Kernel-Based Outlier Detection For IoT Networks

Adam Kelly

Advisors: Robert Vanderbei, Samory Kpotufe, Nick Feamster

April 2019

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Engineering

Department of Operations Research and Financial Engineering

Princeton University

Abstract

Due to the increasing number of Internet of Things (IoT) devices surfacing, the network security of IoT devices is becoming an increasingly complex problem. With a growth in the number of devices on the market, attacks aimed at and utilizing these devices are also rising commensurately. As a result, being able to secure IoT networks and devices by finding effective ways to monitor and protect them is necessary. This project provides a novel method from unsupervised machine learning literature to identify anomalies in IoT networks. Anomalies are expected to be harmful activity in networks and are most commonly attacks by a botnet, privacy leaks, or intrusions. In current literature, there exist a multitude of different methods aimed at accomplishing outlier detection to improve the security of IoT networks, each with different capabilities. In this thesis, Kernel K-means is proposed as a basis for a generalizable outlier detection method for network security applications. It is run on a sample of benign network data in order to capture regular activity. This is then compared to potentially anomalous data, containing a combination of attack data and normal data to be classified with respect to the benign clustering. Due to the limited assumptions necessary to use Kernel K-means and its ability to capture highly irregular geometry in models, it is well suited to this problem and this is demonstrated in both toy examples and real network data.

Acknowledgements

There have been a plethora of amazing people that have been essential to making this project possible.

Professor Samory Kpotufe, who was kind enough to take an inexperienced statistics researcher under his wing and help him develop the skills to work on a machine learning thesis, and for providing an exciting and remarkably relevant project to work on.

Professor Vanderbei, whose invaluable advice and guidance on research helped me develop as a researcher throughout the summer and then encouraged me to pursue my interests in this project.

Professor Nick Feamster and Noah Apthorpe were both essential in getting an ORFE major to understand the network security components of the project.

My family who has fostered my appreciation for mathematics and problem solving. The support they have given me not only through my thesis endeavors but also my entire time at Princeton has been immense.

To my friends who have patiently waited through the constantly developing explanations of this project, and have been an incredible group for my entire time through Princeton.

Mattie, who both provided endless encouragement and had the patience to edit and decipher my grammar into English.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the scope of Internet of Things (IoT) networks expanding rapidly, the difficulty of securing is swiftly multiplying. Given the existence and open source code available for effective botnets such as Mirai and Gafgyt, which are capable of launching Distributed Denial of Service attacks in large numbers, the ability to detect infected devices that can contribute to these attacks becomes essential. However, there are a variety of other purposes for detecting outliers in a network ranging from unusual behavior, such as data leakages, to other types of infections in devices. The size and scope of IoT networks as well as the information they present make them excellent candidates for using clustering algorithms in order to determine their ability to capture structures in the data. There are also numerous steps to taking in the data given by IoT netorks and getting it into a usable form. These methods will have a large effect on the ability to interpret and obtain results, and vary in the types of attacks that they can be expected to capture. This project will show the viability of using unsupervised clustering techniques in order to identify outliers in IoT network data, and give certain assumptions on the data following the type of processing used.

1.1 Motivation

There are a variety of purposes for detecting outliers in a network ranging from unusual behavior such as data leakages, identifying infected devices scanning for additional vulnerabilities, and different types of attacks. The size, scope, and vulnerability of modern day IoT networks make them excellent candidates for research in improving their security. It is predicted by Gartner that the total number of IoT devices will be 20.4 billion by 2020 [16]. Also, according to Corero Network Security, in Q3 2017, monthly DDoS attack attempts increased 35 percent over Q2, and 91 percent over Q1 [7]. With demonstrated attacks utilizing botnets with hundreds of thousands infected devices the ability to prevent these attacks lies in large-scale detection methods that can apply to different types of networks and are scalable in both memory and run time [8]. There is an incredible variety of devices in networks, and the number of manufacturers and types of different devices being brought into the IoT is still increasing. As a result, the methods also need to be flexible to very different types of benign activity in and depend upon limited assumptions within the data. Using flexible, scalable, and tested methods will allow increased network security for a massive selection of applications.

