University of Southampton

Faculty of Physical Sciences and Engineering

School of Electronics and Computer Science

Intrusion Detection using Outliers in a modern Cyber-Security Dataset

by

Nikolaos Perrakis

2 September 2016

Supervisor: prof. Mahesan Niranjan
2nd Examiner: prof. Jonathon Hare

A report submitted for the award of

MSc Data Science

University of Southampton

Faculty of Physical Sciences and Engineering

School of Electronics and Computer Science

Intrusion Detection using Outliers in a modern Cyber-Security Dataset

by

Nikolaos Perrakis

2 September 2016

Supervisor: prof. Mahesan Niranjan
2nd Examiner: prof. Jonathon Hare

A report submitted for the award of

MSc Data Science

Abstract

In this project we study outlier detection in an intrusion detection dataset. In order to base our research on solid foundation we first review the relevant machine learning literature on outlier detection as well as the information security literature on intrusion detection techniques. We did a short review of the available datasets in the literature to show what one should look for when searching for an intrusion detection dataset. After that we start exploring the dataset performing exploratory data analysis. This enables us to replicate previous work done on the dataset. We then expanded on that work exploring the full potential of the frequency feature space using support vector machines. We get results with better performance and then move on implementing an outlier detection algorithm, one-class support vector machines. We see a drop in performance as we would expect from moving from a supervised to an unsupervised learning approach. Then we move to the next step of our analysis. We construct the two-sequence feature space and use support vector machines as a two pattern classification problem. We see that we get significantly better results compared to the frequency space. On the other hand we see only minor improvements when we use the outlier detection algorithm. This leads up to conclude that there many open ended research questions on this dataset in the two-sequence feature space and unsupervised outlier detection.

Acknowledgements

I would like to express my gratitude to professor Mahesan Niranjan for his guidance, critical appraisal and insight. I would also like to thank my parents and my brother for their great support during this difficult year. Last but not least I would like to thank all my colleagues on the MSc of Data Science for the camaraderie we have built this year.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In the industry we 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 our life and society are exposed online making cyber-security even more important. During the last couple of years we have seen many examples of this danger.

Further expanding on the example of cars we 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 aforementioned 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

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

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 we 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**].

CHAPTER 2

Foundations of the project

In this project we are drawing knowledge from two related but distinct fields. [1] 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 we will describe the foundations of this project. They have been studied to a big extend during the Project preparation course.

2.1. 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 we 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 our work.

The first method has been put forward by Roberts et al.[**5**]. They use a Gaussian Mixture model to map the normal behaviour of the system their dataset describes. This creates a set of normal system states their trained model recognises. In their method they minimise 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 learnt normal state before it gets classified as belonging to another state. When we are training out model, with normal operation data, this results in a new learnt normal state. Afterwards 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

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

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 Schlkopf 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 more accurately train their model. Moreover during the discussion of their results Hayton et al. give 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 put 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 behaviour 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 One 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 recognise anomalous behaviour. 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 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[2]. Their technique uses two steps. The first is using a method called Voted Spheres that runs over the the dataset once and performs non-parametric classification. It results in partitioning the feature space in 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. The 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 it's reliance in quadratic programming something that may negate the low computation load of the Voted Spheres method.

2.2. 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 our answer to that threat. The basic assumption we make is that our 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 we did during Project Preparation[3] we will use the classification used in Bhuyan et al.[**10**] and Ahmed et al.[**11**]. We classify intrusions as Malware attacks, Denial of Service attacks, Network attacks, Physical attacks, Password Attacks, Information Gathering Attacks, Remote to User attacks and User to Root attacks. Another important property of intrusions is that they have varying anomaly characteristics. We therefore classify them in 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 behaviour of the system. This type of intrusion fits closely the Machine Learning problem of anomaly detection and it will be the main focus of our work. There are also *contextual anomalies* when a data point may be considered anomalous according to the context it is

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

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

TABLE 2.1. 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 performs 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 trying 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)

associated as well as it's place in the feature space. The context may be additional features engineered in our feature space or an evaluation of circumstances associated with the data point from the IDS (or it's 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 2.1[4] we 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 we 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. We then measure the distance of new data points with our learnt 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**]

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

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 labelled as different classes in our classification scheme. The third class is classification methods. They include both supervised and unsupervised algorithms. One good example, that we will use later, are support vector machines (SVM). They can be trained either on pre-labelled data or at non-labelled 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 learnt from previous attacks so that they are able to identify them when they re-appear. These methods include ontology, signature based and logic based approaches. One example of an 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 are soft computing methods. The key point behind them is that if finding the exact solution is not feasible then we can look for approximate solutions. These methods include Artificial Neural Networks, Rough Sets, Fuzzy Sets, Ant Colony Algorithms, Artificial Immune Systems and Genetic Algorithms. We mention as an example the work of Amini et al.[**16**] where they use adaptive resonance theory (ART) and self organising 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 method is the work of Borji[**17**]. He uses four classifiers, decision tress, SVM's, ANN's and kNN, that 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 neighbour 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 we 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-times artificial. This results in the subsequently trained model becoming outdated quite fast because of the fast paced evolution of the IT field. Using partially labelled data for semi-supervised clustering techniques can be more efficient than classification methods provided we have a feature space with a good distance measure. If not statistical techniques are a better option. As we mentioned another point to consider is how easy it is to update our application. Statistical, clustering and classification methods are all hard to train. But it can be done off-line and their testing phase is fast. Last but not least there are times when the assumption that intrusion events 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 normal operation of the user.

As we 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 we will limit ourselves to a particular case.

2.3. Datasets

A critical part in creating an IDS application is the data you use to train it on. In the enterprise world you have access to the company's data. In the 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 at the DARPA 1999 IDS Dataset[5]. 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 the 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 artefacts of the simulation used to create the datasets

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

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 be come a standard. One reason for this is that it is dificult to create a dataset that maintain the quality standards needed to not be criticised with any of the shortcomings the DARPA datasets have. This is the case for the dataset we will be using as well. Another reason is that the diverse needs of an IDS system means 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**]. We will describe it in detail in the next chapter.

2.4. Project Planning

An important part of any project is time management. In order to decide how to manage our time we first had to see the situation we were at and the goals we wanted to achieve. While the MSc project supervisor has done previous work in the field of Intrusion Detection he and his research team were 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 laying the ground work for further researching in the field. This means that our work will not be focused on only one approach but we will try many approaches to enhance our wider understanding of the subject. That doesn't mean we 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 were we each presented the work we were doing to the other members.

To assist us in better management we created the Gantt Chart shown in figure 2.4. It is separated in eight parts. Initially we explore the dataset, set up our system and decide the architecture of our workflow. After that we worked on replicating results of previous papers[**23**], [**24**]. This work overlapped with the work on frequency based algorithms and 2-sequence based algorithms. Once we replicated the work of previous papers we worked on new ways to analyse the dataset. One way of doing so was to get out of the boundaries established by the

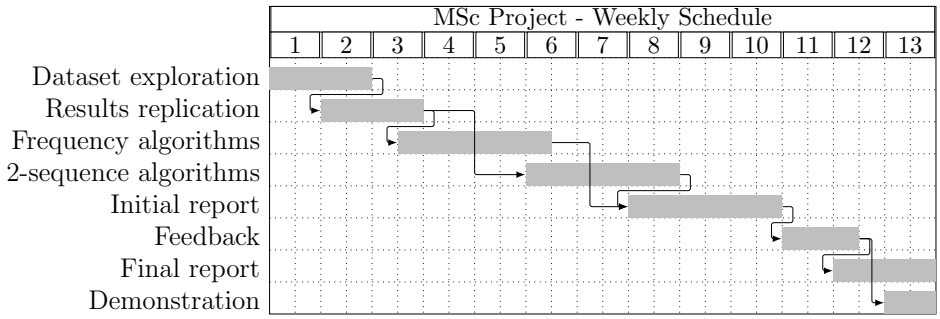

FIGURE 2.1. MSc Project Gantt Chart

computer science community in handling the dataset and using it in ways more conventional to the machine learning community. After that we proceeded in writing the initial report and deliver a draft to the first supervisor. Subsequently we used his feedback to improve our analysis and our report. Lastly we prepared a presentation in order to present our work to the second supervisor.

