He uses my classifiers, decision trees, SVM's, ANN, and kNN, which he combines with three different strategies: majority voting, belief measure, and Bayesian averaging.

The variety of methods for Intrusion detection can be attributed to their diverse strengths and weaknesses. Additionally, the intrusions each IT infrastructure faces are different, which means that different IT networks are best suited to different Intrusion Detection methods. For example, clustering and Nearest neighbor algorithms have poor performance on high dimensional data because, in those cases, distance methods cannot accurately differentiate between anomalous and normal data points. As an example, this problem is studied by Aggarwal et al.[18], where they mention that the meaning of proximity in high dimensions is problematic. They study the problem empirically, focusing on L_k norms, and propose fractional k's as a potential solution, though I will not try them here. Other ways to overcome this are feature reduction techniques, such as spectral techniques or principal component analysis, but he has to be careful to maintain the separation between anomalous and normal data points. In general, classification techniques suffer because of the need to pre-label data points. Creating those labels is hard and often artificial. This results in the subsequently trained model becoming outdated quite fast because of the fast-paced evolution of the IT field. Using partially labeled data for semi-supervised clustering techniques can be more efficient than classification methods, provided I have a feature space with a good distance measure. If not, statistical techniques are a better option. As I mentioned, another point to consider is how easy it is to update my application. Statistical, clustering, and classification methods are all hard to train. But it can be done offline, and their testing phase is fast. Last but not least, there are times when the assumption that intrusion events

[5]Table taken from Project Preparation: Outlier Detection in Cybersecurity Application, 2016, Nikolaos Perrakis

Table I
TYPES OF ATTACKS AND ASSOCIATED ANOMALIES.

Attack Type	Description	Examples (Anomaly Type)
Malware	Virus, Worm, Troyan: A program that may replicate and transfer on its own and perform harmful operations on the infected computer.	Stuxnet Worm (point)
Denial of Service	Attacks that make Network resources inaccessible.	Smurf (collective)
Network	Compromising the security of a Network by exploiting Network protocols.	Man-in-the-Middle (point)
Physical	Compromising a system or a network through physical access.	Evil Maid (point)
Password	Trying to find a user's password, usually through multiple login attempts.	Dictionary attack (collective)
Information Gathering	gathering information to try to find vulnerabilities in a network.	Port scan (collective)
Remote to User	Trying to get remote access as a user to a system.	phf (point)
User to Root	Upgrading a user's privilege to superuser.	Rootkit (point)

are rare compared to normal events is not true. In that case, an intrusion detection method may end up with a high false positive rate. This is a big problem because it interrupts the user's normal operation.

As I see Intrusion Detection is a very complex problem. Each IT system has different characteristics and they are better addressed by different techniques. This means that addressing Intrusion Detection to it's entirety goes well far beyond the scope of this project. Therefore I will limit ourselves to a particular case.

V. DATASETS

A critical part of creating an IDS application is the data you use to train it. In the enterprise world, you have access to the company's data. In academia, however, good datasets for IDS are hard to find. One of the early attempts to solve that problem was initiated by DARPA, and it resulted in the DARPA 1999 IDS Dataset[6]. The dataset was a result of the work done by Stolfo et al.[19]. Even though the DARPA 1999 dataset has been a standard in academia for more than a decade, it has been heavily criticized as well. McHugh[20] has criticized the procedure used to generate the dataset and, more specifically, the validation process used for the validation set. Mahoney et al.[21] have found artifacts of the simulation used to create the datasets within the data undermining the credibility of performance results from intrusion detection algorithms. Moreover Ahmed et al[22] argue that the software used to create the dataset is no longer relevant and even at it's time it didn't have a significant market share. Lastly, the documentation of the dataset is questioned, with some researchers suggesting that the number of attacks present in the dataset is inaccurate.

Over time, other datasets started to emerge, but none has managed to become a standard. One reason for this is that it is difficult to create a dataset that maintains the quality standards needed to not be criticized for any of the shortcomings the DARPA datasets have. This is the case for the dataset I will be using as well. Another reason is that the diverse needs of an IDS system mean that different datasets address different aspects of the IDS landscape. The dataset used in this project addresses host-based intrusion detection (HIDS). It is called ADFA LD-12 and it was presented by Creech et al.[1]. I will describe it in detail in the next chapter.

[6]DARPA IDS 1998 - 1999 Datasets: https://www.ll.mit.edu/ideval/data/ - Accessed at 9 Aug 2016

VI. PROJECT PLANNING

An important part of any project is time management. In order to decide how to manage my time, I first had to see the situation I was in and the goals I wanted to achieve. While the MSc project supervisor has done previous work in the field of intrusion detection, he and his research team are not currently working on it. Therefore, part of the work of this project is to understand recent developments and trends in this field as well as lay the groundwork for further research in the field. This means that my work will not be focused on only one approach, but I will try many approaches to enhance my wider understanding of the subject. That doesn't mean I shouldn't have a specific goal, and this was to employ unsupervised learning outlier detection algorithms. There were also weekly meetings with the supervisor to report on the progress done and decide on the best way forward. Some of the meetings were with the wider research team, where I each presented the work I was doing to the other members.

To assist us in better management, I created the Gantt Chart shown in figure VI. It is separated in eight parts. Initially, I explored the dataset, set up my system, and decided on the architecture of my workflow. After that, I worked on replicating the results of previous papers[23], [24]. This work overlapped with the work on frequency-based algorithms and 2-sequence-based algorithms. Once I replicated the work of previous papers, I worked on new ways to analyze the dataset. One way of doing so was to get out of the boundaries established by the computer science community in handling the dataset and using it in ways more conventional to the machine learning community. After that, I proceeded to write the initial report and deliver a draft to the first supervisor. Subsequently, I used his feedback to improve my analysis and my report. Lastly, I prepared a presentation in order to present my work to the second supervisor.