The purpose of this project is to provide a failsafe for network security in the IoT that is able to be easily applied to a wide variety of possible network schemes. While a number of other measures exist to stop attacks earlier, such as to prevent attacks and detect botnet scanning and identifying vulnerable devices, they are not uniform across devices and often can require device resources. While these methods can be effective, if it requires software be downloaded on every device, or requires device resources, it is not nearly as simple to apply to either growing networks or to a vast number of products included in the network.

In addition, it is a goal to not have to maintain a significant amount of data in this process. If this is the case, it requires a large amount of network resources, and incurs significant cost and timing issues when dealing with a cumbersome dataset. Throughout the project, scalability and moving to large and unknown networks is always an important consideration. In order for the final algorithm to be used in larger networks, it is important to keep the algorithm such that it can adapt to different networks without the need for too much tuning throughout the process of training it for a new network.

1.2 Applications

The applications of this project will be in large IoT networks. The results presented follows the process of finding infected devices or unusual activity in specific devices across large networks, following the intrusion, and being able to identify different attacks or problems with a device based on its network flows. This specific route of combating botnets and other intrusions of privacy within a network is more feasible than other suggested methods due to its scalability and ability to learn different types of networks. In the case of installing software on each individual device to protect against such attacks, in order to make improvements in the techniques used in the device, new software needs to be installed on each and every device. This can be impossible for a single entity trying to improve their network security when they have devices from multiple vendors. Therefore, the most efficient approach is to study techniques that can rapidly identify issues on all the devices of the network.

1.3 Overview

Following the literature review, the K-Means and Kernel K-Means algorithms will be described. The specific kernel used in this case, the Radial Basis Function Kernel and its explicit mapping are covered. The pseudocode for this specific implementation is shown, and the kmeans++ initialization used for more accurate results and better runtime is covered. Following this, since the Kernel K-Means is the method for learning the normal model of the data, the outlier detection process is described for this model. Due to the specificity of the application, details of how to deal with different levels of assumptions in the data are covered. This allows for further generality in being able to detect outliers with a wide range of different types of data. Finally, testing results are provided for two different types of data. First, a toy dataset is used in order to demonstrate the specific ability that Kernel K-Means has compared to the Gaussian Mixture assumption. Then, these two are tested on both data obtained from the UC Irvine data repository, originally created by Meidan et al [15]. Finally a conclusion on the effectiveness of the results as a whole and the ability to apply this method in real world applications for real time outlier detection is provided. The future research in this field is very broad and rife with possibilities, so a few specific options that have strong potential based on the current literature and project results are given as well.

1.4 Literature review

1.4.1 Outlier Detection

For outlier detection there are two different general techniques used. One is clustering assuming some outlier cluster or accounting for outliers in the process of clustering. This technique suffers from needing to know the number of expected outliers in the data for accurate results, otherwise will have its clusters drastically skewed by the outliers [6]. Knowing the number of outliers is often not feasible and testing for the number of outliers is usually even more difficult than the original problem of detecting the outliers. While there can be cases where clustering with outliers can still produce good results, this often is extremely dependent upon the geometry of the problem's data and location of the outliers relative to the benign clusters. Therefore, it may not scale to the plethora of different networks types that are presented in practical applications. The second approach, which is utilized in this paper, involves outlier detection starting with a set of “benign” data, which is assumed to be free of any outliers. While it is possible to perform this with some outliers in the data, the results will be affected adversely because then the original model faces similar problems as clustering with outliers present in the data. Except now these concerns of skewed clusters are additionally compounded into the anomaly detection. Then, once new points are identified and put into the outlier detection algorithm, the outlier detection process identifies each point as anomalous or non-anomalous, and from this specific devices' activity is determined to be either anomalous or benign.