An overall view of the velocity of our work can be seen in figure 2.2. It includes both our 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 in work outside of those two such us computation time and literature review.

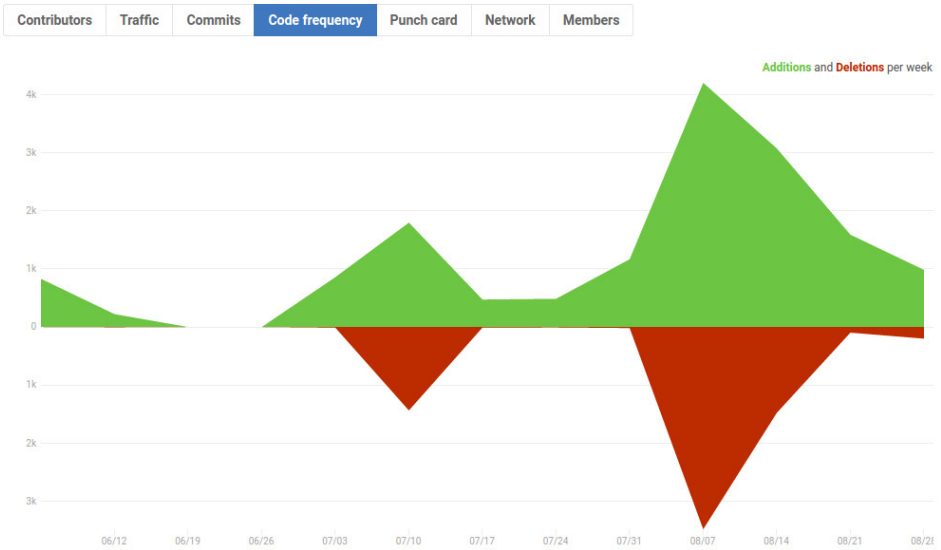

FIGURE 2.2. Code Frequency in the project's github repository.

CHAPTER 3

Exploratory Data Analysis

3.1. Dataset Documentation

The dataset we will use for this project is a modern dataset. It has been presented in 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, PHP after the names of the core software this approach uses. Consequently for the creation of the dataset Ubuntu 11.04 is used as the server operating system. 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) and were enabled in order to simulate 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[1] 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. Ubutnu 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 3.1 which was taken from [**1**]. The dataset is separated in three sets, the training set, the attack set and the validation set. The training and the validation set contain sequences of system calls from the normal operation

[1]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 meterpreter	Tiki Wiki Vulnerability exploit
Linux meterpreter payload	Client sidepoison executable
C100 Webshell	Php remote file inclusion vulnerability

TABLE 3.1. Attacks used in ADFA-LD 12 dataset.

Subset	Data points
Training	833
Validation	4372
Attack	719

TABLE 3.2. Subsets of ADFA-LD 12 dataset.

of the system and the attack set contains 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 3.2 shows the number of data points per subset of ADFA-LD 12.

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 we will use that schema we will not limit ourselves to it.

3.2. Preprocessing and Visualization

The first step in processing our dataset is to download it[2]. Subsequently we unzip the files we see that we 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.

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

The structure of the dataset after, it is unzipped, is not so helpful so we transform it to a format that will enable us to process it easier. Some of the standard tools for this purple are pickle and numpy[3]. They are python libraries that are used to save python objects. Our 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 we use pickle to save that object. Through this process we 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.

Our next step is to do basic exploration in our dataset. The kernel of the host operating system provides 325 system calls but we are not sure if all are represented and how that representation is distributed. As a first step we run through our dataset once and create a set[4] where we add all the system calls we encounter. Thus we end up with the following list of 175 system calls present in our 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 we only have 325 system calls only integers up to 325 would be used.[5] This is a very good example that no assumptions should be made when addressing a dataset and that we should always clean and validate the form of the data we expect to have

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

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

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

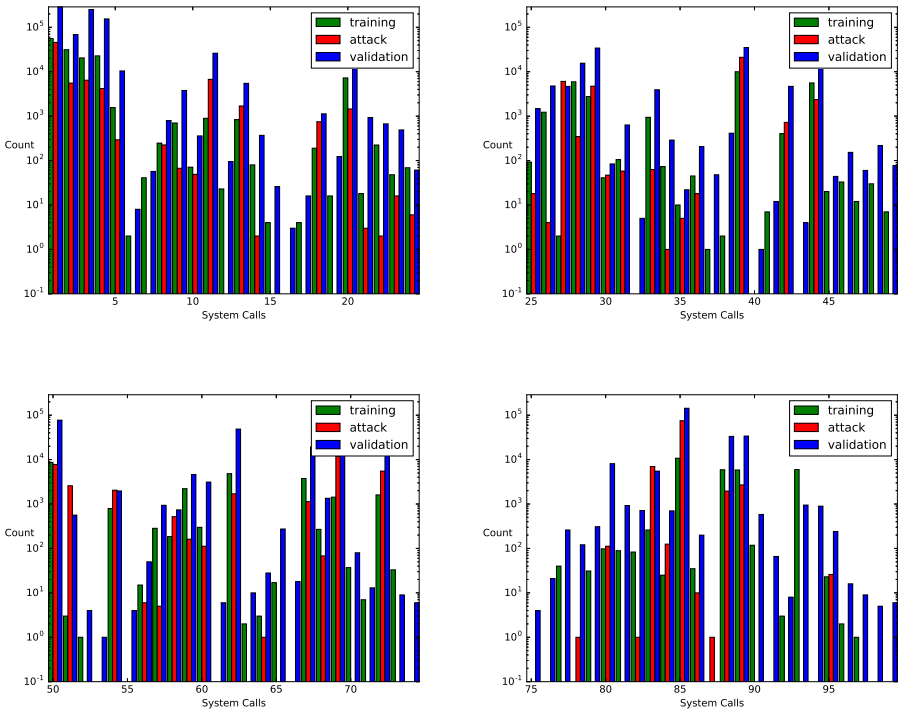

FIGURE 3.1. Counting how many times each system call occurs in the the training, attack and validation set.

for the next step of our pipeline. Another notable issue here is that we had not identified this problem initially when we were modelling the dataset. But numerical tests performed after constructing the system call frequency feature space, about which we talk more later, showed that we had not captured all the dataset. This resulted in more rigorous exploration that revealed the unexpected labelling of the system calls.

Moving forward we count the presences of each system call in our dataset. The result of this computation is presented on figures 3.1 and 3.2. As we see we had to use logarithmic scale for our y axis because of how unevenly and spread out the distribution of our system calls is. Moreover we 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 behaviour.