An overall view of the velocity of my work can be seen in figure 2. It includes both my programming work done in Python as well as the writing of this report and presentation that were written in LaTeX. It does not include time spent

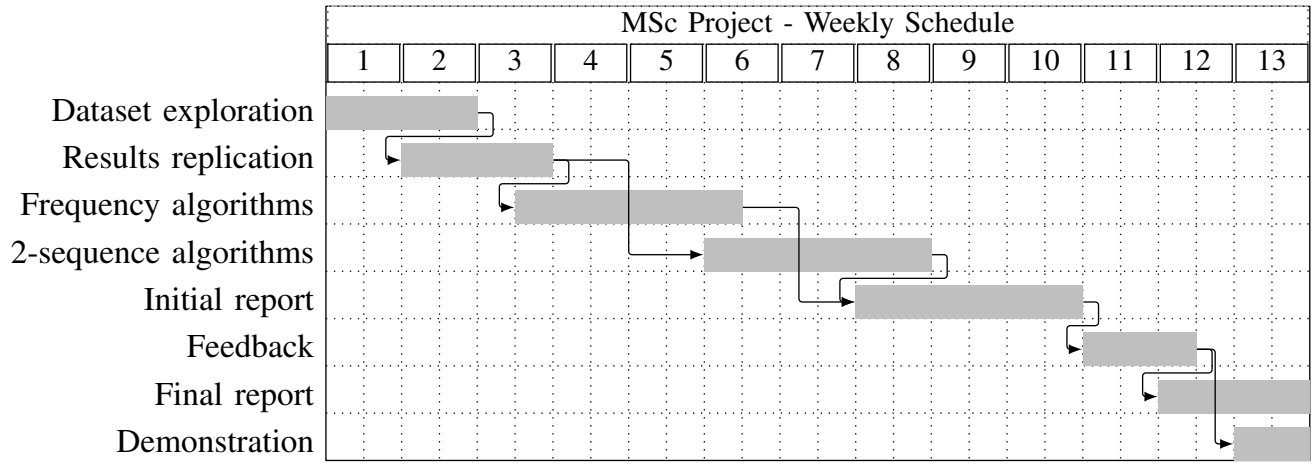

Figure 1. MSc Project Gantt Chart

in work outside of those two, such as computation time and literature review.

VII. Dataset Documentation

The dataset I will use for this project is a modern dataset. It was presented at the 2013 IEEE Wireless Communications and Networking Conference[1]. It uses modern software that simulates real user cases in the IT landscape. The data was produced by Creech et al.[1] by the following described below. Given their goal to simulate a typical real-world case, they used a common architectural configuration used in web servers called LAMP stack. LAMP means Linux, Apache, MySQL, and PHP, after the names of the core software this approach uses. Consequently, Ubuntu 11.04 is used as the server operating system for the creation of the dataset. In order to enable intrusion from the internet, Apache v2.2.17 and PHP v5.3.5 were also installed, and lastly, MySQL v14.14. Apart from that, file transfer protocol (FTP) secure shell (SSH) was enabled in order to simulate the remote administration of the server and the attack surface that comes with it. Their default configuration settings were used. Lastly, a web-based collaborative tool, Tiki Wiki v8.1, was installed and enabled. This version has a documented vulnurenability[7] that allows web exploitation. Those settings are representative of a local server offering basic web services on the internet. Tiki Wiki's known vulnerability represents the ever-present danger of previously unknown vulnerabilities on up-to-date software running on production infrastructure.

A program called *auditd* was used to monitor kernel system call traces. System call traces are API's provided by the kernel of the operating system for userspace applications to access. Ubuntu 11.04 uses the Linux kernel version 2.6.38, which has 325 system calls available[23]. The researchers who created the dataset used 6 different types of attacks to compromise their server. They are presented in table II which was taken from [1]. The dataset is separated into three sets: the training set, the attack set, and the validation set. The training and the validation sets contain sequences of system calls from the normal operation of the system, and the attack sets contain the intrusions performed by the creators of the dataset. Each individual attack method is carried out ten times. Each data point of the dataset consists of a series of system calls. Table III shows the number of data points per subset of ADFA-LD 12.

[7]Packet Storm: All things security, https://packetstormsecurity.com/files/108036/INFOSERVE-ADV2011-07.txt, Accessed 9 August 2016.

Payload/Effect	Vector
Password brute force	ftp by hydra
Password brute force	ssh by hydra
Add new superuser	Client side poison executable
Java-based interpreter	Tiki Wiki Vulnerability exploit
Linux meterpreter payload	Client side poison executable
C100 Webshell	Php remote file inclusion vulnerability

Table II
Attacks used in ADFA-LD 12 dataset.

Subset	Data points
Training	833
Validation	4372
Attack	719

Table III
Subsets of ADFA-LD 12 dataset.

The traditional approach on the cyber security field is to use the training set to train a model, use the attack set to find it's accuracy and use the validation set to find it's false positive rate. While I will use that schema I will not limit ourselves to it.

VIII. Preprocessing and Visualization

The first step in processing my dataset is to download it[8]. Subsequently I unzip the files I see that I create 3 folders for each subset of the dataset. The training and validation set have a long list of text files in their respective folders. Each text file represents a data point and contains a series of integers separated by white spaces that correspond to kernel system calls. The attack set is structured in folders according to the type of attack, and within them are the text files describing the attack data points.

The structure of the dataset after it is unzipped is not so helpful, so I transformed it into a format that will enable us to process it more easily. Some of the standard tools for this purple are pickle and numpy[9]. They are Python libraries that are used to save Python objects. My first preprocessing step is to create a Python dictionary for each of the subsets containing the information about the sequence of system calls for each data point. Then, I use pickle to save that object. Through this process, I create the following three pickle files:

1_training.p , 1_attack.p , 1_validation.p

After that the unzipped dataset files are deleted - they were too inefficient to use. The 3 new files are used to load the dataset for further computations.