There have been many approaches taken so far with this second form of outlier detection. The most successful are the autoencoders run on the dataset from UC Irvine in [1] and [15] by Meidan et al. who provides results of a perfect true positive rate and a minimal false positive rate on identifying attacks. The approach used in this paper is completely unsupervised and works effectively for all the botnets it is run on, with results in timing and accuracy that outperform a number of other methods' tests. The results are then expanded into an ensemble of autoencoders designed for large networks. However, the authors note that the method struggles on specific devices due to the differing number fo activities taken by each of these devices.

1.4.2 IoT patterns

The difficulty of identifying network patterns in IoT is demonstrated by Sivanathan et al. [22]. The results from this paper show there is a significant amount of deviation between devices in even different brands of IoT devices from the same type of device. Therefore, it shows the difficulty in treating IoT devices as a class or even with sub-classes significantly larger than a few devices. However, Doshi et al. observes basic behavior within the family of IoT devices, noting that it differed from regular internet traffic in the small number of other ports each device communicates with and often can have more patterns due to the need for these devices to log activity [12]. As a result, with more patterns in their network data, this is promising for research in this field. There may be difficulty in finding similarities across IoT device patterns, but there are still notable patterns within the activity of IoT devices as a whole.

1.4.3 Kernel K-means and General Outlier Detection

Kernel K-Means and other Kernel methods have been long researched for clustering and other classification methods. Kernel PCA, SVM, and many variations of each of these modified for different applications have extensive literature and have been used in a variety of applications and have various existing implementations [19] [5] [11]. Specifically, the research by Das et al. shows a specific version of Sparse Kernel PCA for solving outlier detection problems [9]. This approach formulates sparse kernel PCA as elastic net regularization in order to solve for the top eigenvalues and guarantees sparse solutions for each eigenvalue. Then, by using the reconstruction error of the Kernel PCA, outlier detection is performed. Kernel K-Means is not as often used for outlier detection because of the sensitivity it has to outliers, since outliers can dramatically shift averages, and the number of outliers is rarely known. There exist methods which involve calculating a number of outliers as the k- means algorithm is running. However, these often require knowledge of the number of outliers before the algorithm is run [6]. In many cases, it is possible to obtain some representative data of a network before any infection. Especially due to the difficulty these algorithms have in dealing with a wider variety of data, it is not amenable to use these algorithms in this setting.

1.4.4 Outlier Detection for Network Anomalies

There are a number of different approaches which have been specific to networks in anomaly detection literature. Most notably, the results obtained by Meidan et al. and Amar et al. using autoencoders and an ensemble of autoencoders follow a similar structure to the approach used for detection, and the same dataset is used for comparison [1] [15]. The approach used autoencoders in this realm with the same purpose of being able to detect outliers based on a learned normal model of the data. This method was shown to be effective over other techniques such as Isolation Forests, One-Class-SVMs, as well as Local Outlier Factor because it had a perfect true positive rate (TPR) while maintaining the lowest false positive rate (FPR) of all tested methods. While this is the most general result, and applies to a wide variety of network data there are multiple approaches to making IoT devices safer. One particular specfific approach, which is particularly difficult because of the difficulty of capturing HTTP Get attacks with traditional traffic volume analysis, is by Mirvaziri [17]. His approach to this problem was applying a random wait time on suspicious devices, using a trap link, and moving the link of the requested page. These methods are a part of a large existing effort towards a very different approach than taken by most machine learning practitioners to network security and this project. While extremely effective at protecting against this specific attack if implemented properly, the ability to scale as a resource to large IoT networks is limited. it also only guarantees protection against one attack, and while it is useful if most methods fail at detecting this specific type of attack it is limited to this specific attack. Other methods, which take a device-by-device security approach, also struggle to scale as it requires software and resources from devices which cannot always handle the extra load and also need to have this specific software installed on every device in a network.