Another way to improve our basic understanding of our dataset is to look at it's principal components. In order to compute the principal components we merge the whole dataset. As we can see in figures 3.3

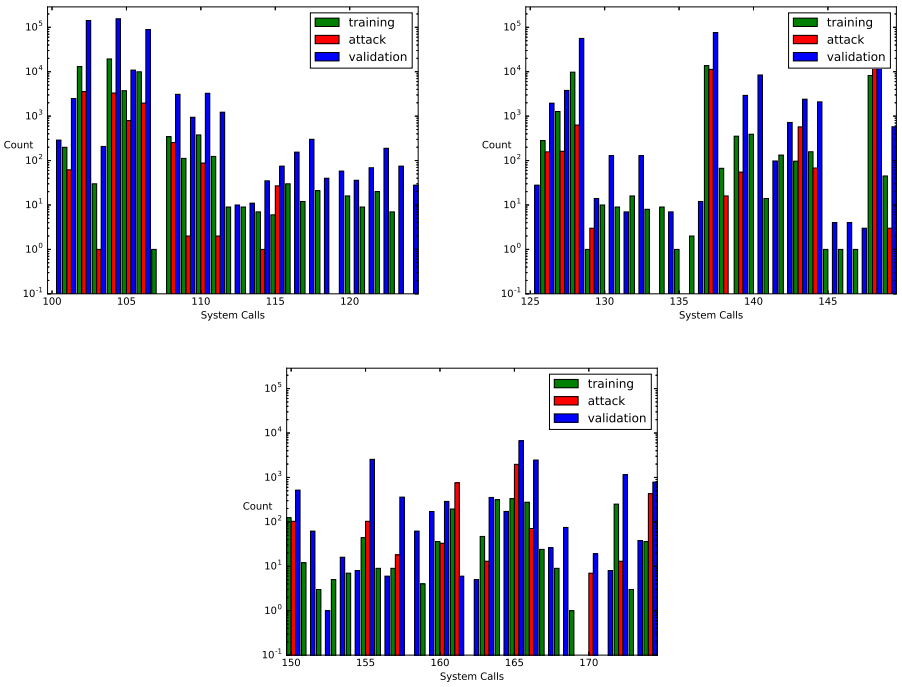

FIGURE 3.2. Counting how many times each system call occurs in the the training, attack and validation set.

and 3.4 we have used different colors for the three subsets of ADFA-LD 12. In the figure 3.3 we only plot the training and attacks sets. In figure 3.4 we also plot the validation set. By observing figure 3.3 we can see some degree of differentiation between the distributions of the attack and training set. However once we add the validation set, in figure 3.4, we 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.

We can see how much of the variance of the dataset is captured with each principal component of the complete frequency space in table 3.3. It is is worthwhile here to mention some specifics about the variance of our datasets. At table 3.3 we 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 we will reproduce in the next chapter they perform principal component analysis on the training set only.

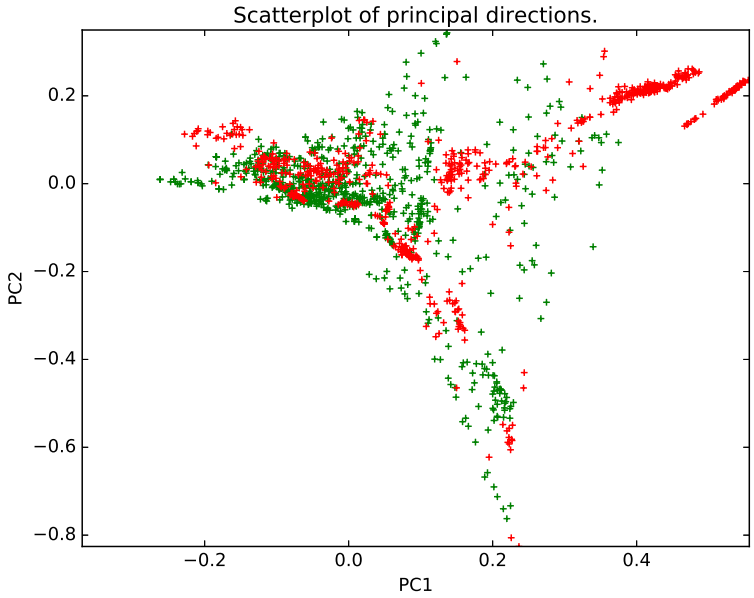

FIGURE 3.3. Principal Components of training and attack subsets.

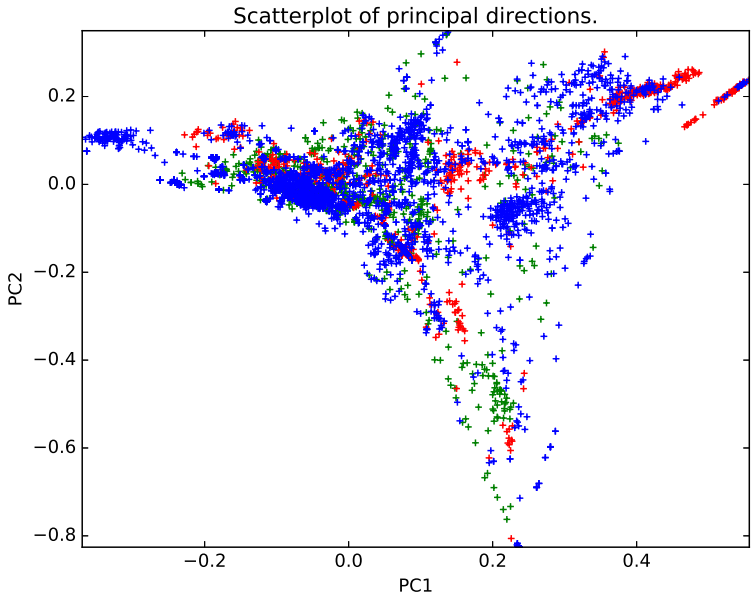

FIGURE 3.4. Principal Components of training, attack and validation subsets.

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 3.3. 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 3.4. Variance of the principal components of the training set.

This results in them needing only 9 dimensions in order to capture 80% of the variance of the training set. While we consider that choice sub-optimum, because we want our principal components to contain the diversity of the attack dataset in order for a classifier to take advantage of it, we will use their approach in order to be able to compare our results with them. The percentage of variance of the training set explained in each principal component in this case is presented in table 3.4. In both vases we can see that the variance explained by each subsequent principal component decays slowly.

CHAPTER 4

Frequency Analysis of the ADFA-LD 12 Dataset

We now proceed deeper in our analysis of the ADFA-LD 12 dataset. Our first goal is to replicate the results produced by Xie et al.[**23**]. The paper focuses on frequency based feature engineering, similar to the one we used to explore our dataset. Their analysis is based in two algorithms, k-nearest neighbours and k-means clustering. As we 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. We will follow their approach while we are trying to replicate their results in the following sections even though this means our reduced frequency feature space does not capture the dataset as efficiently as it could. This way ensures that we can verify that our replication has been successful. After that we can move in to our own approach.

4.1. k - Nearest Neighbours

We now describe the kNN implementation of Xie et al.[**23**], that we 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 where chosen empirically by the authors. For k they chose 20. Their choice of parameter d depends on the distance metric used. We present their choices[**23**] in table 4.1. The change in distance parameter d allows us to calibrate the sensitivity on identifying a data point as abnormal. The bigger

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

TABLE 4.1. 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.

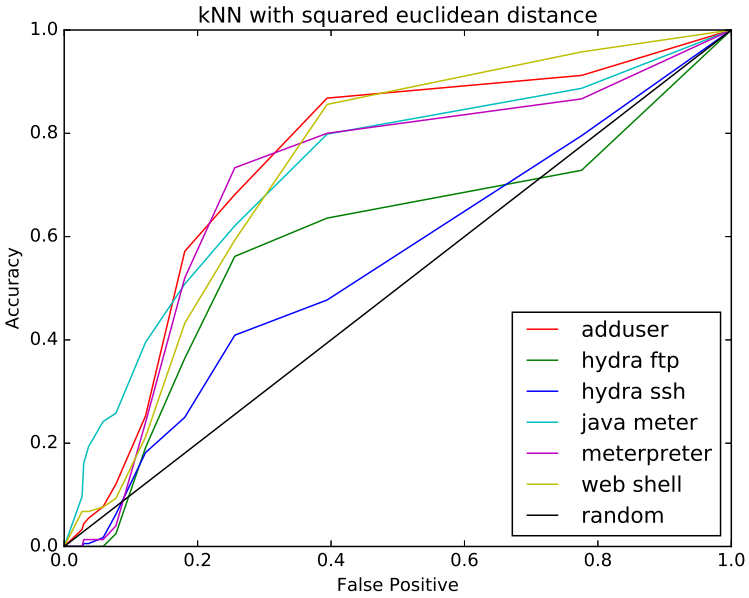

FIGURE 4.1. Receiver Operating Characteristic curve for k-Nearest Neighbours with square euclidean distance in the reduced frequency feature space.

the distance, the more likely we are to find k points and call that data point normal. This calibration procedure allows us to construct the ROC curves. After performing our computation we plot the receiver operating characteristic curves that have also been plotted by Xie et al.[**23**][1]. We can see them in figures 4.1 and 4.2.