my next step is to do a basic exploration of my dataset. The kernel of the host operating system provides 325 system calls, but I am not sure if all are represented and how that representation is distributed. As a first step, I run through my dataset once and create a set[10] where I add all the system calls I encounter. Thus, I end up with the following list of 175 system calls present in my dataset:
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 19, 20, 21, 22, 26, 27, 30, 33, 37, 38, 39, 40, 41, 42, 43, 45, 54, 57, 60, 61, 63, 64, 65, 66, 75, 77, 78, 79, 83, 85, 90, 91, 93, 94, 96, 97, 99, 102, 104, 110, 111, 114, 116, 117, 118, 119, 120, 122, 124, 125, 128, 132, 133, 136, 140, 141, 142, 143, 144, 146, 148, 150, 151, 154, 155, 156, 157, 158, 159, 160, 162, 163, 168, 172, 173, 174, 175, 176, 177, 179, 180, 181, 183, 184, 185, 186, 187, 190, 191, 192, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 219, 220, 221, 224, 226, 228, 229, 230, 231, 233, 234, 240, 242, 243, 252, 254, 255, 256, 258, 259, 260, 264, 265, 266, 268, 269, 270, 272, 289, 292, 293, 295, 296, 298, 300, 301, 306, 307, 308, 309, 311, 314, 320, 322, 324, 328, 331, 332, 340]
A curious thing to mention here is the presence of integers higher than 325. One would think that since I only have 325 system calls, only integers up to 325 would be used.[11] This is a very good example that no assumptions should be made when addressing a dataset and that I should always clean and validate the form of the data I expect to have for the next step of my pipeline. Another notable issue here is that I had not identified this problem initially when I was modeling the dataset. However, numerical tests performed after constructing the system called frequency feature space, about which I talked more later, showed that I had not captured all the dataset. This resulted in more rigorous exploration that revealed the unexpected labeling of the system calls.

Moving forward, I count the presence of each system call in my dataset. The result of this computation is presented on figures 3 and 4. As I see, I had to use a logarithmic scale for my y-axis because of how unevenly and spread out the distribution of my system calls is. Moreover, I observe that the norm is that system calls are present in all three subsets, hinting that there is little room for associating particular system calls with malicious behavior.

Another way to improve my basic understanding of my dataset is to look at its principal components. In order to compute the principal components, I merge the whole dataset. As I can see in figures 5 and 6 I have used different colors for the three subsets of ADFA-LD 12. In the figure 5, I only plot the training and attack sets. In figure 6 I also plot the validation set. By observing figure 5 I can see some degree of differentiation between the distributions of the attack and training set. However, once I add the validation set, in figure 6, I see that there is little differentiation between the attack dataset and the training and validation sets combined. The validation set's spatial distribution on the first two principal components is a bit different than the training set's. This means that the training and the validation sets as provided do not carry the same information content.

I can see how much of the variance of the dataset is captured with each principal component of the complete frequency space in table IV. It is worthwhile to mention some specifics about the variance of my datasets. In table IV, I present the variance of the complete dataset in the complete frequency feature space. In that case, one needs 12 dimensions to capture 80% of the variance of the dataset. However, in the work of Xie et al.[23] that I will reproduce in the next chapter, they perform principal component analysis on the training set only. This results in them needing only 9 dimensions in order to capture 80% of the variance of the training set. While I consider that choice sub-optimum because I want my principal components to contain the diversity of the attack dataset in order for a classifier to take advantage of it, I will use their approach in order to be able to compare my results with them. The percentage of the variance of the training set explained in each principal component, in this case, is presented in table V. In both vases I can see that the variance explained by each subsequent principal component decays slowly.

[8]Download url: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-IDS-Datasets/

[9]Pickle is the standard library to save python objects. However, when, later, I use numpy objects, it is better to save them with numpy instead of pickle as it is more efficient.

[10]A set, in Python, is an object with the useful property that it does not allow duplicate items.

[11]Nothing in the documentation of the dataset indicated this would (or would not) happen.

Principal Component	Explained Variance
1	0.2042
2	0.1233
3	0.0949
4	0.0777
5	0.0646
6	0.0490
7	0.0386
8	0.0346
9	0.0328
10	0.0303
11	0.0273
12	0.0260

Table IV
FRACTION OF EXPLAINED VARIANCE FROM THE PRINCIPAL COMPONENTS OF THE COMPLETE FREQUENCY SPACE THAT CONTAIN 80% OF THE VARIANCE OF ADFA-LD 12 DATASET.

Principal Component	Explained Variance
1	0.1969
2	0.1548
3	0.1130
4	0.0965
5	0.0709
6	0.0689
7	0.0517
8	0.0380
9	0.0293

Table V
VARIANCE OF THE PRINCIPAL COMPONENTS OF THE TRAINING SET.

IX. FREQUENCY ANALYSIS OF THE ADFA-LD 12 DATASET

I now proceed deeper in my analysis of the ADFA-LD 12 dataset. My first goal is to replicate the results produced by Xie et al.[23]. The paper focuses on frequency-based feature engineering, which is similar to the one I used to explore my dataset. Their analysis is based on two algorithms: k-nearest neighbors and k-means clustering. As I mentioned, a noteworthy part of [23] is that the authors perform principal component analysis not on the whole of the dataset but only on the training set. I will follow their approach while I am trying to replicate their results in the following sections, even though this means my reduced frequency feature space does not capture the dataset as efficiently as it could. This way ensures that I can verify that my replication has been successful. After that, I can move into my own approach.

X. K - NEAREST NEIGHBOURS

I now describe the kNN implementation of Xie et al.[23], which I will replicate.

They begin by performing feature reduction through principal component analysis. The criterion for classifying a data point as normal or not is whether it has more than k data points from the training set within a distance d. The parameters k and d were chosen empirically by the authors. For k they chose 20. Their choice of parameter d depends on the distance metric used. I present their choices[23] in table VI. The change in distance parameter d allows us to calibrate the sensitivity when identifying a data point as abnormal. The bigger the distance, the more likely I am to find k points and call that data point normal. This calibration procedure allows us to construct the ROC curves. After performing my computation, I plot the receiver operating characteristic curves that have also been plotted by Xie et al.[23][12]. I can see them in figures 7 and 8.

Metric	distance (d)	step width
squared euclidean	[0.01, 0.1]	0.01
standardised squared euclidean	[1, 10]	1

Table VI
K-NEAREST NEIGHBOURS PARAMETERS USED IN DIFFERENT ITERATIONS OF THE ALGORITHM IN ORDER TO CREATE THE ROC CURVES FOR KNN ALGORITHM IN THE REDUCED FREQUENCY FEATURE SPACE.

Attack Used	Area under ROC curve
adduser	0.745
hydra ftp	0.593
hydra ssh	0.549
java meterpreter	0.727
meterpreter	0.710
web shell	0.734

Table VII
AREA UNDER THE CURVE FOR SQUARED EUCLIDEAN DISTANCE KNN CLASSIFIER IN THE REDUCED FREQUENCY SPACE.