The number of approaches taken to prevent attacks in network security is vast, each with a variety of different intended applications. However, the number of approaches which are designed specifically for IoT is limited, and for approaches that can scale, the number continues to decrease. The article by Butun et al. [4] shows the limitation of anomaly detection within the IoT network applications and looks further at which implementations can be done in the cloud. Of the implementations reviewed, including the novel implementation proposed in his work, only three of the 18 anomaly detection methods analyzed can work in both of these settings.

Chapter 2

Clustering Algorithm Description

2.1 K-Means

K-means solves the minimization problem of finding a partition in a set of points. The algorithm must be given the number of such clusters expected, which is m. Each cluster C_i is defined by its cluster center $\{c_j\}_{j=1}^{m}$ with collections of points closest to each c_j are located in C_j. The algorithm proceeds iteratively to minimize the cost denoted as Φ over the data set $\{x_h\}_{h=1}^{n}$,

$$\Phi = \min_{c_j} \sum_{j=1}^{m} \sum_{x \in C_j} ||x - c_j||_2^2$$

[20]

The specific algorithm being used to solve the Kernel K-Means problem posed is Lloyd's algorithm [20]. This algorithm guarantees convergence to some value for any initialization c_j^0. However, this value is not guaranteed to converge to the global minimum [2]. The algorithm progresses at each iteration by reassigning each point x at iteration i to the closest center c_j^i to the center's corresponding cluster for the next iteration C_j^{i+1} [20]. This can be described as:

$$x \in C_j^{i+1} \quad \text{if} \quad j = \arg\min_k ||x - c_k||_2^2$$

If Φ_i is the error at iteration i, then we have that when $|\Phi_{i+1} - \Phi_i| < 10^{-8}$ the algorithm stops. Following the completion of the algorithm, the explicit centers can be computed from the set of assignments C_j as

$$c_j = \frac{1}{|C_j|} \sum_{x \in C_j}$$

These centers can then be used as the resulting model of the clustering.

2.1.1 Kernel K-Means

Kernel K-Means operates in a similar manner as K-Means, with one important distinction; the algorithm is now working in a much higher dimensional mapping of the original data. While this high dimension is often expensive to work in, Kernel K-Means employs the "Kernel Trick" in order to make the process more efficient, and therefore only a unique Kernel function is used as opposed to calculating the high dimensional expansion of the data. [20]

First, given a dataset $\mathbb{X}^{n*p}$, a mapping ϕ which maps the vectors $x \in \mathbb{R}^n$, and an inner product $< \centerdot, \centerdot >$, a kernel matrix K_{nxn} based on the kernel function $\forall x, y \in \mathbb{R}^p, K(x,y) =< \phi(x), \phi(y) >$ is calculated which contains all possible inner products for the rows of the dataset. It is defined as

$$\forall i, j \in \{1, .., n\} \quad K_{ij} = K(x_i, x_j) =< \phi(x_i), \phi(x_j) >$$

[20]

Every calculation in the algorithm relies only on knowledge of this kernel matrix, so the implicit mapping ϕ is never calculated. Within the Kernel K-means algorithm this means that centers are never calculated. New centers c_j are never explicitly calculated and only exist in the algorithm only as assignments of points x to a cluster C_j. Each cluster center can be described mathematically as

$$c_j = \frac{1}{|C_j|} \sum_{x \in C_j} \phi(x)$$

. However, because the algorithm is often not able to actually compute ϕ or it is just prohibitively expensive, these cluster centers are never calculated. It is possible to calculate Euclidean distance from a center c_j. This will be denoted as $\mu_j(\centerdot)$ for center c_j and assignment set C_j In order to calculate distances from a cluster center, the distance is taken as such, where $|\centerdot|$ is the number of elements in a cluster [20].