To assess the performance of our classifiers we compute the area under the ROC curve. The results are shown in table 4.2. Calculating the area under the ROC curve for kNN under squared standardised euclidean distance we get the results shown in table 4.3.

We see that the password cracking attacks (hydra ssh and hydra ftp) are quite harder to identify compared to the other attacks and our classifier has poor performance on them. For the other attacks our classifier has moderate to good performance overall.

[1]We should note here that we 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. We add this trivial step in our implementation in order to have a measure that we can use to evaluate and compare results from different algorithms.

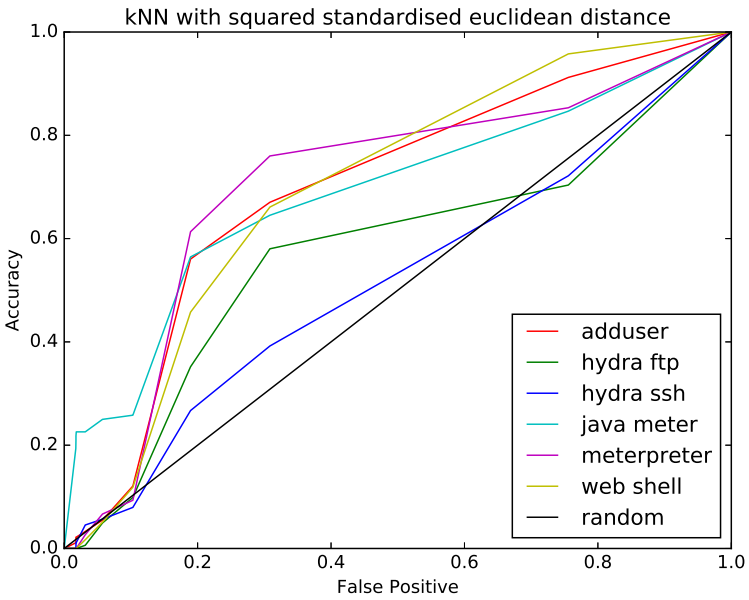

FIGURE 4.2. Receiver Operating Characteristic curve for k-Nearest Neighbours with square standardised euclidean distance in the reduced frequency feature space.

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 4.2. Area under the curve for squared euclidean distance kNN classifier in the reduced frequency space.

4.2. k - Means Clustering

We proceed in trying to replicate the results of Xie et al.[**23**] with regards to the k-means clustering algorithm. We 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 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

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 4.3. Area under the curve for squared standardised euclidean distance kNN classifier in the reduced frequency space.

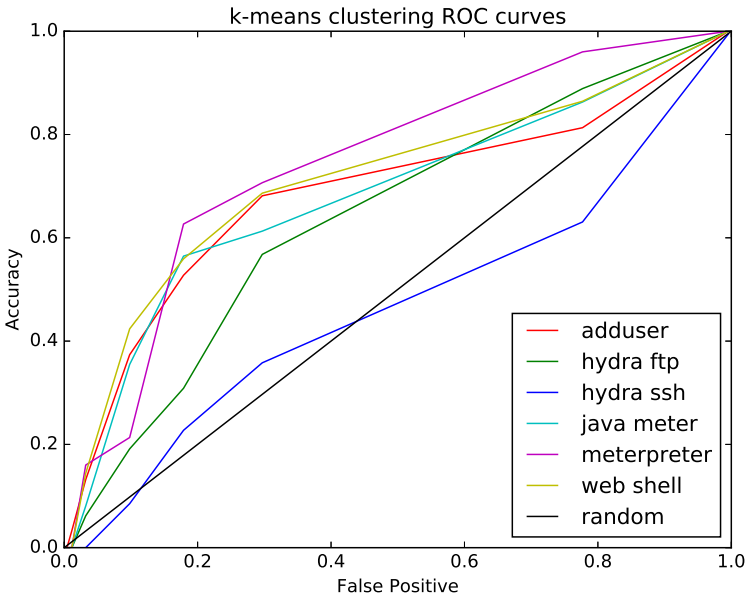

FIGURE 4.3. Receiver operating characteristic curve for k-means clustering on the reduced frequency feature space.

an evenly spaced sample of 10 numbers from $[0, d_{max}]$. In our case the maximum distance is 0.7551. This calibration of the distance parameter allows us to control the sensitivity with which we classify a data point as anomalous enabling the creation of receiver operating characteristic curves.

After performing our computation we plot the receiver operating characteristic curves. They are presented in figure 4.3. To assess

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 4.4. Area under the curve for k-Means Clustering classifier using euclidean distance on the reduced frequency feature space.

the performance of our classifier we compute the area under the ROC curves and present the results in table 4.4. We 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 Fall out (False Positive) rates. With our implementation we can compute it. By looking through figures 4.1, 4.2, 4.3 and the tables presenting the area under the ROC curve, 4.1, 4.2, 4.4, we 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 we were using for this work.

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

4.3. Support Vector Machines

4.3.1. Linear SVM on reduced frequency feature space.

A natural step in continuing our analysis of the ADFA-LD 12 dataset further than Xie et al.[**23**] are Support Vector Machines. As a first we will keep the methodology we used previously, meaning we will work with the first 9 principal components of the training set, so we can compare our results before moving forward. We will train an SVM model, using a linear Kernel, for each attack separately. More Specifically we will use the SVC class from the scikit-learn library[**28**]. Moreover we will use different values of the regularisation parameter C to better map the hypothesis space of available classifiers. The different

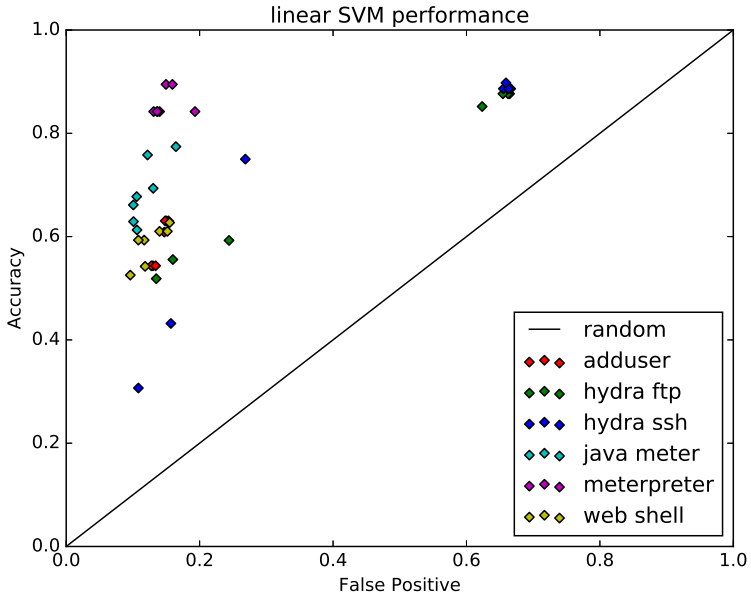

FIGURE 4.4. Performance instances for SVM's with linear kernel on ADFA-LD 12 dataset for different attack profiles in the reduced feature space.

values of C we used are:

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

We can see the first results of this computation in figure 4.4 where we plot various performance points on a True Positive versus False positive map. From there we conclude that for some attacks, namely webshell and meterpreter, the performance of the SVM classifier is not influenced a lot by the choice of regularisation parameter. On the other hand, hydra ftp and hydra ssh attacks are significantly influenced. For each attack we note the best performing value, as we can see in table 4.5.

We proceed with computing the optimum linear SVM classifier for each attack and then plot it's receiver operating characteristic curve. The results are shown in figure 4.5. The area under the curve for each classifier is shown in table 4.6. As usual the hydra attacks are our worst performers while the meterpreter attacks as our best performers.