To assess the performance of my classifiers, I compute the area under the ROC curve. The results are shown in table VII. Calculating the area under the ROC curve for kNN under squared standardized Euclidean distance, I get the results shown in table VIII.

I see that the password cracking attacks (hydra ssh and hydra ftp) are quite harder to identify compared to the other attacks, and my classifier performed poorly on them. For the other attacks, my classifier performed moderately to well overall.

XI. K - MEANS CLUSTERING

I proceed in trying to replicate the results of Xie et al.[23] with regards to the k-means clustering algorithm. I focus on implementing the algorithm with a Euclidean distance metric. The authors used the training data to create 5 clusters. The number of clusters was chosen empirically. A new data point

[12]I should note here that I modified the procedure a bit. Xie et al[23] in their figures do not include the 100% False Positive and True Positive point as well as the 0% True Positive and False Positive point. Thus, they cannot move forward with computing the area under the ROC curve. I added this trivial step in my implementation in order to have a measure that I can use to evaluate and compare results from different algorithms.

Attack Used	Area under ROC curve
adduser	0.696
hydra ftp	0.574
hydra ssh	0.518
java meterpreter	0.689
meterpreter	0.705
web shell	0.697

Table VIII
AREA UNDER THE CURVE FOR SQUARED STANDARDIZED EUCLIDEAN DISTANCE KNN CLASSIFIER IN THE REDUCED FREQUENCY SPACE.

Attack Used	Area under ROC curve
adduser	0.6893
hydra ftp	0.6428
hydra ssh	0.4690
java meterpreter	0.6858
meterpreter	0.7475
web shell	0.7158

Table IX
AREA UNDER THE CURVE FOR K-MEANS CLUSTERING CLASSIFIER USING EUCLIDEAN DISTANCE ON THE REDUCED FREQUENCY FEATURE SPACE.

Attack Used	Regularisation parameter
adduser	10
hydra ftp	0.05
hydra ssh	0.5
java meterpreter	0.1
meterpreter	10
web shell	1

Table X
OPTIMAL REGULARISATION PARAMETER FOR SVM'S WITH LINEAR KERNEL ON ADFA-LD 12 FOR DIFFERENT ATTACK PROFILES IN THE REDUCED FEATURE SPACE.

Attack Used	Area under ROC curve
adduser	0.8303
hydra ftp	0.6978
hydra ssh	0.7801
java meterpreter	0.8628
meterpreter	0.9105
web shell	0.8354

Table XI
AREA UNDER THE CURVE FOR SVM'S WITH LINEAR KERNEL ON ADFA-LD 12 FOR DIFFERENT ATTACK PROFILES ON THE REDUCED FEATURE SPACE.

was classified as normal when it was within a distance d of a cluster center. The distance parameter was chosen by finding the maximum in cluster distance (d_{max}) and getting an evenly spaced sample of 10 numbers from $[0, d_{max}]$. In my case, the maximum distance is 0.7551. This calibration of the distance parameter allows us to control the sensitivity with which I classify a data point as anomalous, enabling the creation of receiver operating characteristic curves.

After performing my computation, I plot the receiver operating characteristic curves. They are presented in figure 9. To assess the performance of my classifier, I compute the area under the ROC curves and present the results in table IX. I can see that in this case, the hydra ssh attack has bad performance, while for the other attacks, the classifier has moderate performance.

The authors of [23] do not comment on the area under the ROC curves for the classifiers they used. Instead, they just assess individual Precision (True Positive) and Fallout (False Positive) rates. With my implementation, I can compute it. By looking through figures 7, 8, 9 and the tables presenting the area under the ROC curve, VI, VII, IX, I can see that the k-means clustering algorithm is a bit more reliable. Additionally k-means clustering is a lot cheaper computationally. The training times for k-means clustering were at least one order of magnitude less than those of k-nearest neighbours on the computing resources I were using for this work.

As a final remark for using k-means clustering for anomaly detection, I should mention that conceptually, it has a lot of similarities to the Gaussian Mixture Model pioneered by Roberts et al.[5].

XII. SUPPORT VECTOR MACHINES

A. Linear SVM on reduced frequency feature space

A natural step in continuing my analysis of the ADFA-LD 12 dataset further than Xie et al.[23] are Support Vector Machines. As a first, I will keep the methodology I used previously, meaning I will work with the first nine principal components of the training set so I can compare my results before moving forward. I will train an SVM model, using a linear Kernel, for each attack separately. More Specifically, I will use the SVC class from the scikit-learn library[28]. Moreover I will use different values of the regularisation parameter C to better map the hypothesis space of available classifiers. The different values of C I used are:

$$[0.05, 0.1, 0.5, 1, 5, 10, 50]$$

I can see the first results of this computation in figure 10 where I plot various performance points on a True Positive versus False positive map. From there, I conclude that for some attacks, namely web shell and meterpreter, the performance of the SVM classifier is not influenced significantly by the choice of regularisation parameter. On the other hand, hydra ftp and hydra ssh attacks are significantly influenced. For each attack, I note the best-performing value, as I can see in table X.

I proceed with computing the optimum linear SVM classifier for each attack and then plot its receiver operating characteristic curve. The results are shown in figure 11. The area under the curve for each classifier is shown in table XI. As usual, the hydra attacks are my worst performers, while the meterpreter attacks are my best performers.

At this stage, I am able to compare my results, from table XI, with those of the k-means clustering algorithm, from table IX. It is quite clear that Support Vector Machines are a more accurate and reliable classifier for every attack. On average, the SVM classifier has 0.15 higher area under the curve.

Moving forward, I want to see the performance of Support Vector Machines on the dataset as a whole instead of training them specifically for each attack. Hence I will treat the problem as a two class classification problem with my classes being normal behavimy and attack behaviour. To get a better idea of the performance of my classifier in this case, I will use cross-validation. To address the issue of class imbalance in my dataset, I will use stratified 8-fold cross-validation[13].

[13]I chose 8-fold cross-validation instead of 10-fold due to the class imbalance against my attack set and the inner divisions within it. The change is minor but helps us sample the attack set better across folds.