$$\mu_j(x) = ||\phi(x) - c_j||_2^2 =< \phi(x), \phi(x) > -2 < \phi(x), c_j > + < c_j, c_j >$$

$$= K(x,x) - 2 \sum_{x_k \in C_j} K(x, x_k)/|C_j| + \sum_{x_k \in C_j} \sum_{x_h \in C_j} K(x_k, x_j)/|C_j|^2$$

If x_ℓ is in the original dataset, then it can be described using its index in the Kernel matrix K and $\mu_j(x_\ell)$ is now:

$$\mu_j(x_\ell) = K_{\ell\ell} - 2 \sum_{x_k \in C_j} K_{\ell k}/|C_j| + \sum_{x_k \in C_j} \sum_{x_h \in C_j} K_{jk}/|C_j|^2$$

Then $x_\ell \in C_j^{i+1}$ if

$$j = \arg\min_k \mu_k(x_\ell)$$

The implementation stops at the iteration where the total error function being minimized decreases within a specified tolerance of $|\Phi_i - \Phi_{i+1}| < 10^{-8}$.

2.1.2 Pseudocode

Psuedocode for Kernel K-Means:

```
X is data matrix size n*m, with m Xi vectors
initialize assignments randomly or kmeans++
k clusters , cluster j defined by group Cj with center cj
generate Kernel matrix K
While difference in distance is within criterion
    for each Xi
        Find Minimum distance from center cj for each Xi
        Assign Xi to its closest cluster
```

$$K(i,c_j) = \sum_{X_k \in C_j} K(i,k)$$

$$K(c_j,c_j) = \sum_{X_i \in C_j} \sum_{X_k \in C_j} K(i,k)$$

$$K(i,j) = e^{-\gamma ||x_i - x_j||_2^2}$$

2.2 Algorithm Complexity and Implementation

2.2.1 RBF Kernel

The Radial Basis Function Kernel is defined for any vectors $x, y \in \mathbb{R}^n$ as $K(x,y) = e^{-\gamma||x-y||_2^2}$. In some representations $\gamma = \frac{1}{2\sigma^2}$. This kernel satisfies a number of important properties, the most significant is that it is always positive and symmetric. As a result, the function $K(.,.)$ is a positive definite kernel function. Then, by Mercer's theorem, the mapping $\phi()$ which satisfies $K(x, y) =< \phi(x), \phi(y) >$ is an infinite dimensional mapping [10]. For the RBF kernel this inner product of explicit mappings where $\gamma = 1/2$ is

$$K(x, y) =< \phi(x), \phi(y) >$$

$$= \sum_{j=1}^{\infty} \frac{(x^T y)^j}{j!} e^{-||x||_2^2} e^{-||y||_2^2} = \sum_{j=0}^{\infty} \sum_{\sum n_i = j} e^{-||x||_2^2} \frac{x_1^{n_1} ... x_k^{n_k}}{\sqrt{n_1! ... n_k!}} e^{-||y||_2^2} \frac{y_1^{n_1} ... y_k^{n_k}}{\sqrt{n_1! ... n_k!}}$$

Where the jth element of the mapping ϕ is described as

$$\phi(x)_j = \left(\frac{e^{-||x||^2}}{\sqrt{j!}} \binom{j}{n_1, \ldots, n_k}^{1/2} x_1^{n_1}, \ldots, x_k^{n_k} \right)_{\sum_{i=1}^{k} n_i = j}, \quad j \in \mathbb{N}$$

[21] This is then an infinite feature mapping, however because the inner product of $\langle \phi(x), \phi(y) \rangle = K(x, y) \quad \forall x, y \in \mathbb{R}$, all that is needed is to know $K(\cdot, \cdot)$. Using this kernel, as a result of its infinite dimensional implicit mapping, is what allows for Kernel-K-means' boundaries to reflect a variety of different types of geometries.