At this stage we are able to compare our results, from table 4.6, with those of the k-means clustering algorithm, from table 4.4. Is is quite clear that Support Vector Machines are a more accurate and

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

TABLE 4.5. Optimal regularisation parameter for SVM's with linear kernel on ADFA-LD 12 for different attack profiles in the reduced feature space.

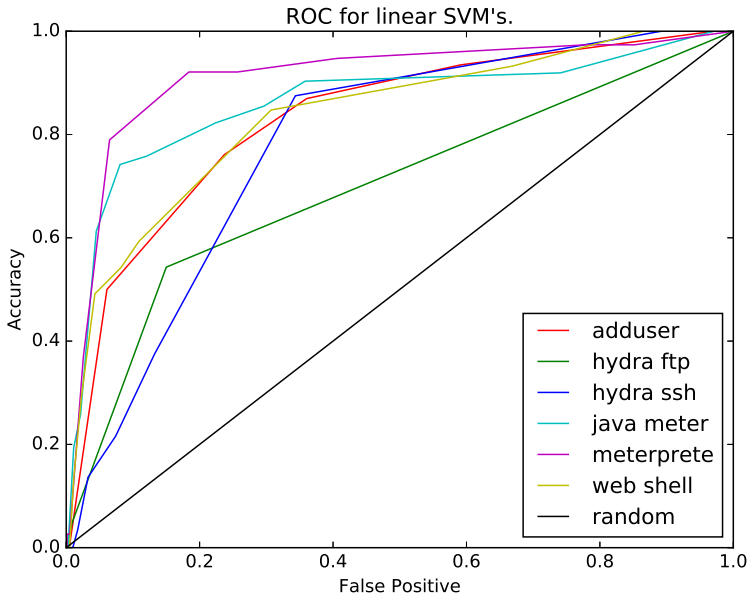

FIGURE 4.5. Receiver Operating Characteristic curves for SVM's with linear kernel on ADFA-LD 12 for different attack profiles on the reduced feature space.

reliable classifier for every attack. On average the SVM classifier has 0.15 higher area under the curve.

Moving forward we want to see the performance of Support Vector Machines on the dataset as a whole instead of training for each attack specifically. Hence we will treat the problem as a two class classification problem with our classes being normal behaviour and attack behaviour. To get a better idea of the performance of our classifier in this case we

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 4.6. Area under the curve for SVM's with linear kernel on ADFA-LD 12 for different attack profiles on the reduced feature space.

will use cross validation. To address the issue of class imbalance in our dataset we will use stratified 8-fold cross validation[2]. Going even further we run our simulation for various values of the regularisation parameter. We present our results in table 4.7. We can see that we get better performance[3] for low regularisation values. The optimum value is $C = 0.5$.

4.3.2. Linear SVM on the complete frequency feature space. Moving further away from the approach of Xie et al.[**23**] we investigate the application of support vector machines in the frequency feature space without reducing it's dimensionality through principal component analysis. We pool all of our dataset together and perform 8 fold stratified cross validation training a linear SVM classifier. We present our results at table 4.8.

As we can see by comparing tables 4.7 and 4.8 using the complete frequency feature space yields significantly better results. The precision rate is increased noticeably and the fall out rate is slightly decreased. The computation cost was not noticeably increased for the whole frequency feature space when running our scripts.

4.3.3. Recursive feature elimination.

Moving forward with our analysis of ADFA-LD 12 dataset we will conduct recursive feature elimination. We will train an SVM classifier with linear kernel for a two pattern classification problem. Our two classes will be attack and normal behaviour. A property of k-fold

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

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

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 4.7. 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 4.8. 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.

cross validation is that only a small part of the dataset is used as a validation set meaning our under-represented attack class may not be appropriately represented in the validation set. For this reason we will use a sampling method to perform cross validation. We will be sampling our dataset with the StratifiedShuffleSplit method of scikit-learn[**28**] 6 times and our training and validation sets will be equal in size. To create a scorer and assess the performance of our classifier we will use the Precision (True Positive rate) and Fall out (False Positive rate) ratios. But because recursive feature elimination works by optimising one measure we will use their difference, namely:

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

When Scorer=1 we have perfect classification and if Scorer is 0 we are classifying everything as attack behaviour. After writing our code and performing the necessary computations we plot our results in figure 4.6.

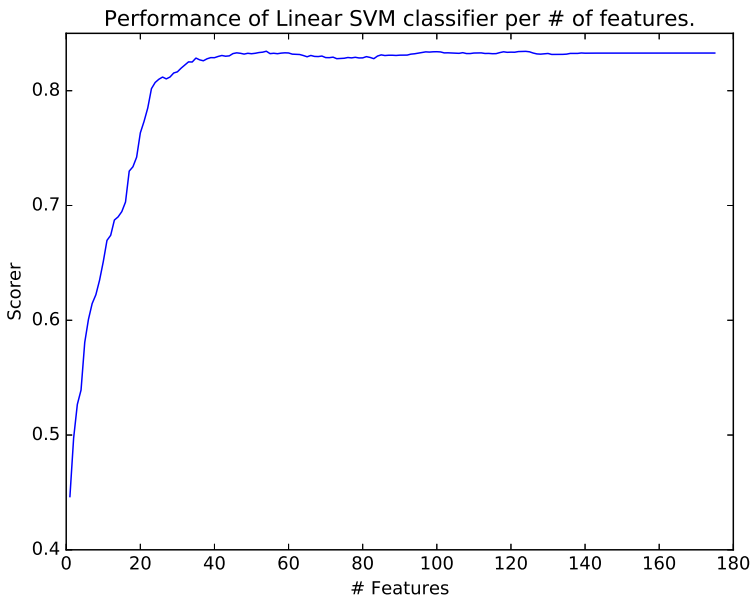

FIGURE 4.6. Scorer (= Precision - Fall out) performance metric compared to number of features in the complete system calls frequency feature space while conducting Recursive Feature Elimination. StratifiedShuffleSplit method was used for cross validation in order to assess scorer. Optimal number of features is 54.

By looking at the plot we see that when we start removing the less relevant features from the complete feature space our performance, measured by our scorer metric, remains steady. That plateau remains with minimal variation until we have 50 remaining features with a global maximum at 54 features. Then it has a small decline that keeps increasing and becomes a steep decline after we are left with only 25 features. Therefore we 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.

4.4. One class Support Vector Machines - Outlier Detection

Before we finish our analysis of ADFA-LD 12 on the frequency feature space we will study one class Support Vector Machines. The 1-class SVM algorithm has been introduced by Schlkopf et al.[**6**] and can work with a variety of kernel function just like normal SVM classifiers. From our preliminary results we saw that linear kernels were giving bad performance. After various trials we 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 we are asked to specify the upper bound for the fraction of training data that we can tolerate to be incorrectly classified. In order to study the upper bound's (ν) effects we will test it's performance by recursively performing cross validation on a range of values for ν. The results of our first test can be seen on figure 4.7 and table 4.9.

By studying the results of figure 4.7 and table 4.9 we see that there is big volatility for different numbers of ν. Moreover the standard deviation of our metrics is quite varying as well, something we didn't expect. This might be due to the relatively small number, 10, of re-sampling or because of some structure within our dataset that we 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 we repeat our procedure on a smaller range of values. The results of our computation are presented in figure 4.8 and table 4.10.

From table 4.10 we can identify $\nu = 0.41$ as the best performing value for the upper bound on training errors. Moreover from looking at both tables 4.9 and 4.10 we can see that the fall out rate follows the value of ν which makes sense given that the training data are all from normal behaviour and the fall out rate represents misidentification of normal data.

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 4.9. Exploring 1-class SVM with a sigmoid kernel to assess it's 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 behaviour 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 4.10. Exploring 1-class SVM with a sigmoid kernel to find it's optimum 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 behaviour datasets for training and validation. Optimal performance for upper bound $\nu = 0.41$.