Regularisation (C)	Precision	Fall out
0.125	0.62 ± 0.08	0.20 ± 0.03
0.25	0.62 ± 0.08	0.20 ± 0.03
0.5	0.62 ± 0.08	0.19 ± 0.03
1	0.62 ± 0.08	0.19 ± 0.04
2	0.61 ± 0.09	0.19 ± 0.03
4	0.63 ± 0.07	0.20 ± 0.03
8	0.64 ± 0.08	0.21 ± 0.04
16	0.64 ± 0.10	0.23 ± 0.05
32	0.64 ± 0.11	0.24 ± 0.06

Table XII
8-FOLD STRATIFIED CROSS VALIDATION RESULTS FOR VARIOUS REGULARISATION PARAMETER VALUES OF SVM CLASSIFIER WITH LINEAR KERNEL. RESULTS ARE PRESENTED WITH MEAN AND STANDARD DEVIATION FOR TWO METRICS. TRUE POSITIVE RATE (PRECISION) AND FALSE POSITIVE RATE (FALL OUT). REDUCED FREQUENCY FEATURE SPACE WAS USED FOR TRAINING AND VALIDATION.

Regularisation (C)	Precision	Fall out
0.125	0.62 ± 0.10	0.17 ± 0.04
0.25	0.64 ± 0.09	0.17 ± 0.04
0.5	0.66 ± 0.09	0.16 ± 0.04
1	0.68 ± 0.06	0.16 ± 0.04
2	0.76 ± 0.07	0.19 ± 0.05
4	0.81 ± 0.08	0.19 ± 0.04
8	0.91 ± 0.04	0.18 ± 0.03
16	0.91 ± 0.05	0.18 ± 0.03
32	0.92 ± 0.04	0.18 ± 0.03
64	0.93 ± 0.05	0.16 ± 0.04
128	0.92 ± 0.06	0.16 ± 0.04

Table XIII
8-FOLD STRATIFIED CROSS-VALIDATION RESULTS FOR VARIOUS REGULARISATION PARAMETER VALUES OF SVM CLASSIFIER WITH LINEAR KERNEL. RESULTS ARE PRESENTED WITH MEAN AND STANDARD DEVIATION FOR TWO METRICS. TRUE POSITIVE RATE (PRECISION) AND FALSE POSITIVE RATE (FALL OUT). COMPLETE FREQUENCY FEATURE SPACE WAS USED FOR TRAINING AND VALIDATION.

Going even further, I run my simulation for various values of the regularisation parameter. I present my results in table XII. I can see that I get better performance[14] for low regularisation values. The optimum value is $C = 0.5$.

B. Linear SVM on the complete frequency feature space

Moving further away from the approach of Xie et al.[23] I investigate the application of support vector machines in the frequency feature space without reducing it's dimensionality through principal component analysis. I pool all of my dataset together and perform 8 fold stratified cross validation training a linear SVM classifier. I present my results in table XIII.

As I can see by comparing tables XII and XIII using the complete frequency feature space yields significantly better results. The precision rate is increased noticeably, and the fallout rate is slightly decreased. The computation cost was not noticeably increased for the whole frequency feature space when running my scripts.

C. Recursive feature elimination

[14]Increased performance means higher difference between precision and fallout rates, with 1 being perfect classification results.

Moving forward with my analysis of the ADFA-LD 12 dataset, I will conduct recursive feature elimination. I will train an SVM classifier with a linear kernel for a two-pattern classification problem. My two classes will be attack and normal behavior. A property of k-fold cross-validation is that only a small part of the dataset is used as a validation set, meaning my under-represented attack class may not be appropriately represented in the validation set. For this reason, I will use a sampling method to perform cross-validation. I will be sampling my dataset with the StratifiedShuffleSplit method of scikit-learn[28] 6 times, and my training and validation sets will be equal in size. To create a scorer and assess the performance of my classifier, I will use the Precision (True Positive rate) and Fall out (False Positive rate) ratios. But because recursive feature elimination works by optimising one measure, I will use their difference, namely:

$$\text{Scorer} = \text{Precision} - \text{Fallout}$$

When Scorer=1 I have perfect classification and if Scorer is 0 I are classifying everything as attack behaviour. After writing my code and performing the necessary computations, I plot my results in figure 12.

By looking at the plot, I see that when I start removing the less relevant features from the complete feature space, my performance, measured by my scorer metric, remains steady. That plateau remains with minimal variation until I have 50 remaining features with a global maximum of 54 features. Then it has a small decline that keeps increasing and becomes a steep decline after I am left with only 25 features. Therefore I can deduct that the information content required to perform proper identification of attacks in the context of a two pattern classification problem is contained in the 54 most relevant features.

XIII. ONE CLASS SUPPORT VECTOR MACHINES - OUTLIER DETECTION

Before I finish my analysis of ADFA-LD 12 on the frequency feature space, I will study one class, Support Vector Machines. The 1-class SVM algorithm has been introduced by Schölkopf et al.[6] and can work with a variety of kernel functions just like normal SVM classifiers. From my preliminary results, I saw that linear kernels performed badly. After various trials, I found optimum performance on sigmoid kernels with the parameters $\gamma = 0.05$ and $c_0 = 3$.

One more tricky thing about the 1-class SVM classifier is that on the training set, I am asked to specify the upper bound for the fraction of training data that I can tolerate being incorrectly classified. In order to study the upper bound's (ν) effects, I will test its performance by recursively performing cross-validation on a range of values for ν. The results of my first test can be seen in figure 13 and table XIV.

By studying the results of figure 13 and table XIV, I see that there is big volatility for different numbers of ν. Moreover, the standard deviation of my metrics is quite variable, which is something I didn't expect. This might be due to the relatively small number, 10, of re-sampling or because of

ν	Precision	Fall out
0.1	0.55 ± 0.10	0.19 ± 0.05
0.2	0.53 ± 0.04	0.22 ± 0.03
0.3	0.59 ± 0.02	0.30 ± 0.02
0.4	0.82 ± 0.04	0.41 ± 0.01
0.5	0.89 ± 0.01	0.49 ± 0.02
0.6	0.910 ± 0.001	0.61 ± 0.01
0.7	$0.912 \pm 1 \cdot 10^{-16}$	0.73 ± 0.10
0.8	$0.91 \pm 6 \cdot 10^{-4}$	0.81 ± 0.10
0.9	0.93 ± 0.007	0.904 ± 0.009
1.0	1.0 ± 0.0	1.0 ± 0.0

Table XIV
EXPLORING 1-CLASS SVM WITH A SIGMOID KERNEL TO ASSESS ITS PERFORMANCE DEPENDING ON THE UPPER BOUND FOR THE FRACTION OF TRAINING ERRORS. SHUFFLESPLIT METHOD WAS USED FOR CROSS-VALIDATION TO CREATE TWO, EQUAL IN SIZE, NORMAL BEHAVIOR DATASETS FOR TRAINING AND VALIDATION. OPTIMAL PERFORMANCE FOR UPPER BOUND $\nu = 0.4$.