2.2.2 Parameter Selection

The effect that Kernel K-Means has in clustering varies drastically on the value of γ. The variation of γ is what allows for the method to capture a variety of different types of behavior and take on a different selection of boundaries. This parameter needs be tuned to each instance to maximize results, and there are a number of ways to find the best parameter for this instance. The most basic option is to simply analyze visually which clusters best capture the data and base the parameter on this. However, this only works for small data sizes in dimension of one or two that can be visually verified. Therefore, that method is limited. Then next approach is to identify the accuracy of the clusters based on the detection of each device's activity. However, because the method relies on detecting activity, often the data's existing labels of network activity, resulting from different devices are not effective labels for testing the ability of the network to cluster. Often due to the different types of activity from each device, the actual clusters of data are different than just the device labels. This occurs when a device either has multiple patterns of activity, or overlapping activity with another device. If there exists samples of known malicious data then it is possible to train the parameter and number of clusters based on finding the minimum False Positive Rate while maintaining a True Positive Rate of one. This is the approach taken by the method with autoencoders [15]. The results of this project show that this is a more feasible approach than basing the parameter selection off of the one that produces the best AMI with the device model as labels.

Otherwise, assuming neither of these assumptions are true, then there are still methods for identifying the best potential parameter and number of clusters. Many of these techniques involve comparing an assortment of point, center, and total distances for different parameter values. However, while tuning for the γ parameter, the distance metric itself is changing. The final distance for each iteration is in the general form $e^{-\gamma * a} + e^{-\gamma * b} + ...$, making the different values of γ in this function difficult to compare. Therefore, the clusters between each set must be compared in the space that the data exists. The method used in this project compares the closeness of the inter-cluster distances and intra-cluster distances. There are a multitude of notions of representative inter-cluster and intra-cluster distance which can be used, however the notion selected will need to be able to identify success in clustering a variety of geometries. For inter-cluster distances, the estimation is done by taking the minimum inter-cluster distances for each cluster in the space the data is in, and averaging these values. This metric is described for cluster C_j as,

$$f_{inter}(C_j) = \min_{x \notin C_j, x' \in C_j} ||x - x'||_2^2$$

Then $\overline{f_{inter}}$ is the arithmetic mean of this metric for all clusters.

Next, the intra-cluster metric utilized is the maximum over all points in a cluster of the minimum distance between that point and any point in a cluster. For any cluster the minimum distance from one point in the cluster to any other point is a measure of how compact the cluster is around that point. In the case of taking the maximum of all such points, it is possible to find the areas in which the cluster is not nearly as compact. Therefore, identifying the overall quality of the cluster could become an issue. When this maximum is low, it indicates each point is tightly focused around other points in the cluster. The metric can be described as:

$$f_{intra}(C_j) = \max_{x \in C_j} \min_{x' \neq \in C_j} ||x - x'||_2^2$$

From this, the arithmetic mean over all clusters is described as $\overline{f_{intra}}$. The parameter which creates partition P that minimizes the difference $f(P)$ the one that is utilized:

$$f(P) = \overline{f_{inter}} - \overline{f_{intra}}$$

2.2.3 Initialization

The simplest method of initialization for this algorithm is just by random initialization of the assignments. However, this often yields centers which are relatively near each other. Since the distribution is uniform, all the cluster centers will be expected to be at the same point; at the center of all such points in the dataset. Then, initializing uniformly random centers is a better method, but there are still issues with this if one cluster has the majority of the data. If this occurs, the likelihood of having two centers in this data is likely, and will inhibit the clustering. K-Means is known to be dependent upon the initialization, making the initialization important.

The benefit of using K-means++ has been shown through improving the results of K-means algorithms in achieving an expected solution close to the global minimum and preventing getting stuck at solutions farther from this minimum. It also demonstrated improved run time performance from the K-means algorithm with this initial seeding. [2] While this technique is traditionally used for K-means and not Kernel K-means, using it in this context still demonstrated improved results in both efficiency as well as cluster quality.