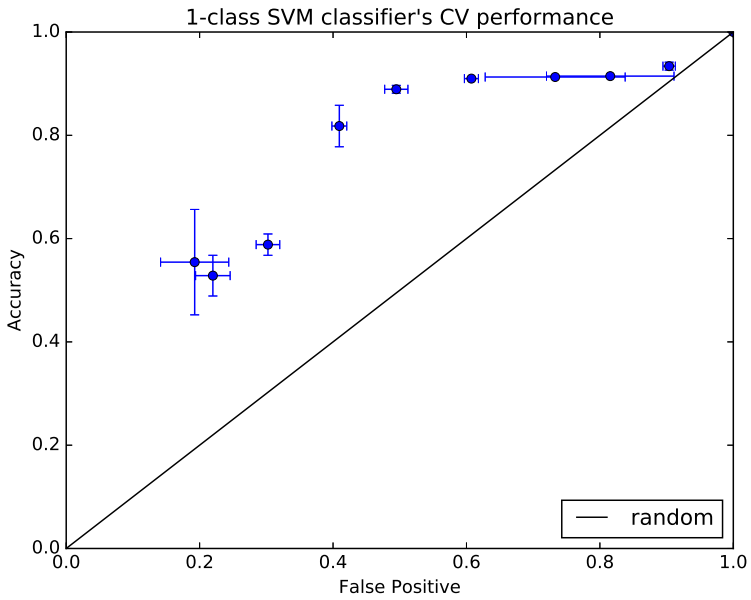

FIGURE 4.7. Exploring 1-class SVM with a sigmoid kernel to assess it's performance depending on the upper bound for the fraction of training errors. ShuffleSplit method was used for cross validation to create two, equal in size, normal behaviour datasets for training and validation.

Because the LIBSVM library[**30**] implementing the One-class SVM algorithm that we use does not predict probabilities for a data point to be classified as outlier but only classifies them as such we cannot create ROC curves. However by observing the performance of the algorithm depending on the bound for the training error at figures 4.7 and 4.8 we see that there is a resemblance to a receiver operating characteristic.

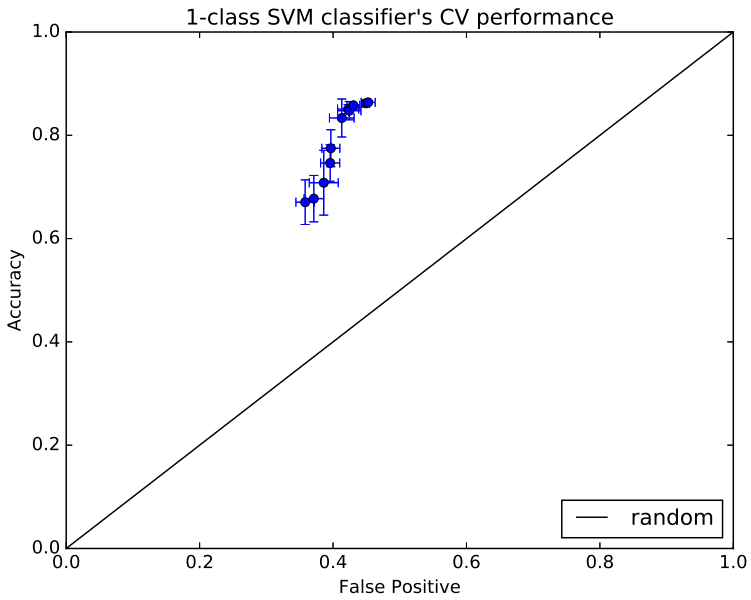

FIGURE 4.8. Exploring 1-class SVM with a sigmoid kernel to assess it's performance depending on the upper bound for the fraction of training errors. Dataset was feature engineered on complete frequency feature space. ShuffleSplit method was used for cross validation to create two, equal in size, normal behaviour datasets for training and validation. Using more fine grained parameter search to find optimum value. Optimal performance for upper bound $\nu = 0.41$

CHAPTER 5

2-Sequence Analysis of the ADFA-LD 12 Dataset

After our analysis of ADFA-LD 12 dataset on the frequency feature space we proceed to analyse it on the two sequence feature space.

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

5.1. Support Vector Machines

We start our analysis of the two-sequence feature space by addressing our data as a two pattern classification problem. We will use a linear kernel for our SVM classifier. We will test it's performance on various values of the regularisation parameter (C) performing stratified cross validation. After computing the necessary calculations we get the results shown in table 5.1.

We see that we get our best performing results are when we have $C = 0.25$. By comparing the results from tables 4.8 and 5.1 we 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 space captures more information from the dataset.

Moving forward we will conduct recursive feature elimination to see how much information is contained in each feature. We will use the same procedure as before. We will train our classifier to distinguish a two pattern classification problem. We will use Stratified Shuffle Split method to perform cross validation. In order to make the computation time reasonable we will use a reduction step of 24 features (out of total 3792) per iteration. After performing our computation we get the results depicted in figure 5.1. We see that performance plateaus at 240

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 5.1. 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.

features and stays relatively stable afterwards with an obtuse maximum at 2088 features. Below 240 features performance drops significantly, hence we can say that the core information content on normal and attack classes is contained in the 240 most important features.

5.2. One Class Support Vector Machines - Outlier Detection

The last step of our analysis of ADFA-LD 12 on the two-sequence feature space is studying one class Support Vector Machines. Again from our preliminary results we saw that linear kernels were giving bad performance. 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 we will use the same parameters that we used before, $\gamma = 0.05$ and $c_0 = 3$.

As we have already said one more tricky thing about the 1-class SVM classifier is that on the training set we are asked to specify the upper bound for the fraction of training data that we can tolerate to be incorrectly classified. In order to study the upper bound's (ν) effects on the two-sequence feature space we will test it's performance by recursively performing cross validation on a range of values for ν. The results of our first test can be seen on figure 5.2 and table 5.2.

While figure 5.2 gives us a good overview of the performance of one class support vector machine the iteration step, 0.1, for ν (maximum

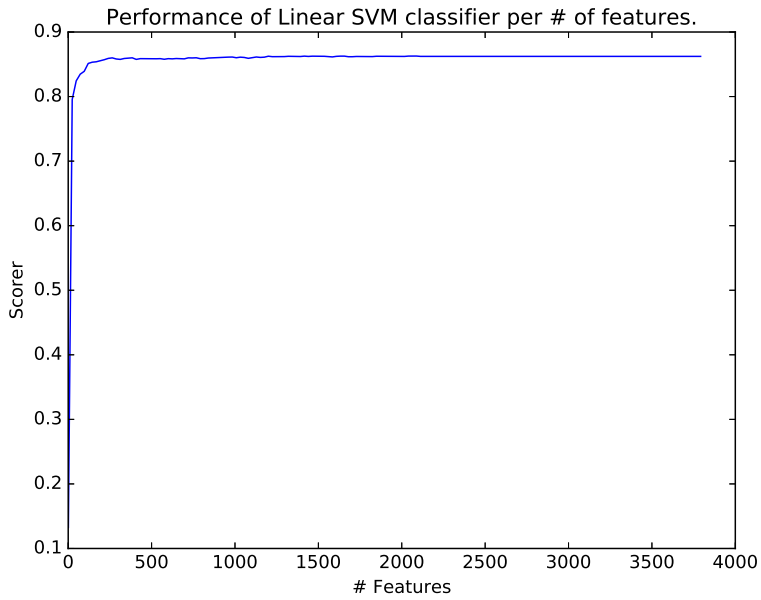

FIGURE 5.1. Scorer (= Precision - Fall out) performance metric compared to number of features in the complete system calls two-sequence feature space while conducting Recursive Feature Elimination. StratifiedShuffleSplit method was used for cross validation in order to assess scorer. Elimination step is 24. Optimal number of features is 2088.

error ratio tolerance for training set) is too big to tell us the optimum value. Hence we re-iterate with smaller iteration step, 0.01, for values between $\nu \in (0.35, 0, 45)$ to find the best performing value. We see our results for $\nu = 0.36$.