ν	Precision	Fall out
0.35	0.67 ± 0.04	0.36 ± 0.01
0.36	0.68 ± 0.04	0.37 ± 0.01
0.37	0.71 ± 0.06	0.39 ± 0.02
0.38	0.75 ± 0.04	0.40 ± 0.01
0.39	0.77 ± 0.04	0.40 ± 0.01
0.40	0.83 ± 0.04	0.41 ± 0.02
0.41	0.85 ± 0.01	0.42 ± 0.02
0.42	0.85 ± 0.02	0.42 ± 0.02
0.43	0.858 ± 0.001	0.431 ± 0.010
0.44	0.862 ± 0.002	0.449 ± 0.006
0.45	0.864 ± 0.004	0.45 ± 0.01

Table XV
EXPLORING 1-CLASS SVM WITH A SIGMOID KERNEL TO FIND ITS OPTIMUM PERFORMANCE DEPENDING ON THE UPPER BOUND FOR THE FRACTION OF TRAINING ERRORS. THE SHUFFLESPLIT METHOD WAS USED FOR CROSS-VALIDATION TO CREATE TWO EQUAL-IN-SIZE, NORMAL-BEHAVIOR DATASETS FOR TRAINING AND VALIDATION. OPTIMAL PERFORMANCE FOR UPPER BOUND $\nu = 0.41$.

some structure within my dataset that I haven't identified yet. Further investigation is needed to determine this. In order to identify the best-performing value of the upper bound for misclassification, I repeat my procedure on a smaller range of values. The results of my computation are presented in figure 14 and table XV.

From table XV, I can identify $\nu = 0.41$ as the best-performing value for the upper bound on training errors. Moreover, from looking at both tables XIV and XV, I can see that the fallout rate follows the value of ν, which makes sense given that the training data are all from normal behavior and the fallout rate represents misidentification of normal data.

Because the LIBSVM library[30] implementing the One-class SVM algorithm that I use does not predict probabilities for a data point to be classified as outlier but only classifies them as such I cannot create ROC curves. However, by observing the performance of the algorithm depending on the bound for the training error at figures 13 and 14, I see that there is a resemblance to a receiver operating characteristic.

Regularisation (C)	Precision	Fall out
0.125	0.93 ± 0.02	0.068 ± 0.010
0.25	0.93 ± 0.02	0.062 ± 0.009
0.5	0.92 ± 0.02	0.054 ± 0.009
1	0.91 ± 0.02	0.049 ± 0.007
2	0.91 ± 0.02	0.047 ± 0.006
4	0.88 ± 0.03	0.046 ± 0.007
8	0.86 ± 0.03	0.043 ± 0.006
16	0.84 ± 0.04	0.041 ± 0.005
32	0.81 ± 0.03	0.038 ± 0.004

Table XVI
8-FOLD STRATIFIED CROSS-VALIDATION RESULTS FOR VARIOUS REGULARISATION PARAMETER VALUES OF SVM CLASSIFIER WITH LINEAR KERNEL. RESULTS ARE PRESENTED WITH MEAN AND STANDARD DEVIATION FOR TWO METRICS. TRUE POSITIVE RATE (PRECISION) AND FALSE POSITIVE RATE (FALL OUT). TWO-SEQUENCE FEATURE SPACE WAS USED FOR TRAINING AND VALIDATION.

XIV. 2-Sequence Analysis of the ADFA-LD 12 Dataset

After my analysis of the ADFA-LD 12 dataset on the frequency feature space, I proceed to analyze it on the two-sequence feature space.

I begin by giving a specific definition of the two sequence feature space I will use. As I have said, every point of my dataset consists of a series of kernel operating system calls. Instead of counting the frequency of a individual system calls in each point I will count the frequency of combinations of two subsequent system calls in each point. From my previous explorations of my dataset, I know that I have 175 distinct system calls present. Out of the 30625 possible combinations of 2-sequences, only 3792 are present in my dataset. Hence, my two-sequence feature space has 3792 dimensions.

XV. Support Vector Machines

I start my analysis of the two-sequence feature space by addressing my data as a two-pattern classification problem. I will use a linear kernel for my SVM classifier. I will test its performance on various values of the regularisation parameter (C) performing stratified cross-validation. After computing the necessary calculations, I get the results shown in table XVI.

I see that I get my best-performing results when I have $C = 0.25$. By comparing the results from tables XIII and XVI I see that the two-sequence feature space yields significantly better results compared to the full frequency feature space. This is to be expected since the two frequency feature spaces capture more information from the dataset.

Moving forward, I will conduct recursive feature elimination to see how much information is contained in each feature. I will use the same procedure as before. I will train my classifier to distinguish a two-pattern classification problem. I will use the Stratified Shuffle Split method to perform cross-validation. In order to make the computation time reasonable, I will use a reduction step of 24 features (out of a total of 3792) per iteration. After performing my computation, I get the results depicted in figure 15. I see that performance plateaus at 240 features and stays relatively stable afterward, with an obtuse maximum at 2088 features. Below 240 features performance

ν	Precision	Fall out
0.1	0.55 ± 0.03	0.20 ± 0.01
0.2	0.58 ± 0.01	0.245 ± 0.009
0.3	0.72 ± 0.02	0.32 ± 0.02
0.4	0.826 ± 0.004	0.41 ± 0.02
0.5	0.876 ± 0.006	0.50 ± 0.01
0.6	0.9269 ± 0.0008	0.64 ± 0.07
0.7	0.930 ± 0.001	0.76 ± 0.09
0.8	0.936 ± 0.004	0.86 ± 0.07
0.9	0.9560 ± 0.0005	0.909 ± 0.007
1.0	1.0 ± 0.0	1.0 ± 0.0

Table XVII
EXPLORING 1-CLASS SVM WITH A SIGMOID KERNEL TO ASSESS ITS PERFORMANCE DEPENDING ON THE UPPER BOUND FOR THE FRACTION OF TRAINING ERRORS. THE DATASET WAS FEATURE-ENGINEERED ON A TWO-SEQUENCE FEATURE SPACE. THE SHUFFLESPLIT METHOD WAS USED FOR CROSS-VALIDATION TO CREATE TWO EQUAL-IN-SIZE, NORMAL-BEHAVIOR DATASETS FOR TRAINING AND VALIDATION. OPTIMAL PERFORMANCE FOR UPPER BOUND $\nu = 0.4$.