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 5.2. Exploring 1-class SVM with a sigmoid kernel to assess it's performance depending on the upper bound for the fraction of training errors. Dataset was feature engineered on two-sequence feature space. ShuffleSplit method was used for cross validation to create two, equal in size, normal behaviour 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 5.3. Exploring 1-class SVM with a sigmoid kernel to find it's optimum 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 behaviour datasets for training and validation. Optimal performance for upper bound $\nu = 0.36$.

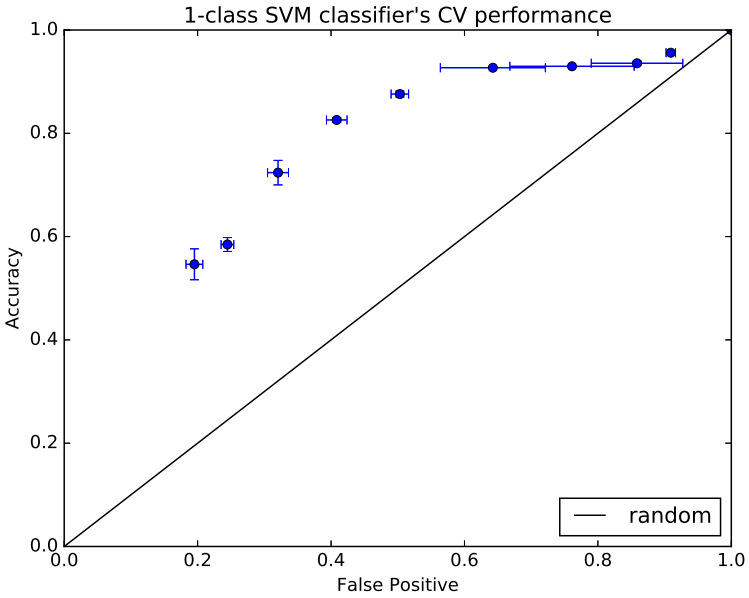

FIGURE 5.2. Exploring 1-class SVM with a sigmoid kernel to assess it's performance depending on the upper bound for the fraction of training errors. Dataset was feature engineered on two-sequence feature space. ShuffleSplit method was used for cross validation to create two, equal in size, normal behaviour datasets for training and validation.

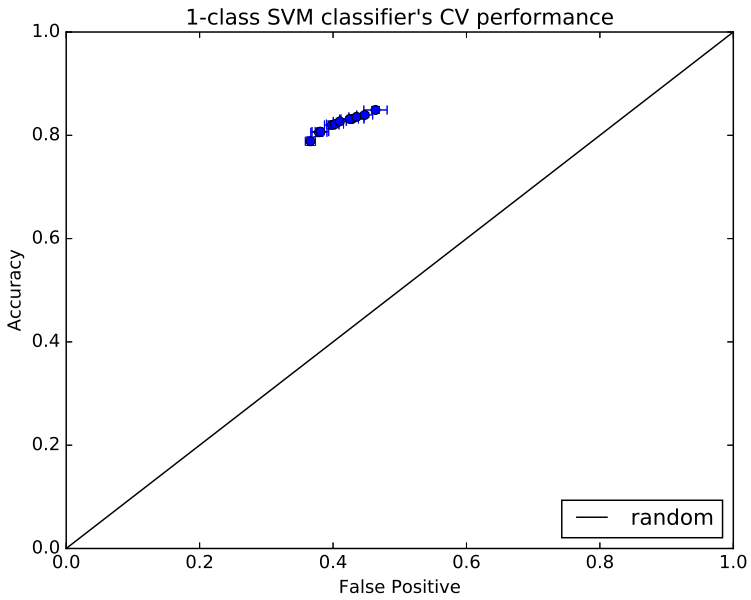

FIGURE 5.3. Exploring 1-class SVM with a sigmoid kernel to assess it's performance depending on the upper bound for the fraction of training errors. Dataset was feature engineered on complete two-sequence feature space. ShuffleSplit method was used for cross validation to create two, equal in size, normal behaviour datasets for training and validation. Using more fine grained parameter search to find optimum value. Optimal performance for upper bound $\nu = 0.36$

CHAPTER 6

Project Evaluation

After having gone through our results it is time to review them with a more critical approach.

6.1. Validation and Testing

Before we do that however is important to mention a key part of our work that has been lurking in the shadows. This is validation and testing. All of our numerical analysis has been done through computations. It is easy to make a mistake that introduces bug in the python code but it is very hard to identify that by looking at the results. Therefore we 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 our log files in order to verify them. This approach has not only helped us identify bugs in our code immediately but it has also shaped how our 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, we had assumed that they were represented but the integers from 1 to 325. Numerical testing, in this case mostly through summation, of the relevant probabilities revealed that something was missing. More rigorous exploration of the dataset revealed the problem.

6.2. Results Analysis

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

We then moved further by implementing Support Vector Machines classifiers. Comparing the area under the ROC curves for the various cases described in tables 4.2, 4.3, 4.4 and 4.6 is not very straightforward so we created figure 6.1 to condense that information. We can see that

k-means clustering and k-nearest neighbours with squared standardised euclidean distance are the worst performers while support vector machines perform a lot better.

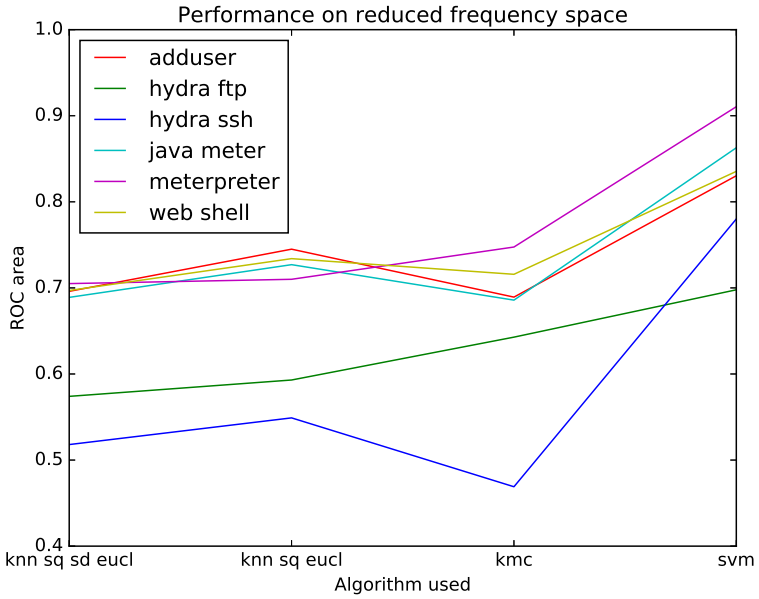

FIGURE 6.1. Performance, measuring the area under the ROC curve, for various attack detection algorithms on the reduced frequency feature space.

We then moved to the complete frequency feature space. When formulating the problem as a two pattern classification we saw, from tables 4.7 and 4.8, 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 it's dimensionality. To that extend we tried recursive feature elimination and we saw that we could eliminate up to 121 out of 175 dimensions of the feature space with no loss of performance. As we can see from figure 4.6 we could even go below that and unless we keep less than the 35 most informative features we are not sacrificing much on performance of our algorithm.

We then move to trying an unsupervised learning approach using one-class support vector machines where we saw a significant drop in performance as evident from our results on tables 4.9 and 4.10.