ν	Precision	Fall out
0.35	0.788 ± 0.008	0.366 ± 0.007
0.36	0.805 ± 0.008	0.379 ± 0.012
0.37	0.807 ± 0.006	0.380 ± 0.013
0.38	0.820 ± 0.008	0.398 ± 0.011
0.39	0.822 ± 0.003	0.403 ± 0.012
0.40	0.827 ± 0.003	0.410 ± 0.010
0.41	0.832 ± 0.005	0.428 ± 0.018
0.42	0.832 ± 0.004	0.424 ± 0.013
0.43	0.835 ± 0.004	0.435 ± 0.011
0.44	0.839 ± 0.005	0.448 ± 0.012
0.45	0.849 ± 0.006	0.464 ± 0.017

Table XVIII
EXPLORING 1-CLASS SVM WITH A SIGMOID KERNEL TO FIND ITS OPTIMUM PERFORMANCE DEPENDING ON THE UPPER BOUND FOR THE FRACTION OF TRAINING ERRORS. THE SHUFFLESPLIT METHOD WAS USED FOR CROSS-VALIDATION TO CREATE TWO EQUAL-IN-SIZE, NORMAL-BEHAVIOR DATASETS FOR TRAINING AND VALIDATION. OPTIMAL PERFORMANCE FOR UPPER BOUND $\nu = 0.36$.

drops significantly, hence I can say that the core information content on normal and attack classes is contained in the 240 most important features.

XVI. ONE CLASS SUPPORT VECTOR MACHINES - OUTLIER DETECTION

The last step of my analysis of ADFA-LD 12 on the two-sequence feature space is studying one class, Support Vector Machines. Again, from my preliminary results, I saw that linear kernels performed badly. On the other hand the sigmoid kernel gives us good performance. In order to get results directly comparable to the ones on the frequency feature space, I will use the same parameters that I used before, $\gamma = 0.05$ and $c_0 = 3$.

As I have already said, one more tricky thing about the 1-class SVM classifier is that on the training set, I am asked to specify the upper bound for the fraction of training data that I can tolerate being incorrectly classified. In order to study the upper bound's (ν) effects on the two-sequence feature space, I will test its performance by recursively performing cross-validation on a range of values for ν. The results of my first test can be seen on figure 16 and table XVII.

While figure 16 gives us a good overview of the performance of one class support vector machine the iteration step, 0.1, for ν (maximum error ratio tolerance for training set) is too big to tell us the optimum value. Hence I re-iterate with smaller iteration step, 0.01, for values between $\nu \in (0.35, 0, 45)$ to find the best performing value. I see my results for $\nu = 0.36$.

XVII. VALIDATION AND TESTING

Before I do that, however, it is important to mention a key part of my work that has been lurking in the shadows. This is validation and testing. All of my numerical analysis has been done through computations. It is easy to make a mistake that introduces a bug in the Python code, but it is very hard to identify that by looking at the results. Therefore, I took the approach that every step of a numerical calculation, once implemented, should be tested through the form of debug messages printed by the terminal or my log files in order to verify them. This approach has not only helped us identify bugs in my code immediately, but it has also shaped how my code was written to make it more resilient.

A key success of that testing was the proper identification of the integers representing the kernel system calls. Initially, through manual exploration of the data files, sampled randomly, I had assumed that they were represented by the integers from 1 to 325. Numerical testing, in this case mostly through summation of the relevant probabilities, revealed that something was missing. A more rigorous exploration of the dataset revealed the problem.

XVIII. RESULTS ANALYSIS

My first goal was to replicate the results of Xie et al.[5]. I successfully achieved that and implemented some trivial improvements that allowed us to fully compute the ROC curve and the area below it.

I then moved further by implementing Support Vector Machines classifiers. Comparing the area under the ROC curves for the various cases described in tables VII, VIII, IX, and XI is not very straightforward, so I created figure 18 to condense that information. I can see that k-means clustering and k-nearest neighbors with squared standardized Euclidean distance are the worst performers, while support vector machines perform a lot better.

I then moved to the complete frequency feature space. When formulating the problem as a two pattern classification I saw, from tables XII and XIII, a significant improvement in performance. This means that the feature reduction approach used by Xie et al.[5] was not efficient in maintaining the information contained in the dataset while reducing its dimensionality. To that extent, I tried recursive feature elimination, and I saw that I could eliminate up to 121 out of 175 dimensions of the feature space with no loss of performance. As I can see from figure 12, I could even go below that, and unless I keep less than the 35 most informative features, I am not sacrificing much on the performance of my algorithm.

I then moved to try an unsupervised learning approach using one-class support vector machines, where I saw a significant drop in performance, as evident from my results on tables XIV and XV.

My next step was to move to a feature space that captured more information from the dataset, the two-sequence feature space. To compare how much my performance improved, I created figure 19, which takes information from tables XIII and XVI. As an evaluation metric I use the scored I have defined earlier as the difference between precision and fall out rate.

Last but not least, I compare the performance of one class support vector machine. This algorithm acts as outlier detection, and from what I see from figures 19, 20, its performance is significantly poorer compared to support vector machines classifier. I can also see that there is very little difference between the two feature spaces. This is in contrast with my previous results when intrusion detection was framed as a two-pattern classification problem.

XIX. Further Research

This work was intended to initiate research on the the field of intrusion detection through modern machine learning approaches. As such it leaves a lot of open questions. One straightforward path to go forward is to take note of the system calls identified as more relevant from recursive feature elimination. Someone with expertise in computer science and, more specifically, with the Linux kernel can give us domain input and provide useful information for more efficient feature engineering.

Another avenue for further research is to check on the scalability of the algorithms I am using. The AWID 2015 dataset by kolias et al.[25] is a very good candidate. Its compressed size is 10Gb, making it a very good candidate for assessing algorithms that are to be trained in a distributed setting or high-performance computer systems. It is a modern dataset using tools representative of the current IT landscape, and results on it will have applications on current networks.

Last but not least, there is great room for improvement in using better kernels in unsupervised outlier detection. The fact that I saw very small performance increase by moving to the two-sequence feature space means that I not training the one-class support vector machine algorithm on a feature space that it can take full advantage of. Taking advantage of kernel methods has the potential to greatly improve performance. And it even if it doesn't, it is useful to know the extent of each algorithm's efficiency. This enhances my overall understanding in the Machine Learning field and helps us find areas in need of novel approaches.

References

[1] G. Creech, J. Huy *Generation of a new IDS Test Dataset: Time to Retire the KDD Collection*, 2013 IEEE Wireless Communications and Networking Conference (WCNC)

[2] M. Schellekens *Car hacking: Navigating the regulatory landscape*, Computer Law & Security Review 32 (2016) 307 - 315

[3] C. Sun, C. Liu and J. Xie *Cyber-Physical System Security of a Power Grid: State-of-the-Art*, MDPI - Electronics open access journal.

[4] I. Raghav, S. Chhikara, N. Hasteer *Intrusion Detection and Prevention in Cloud Environment: A Systematic Review*, Journal of Network and Computer Applications, Volume 36, Issue 1, January 2013, Pages 25-41

[5] S. Roberts and L. Tarassenko, *A probabilistic resource allocating network for novelty detection*, Neural Computation, vol. 6, no. 2, pp. 270 - 284, 1994.

[6] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, R. Williamson, *Estimating the Support of a High-Dimensional Distribution*, Neural Computation, Vol. 13, No. 7, pp. 1443-1471, 2001.

[7] P. Hayton, B. Scholkopf, L. Tarassenko, and P. Anuzis, *Support vector novelty detection applied to jet engine vibration spectra*, in NIPS, pp. 946 - 952, 2000.

[8] L. Clifton, H. Yin, and Y. Zhang, *Support Vector Machine in Novelty Detection for Multi-channel Combustion Data*, Proceedings of the third international conference on Advances in Neural Networks - Volume Part III (2006)

[9] B. Farran, C. Saunders and M. Niranjan, *Machine Learning for Intrusion Detection: Modeling the Distribution Shift*, 2010 IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2010)

[10] M. Bhuyan, D. Bhattacharyya, and J. Kalita (2014) *Network Anomaly Detection: Methods, Systems and Tools*, IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, 2014

[11] M. Ahmed, A. Mahmood, J. Hu *A survey of network anomaly detection techniques*, Journal of Network and Computer Applications. Vol. 60, January 2016, p. 19-31

[12] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, *Bayesian event classification for intrusion detection*, in Proc. 19th Annual Computer Security Applications Conference, 2003

[13] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, *RODD: An Effective Reference-Based Outlier Detection Technique for Large Datasets*, in Advanced Computing. Springer, 2011, vol. 133, pp.76–84.

[14] I. Kang, M. K. Jeong, and D. Kong, *A differentiated one-class classification method with applications to intrusion detection*, Expert Systems with Applications, vol. 39, no. 4, pp. 3899–3905, March 2012.

[15] X. Xu, *Sequential anomaly detection based on temporal-difference learning: Principles, models and case studies*, Applied Soft Computing, vol. 10, no. 3, pp. 859–867, 2010

[16] M. Amini, R. Jalili, and H. R. Shahriari, *RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks*, Computers & Security, vol. 25, no. 6, pp. 459–468, 2006

[17] A. Borji, *Combining heterogeneous classifiers for network intrusion detection*, in Proc. 12th Asian Computing Science Conference on Advances in Computer Science: Computer and Network Security. Springer, 2007, pp. 254–260.

[18] C. Aggarwal, A. Hinneburg, and D. Keim, *An empirical evaluation of supervised learning in high dimensions*, ICML '08: Proceedings of the 25th International Conference on Machine learning Pages 96-103

[19] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, *Cost-Based Modeling for Fraud and Intrusion Detection: Results from the JAM Project*, in Proc. DARPA Information Survivability Conference and Exposition, vol. 2. USA: IEEE CS, 2000, pp. 130–144.

[20] J. McHugh, *Testing intrusion detection systems: A critique of the 1998 and 1999 Darpa intrusion detection system evaluations as performed by Lincoln laboratory*. ACM Transactions on Information Systems and Systems Security, Volum 3, Issue 4, pages 262 - 294, 2000

[21] M. Mahoney and P. Chan, *An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection*, Recent Advances in Intrusion Detection, Volume 2820 of the series Lecture Notes in Computer Science pp 220-237

[22] M. Ahmed, A. Mahmood, J. Hu, *A survey of network anomaly detection techniques*, Journal of Network and Computer Applications. Vol. 60, January 2016, p. 19-31

[23] M. Xie, J. Hu, X. Yu, and Elizabeth Chang *Evaluating Host-Based Anomaly Detection Systems: Application of the Frequency-Based Algorithms to ADFA-LD*, 11th International Conference on Fuzzy Systems and Knowledge Discovery, 2014

[24] M. Xie, J. Hu and J. Slay *Evaluating Host-based Anomaly Detection Systems: Application of the One-class SVM Algorithm to ADFA-LD*, Proceedings of the 11th IEEE International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2014), Xiamen, 19-21 August 2014, 978-982.

[25] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis *Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a*

Public Dataset IEEE Communication Surveys & Tutorials, Vol. 18, No. 1, 2016

[26] G. van Rossum, *Python tutorial, Technical Report CS-R9526*, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[27] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin, and T. Oliphant, *Numerical Python* Lawrence Livermore National Laboratory, UCRL-MA-128569, 1999

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot , E. Duchesnay *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, volume 12, pages 2825 - 2830, 2011

[29] J. Hunter, *Matplotlib: A 2D graphics environment*, Computing In Science & Engineering, vol. 9, no. 3, pp 90 - 95, 2007

[30] C. Chih-Chung, L. Chih-Jen, *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, Volume 2, Issue 3 (2011)