Our 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 our performance improved we create figure 6.2 which takes information from tables 4.8 and 5.1. As an evaluation metric

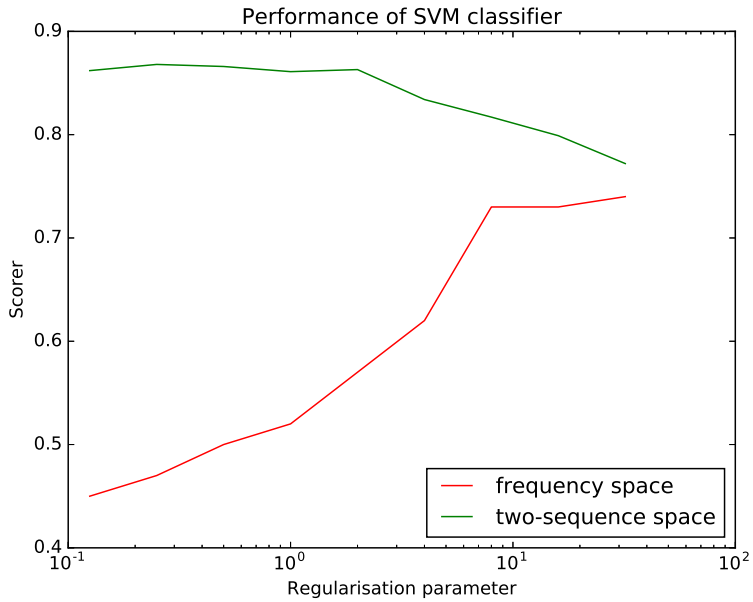

FIGURE 6.2. Performance, measuring the area under the ROC curve, for Support Vector Machine Classifier by varying the regularization parameter Showing results on the frequency and two-sequence feature spaces.

we use the scored we have defined earlier as the difference between precision and fall out rate.

Last but not least we compare the performance of one class support vector machine. This algorithm acts as outlier detection and from what we see from figures 6.2, 6.3 it's performance is significantly poorer compared to support vector machines classifier. We can also see that there is very little difference between the two feature spaces. This is in contrast with our previous results when intrusion detection was framed as a two pattern classification problem.

6.3. 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

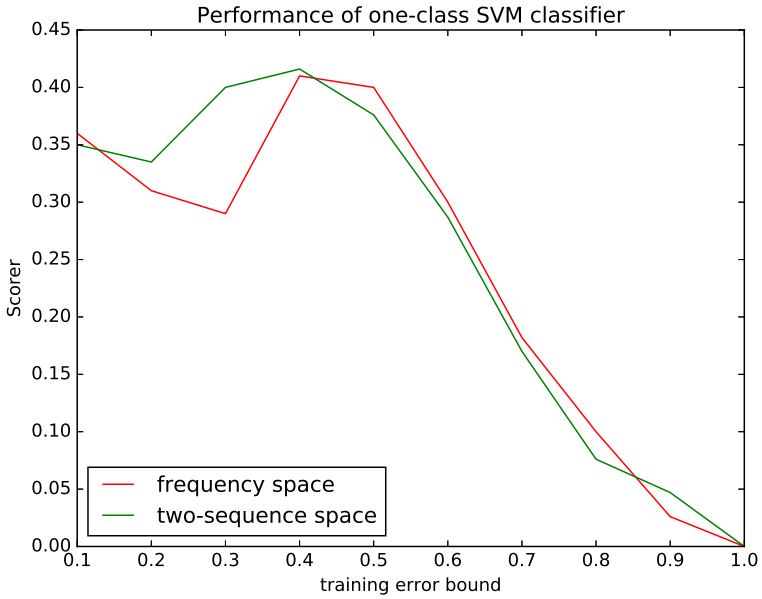

FIGURE 6.3. Performance, measuring the area under the ROC curve, for one class Support Vector Machine Classifier by varying the bound on the fraction of the training error. Showing results on the frequency and two-sequence feature spaces.

forward is to take note of the system calls identified as more relevant from recursive feature elimination.[1] Someone with expertise on 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 we are using. The AWID 2015 dataset by kolias et al.[**25**] is a very good candidate. It's 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.

[1]The 54 more informative system calls of the complete frequency feature space are: 11, 13, 15, 19, 27, 37, 39, 42, 45, 57, 78, 85, 91, 96, 97, 99, 104, 114, 117, 125, 140, 141, 155, 159, 160, 162, 168, 174, 175, 191, 195, 196, 199, 200, 202, 206, 207, 209, 212, 213, 215, 224, 243, 256, 258, 265, 266, 268, 301, 307, 309, 311, 331 and 332

Last but not least there is great room for improvement in using better kernels in unsupervised outlier detection. The fact that we saw very small performance increase by moving to the two-sequence feature space means that we 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 extend of each algorithm's efficiency. This enhances our overall understanding in the Machine Learning field and helps us find areas in need of novel approaches.

APPENDIX A

Methods and Technologies used

Before moving further we would like to mention the technologies we used for this project. Our main computing platform is a x86_64 machine running Ubuntu 16.04.1 LTS[1] as it's operating system. For our analysis we are using the python[**26**] programming language, version 2.7.12. Furthermore to implement our machine learning algorithms we relied on the numpy[**27**], version 1.11.0, and scikit-learn[**28**], version 0.17.1[2], open source libraries. For visualization we relied on the matplotlib[**29**] library version 1.5.1. The code used for the generation of all the results and graphics of this project is publicly available as a github repository[3]. In order to write our report and presentation we used pdfLaTeX and Texmaker. Last but not least we used git, version 2.7.4, as a version control tool.

In order to streamline our work flow we structured our working repository in the following way. We use / to denote the root of our repository. We put out all of our data files in the /data/ folder and configure .gitignore to ignore files in it. For our analysis we use the /scriprs/ folder where we place all our python scripts. The plots we create are put on the /pictures/ folder which is also ignored. Lastly we wrote the report and the presentation on the /report/ folder configuring .gitignore in a way as to only monitor the two .tex files only (and not the auxiliary files used by the pdfLaTeX compiler). This allows an efficient use of git with github allowing us to track important changes more efficiently. One technicality that is important for reproduction purposes is that our python scripts are written in a way that needs them being called from the /scripts/ directory. This is so that they can correctly access files in the /data/ directory during their runtime.

Since we are not building an application that is to be deployed in a production setting we use a more basic approach for our coding

[1]Ubuntu is an open-source operating system using the Linux kernel, based on Debian and developed by Canonical Inc., `http://releases.ubuntu.com/16.04/` - Accessed at August 10, 2016.

[2]An important note here is that pickled scikit-learn objects may be incompatible with other scikit-learn versions.

[3]The repository is available there: `https://github.com/Iolaum/idsandet`

requirements. We create a lot of individual scripts that perform one small task. This gradual progress allows us to monitor keep a very good understanding of our progress because every step we make is assessed before we move to a new one. Another benefit of our modular approach is our ability to test our code. Writing unit tests for our work is inappropriate. Therefore we test manually[4]. The fact that our scripts are small means that we have to test for very few possibilities and the fact that we run our scripts one after another means those possibilities are added instead of multiplied keeping the manual testing feasible and not counter productive. Moreover this segmentation makes it easier to alter our implementation of a particular algorithm when we want to experiment further with our dataset.

As an additional remark we want to mention that since python is able to run in a variety of operating systems our results can be replicated on a variety of operating systems. Depending on the user's system he has to install all the required libraries and associated dependencies. In order to replicate our work one has to download the data and put each subset in the /data/training /data/validation and /data/attack folders. The script a01_intro.py can be used to verify that this has been done correctly. The python scripts a02_train.py, a03_attack.py, a04_validation.py are used to create the first pickle files of the dataset. After that the scripts can be run in sequence. The data files created have in their name the alphanumeric characters of the script that created so it is easy to check which script should have been run before the one we are attempting to run.

[4]In most cases manual testing means that we run a script in a small subset of our dataset and make sure the result is correct. Then we proceed to finalize the script in order to process the whole dataset. Whenever possible we run numerical tests on the final results to verify them.

Bibliography

[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. Schlkopf, 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.7684.

[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. 38993905, 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. 859867, 2010

[16] M. Amini, R. Jalili, and H. R. Shahriari, *RT-UNNID: A practical solu- tion to real-time network-based intrusion detection using unsupervised neural networks*, Computers & Security, vol. 25, no. 6, pp. 459468, 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. 254260.

[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. 130144.

[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)