K-Means++ is implemented by taking the first cluster center c_1 from the set of points uniformly at random among all points. Following this, then a function $d(x)$ is calculated for each remaining point x, where at iteration n

$$d(x) = \min_{i \in 1 \ldots n-1} ||\phi(x) - \phi(c_i)||_2^2$$

Then the probability that point x is selected in the set A of remaining possible points that are not assigned as $c_1, \ldots, c_{n-1}$ is

$$\mathbb{P}(x = c_n) = \frac{d(x)^2}{\sum_{x' \in A} d(x')^2}$$

Once we have the desired number of cluster centers, the algorithm is stopped.

2.3 Outlier Detection Without Benign Data

In this approach, once the Kernel K-means algorithm has been completed there will be a set of clusters each with a number of x_i, where the goal is to identify some of the x_i as outliers. The metric used will then be to identify which of the points are farthest away from all of the clusters' centers. The following formula can be used to calculate this distance $\mu_k(\centerdot)$ from each center c_j

$$\mu_j(x_\ell) = K_{\ell\ell} - 2 \sum_{x_k \in C_j} K_{\ell k}/|C_j| + \sum_{x_k \in C_j} \sum_{x_h \in C_j} K_{jk}/|C_j|^2$$

Then for each point x_i, set

$$\mu'(x_i) = \min_k \mu_k(x_i)$$

. With this information for each of the x_i the outliers will present themselves as being significantly further from any cluster than other points, or if there is an expected number of outliers m', the max m' values can be taken as outliers. However, if we assume the outliers will form a cluster of their own, this changes the approach. Then it will be more valuable to just look at which cluster is farthest from the others. In each case, it is usually impossible to know which case to expect unless the number of outliers is very high and number of clusters is extremely low. This approach will be taken at the end of the running of the K-Means algorithm, which presents potential issues in the quality of clusters generated by K-Means, because there is a possibility of the clusters then being affected and distorted by the outliers. Known approaches by Chawla have demonstrated this phenomena and presented a solution of identifying a number of outliers in each iteration that are not counted to the cluster center calculations [6]. However, this assumes knowledge of a specific number of clusters, as well as outliers. The K-Means problem already has one unknown, the number of clusters, which needs to be determined by multiple runs of the algorithm. This approach adds another unknown and therefore makes the problem of detecting the true values for both values a two-dimensional parameter search problem without specific domain knowledge. In the case of our benign data, the number of clusters can be identified by analyzing the curve produced by clustering error as cluster number is increased. While this curve is guaranteed to increase as the number of clusters increases, by analyzing the rate of this decrease the number of clusters can be inferred. In the case of number of outliers, the tolerance for which outliers are identified can be identified by comparing the true positive and false positive rates which will aid in the identification of outliers, as discussed further in the next section. Therefore, each of these problems in the case of known benign data are much better separated, and the difficulty of the problem is dramatically reduced.

2.3.1 Identifying Outliers With Known Benign Data

In order to identify outliers, a clustering algorithm is first run on a benign model of the data. This is done first in order to capture characteristics of the data without outliers, so that the clusters are not skewed by the presence of outliers. These experiments compare the efficacy of the Kernel K-means clustering with Gaussian Mixture clustering using the Expectation-Maximization (EM) algorithm for identifying these original clusters, and use the results for outlier detection. In each of these cases the algorithms give very different information. Where EM algorithm gives the parameters and weights of a mixture of gaussians, Kernel K-means gives just cluster assignments for linearly sepearable clusters in an infinite dimensional feature space. As a result, the distance metric for outlier detection will be different in each case with the calculation of tolerance varying as well. For the EM clustering, in order to use all of the information given by the means and covariances, it is important to not only consider the mean distance from the cluster centers, but also to take into account the different covariances of each cluster. This can be done through the Mahalanobis distance, which when given a cluster C_1 mean, covariance, and point μ_1, Σ_1, x calculates the distance from the cluster as

$$M(x, C_i) = \sqrt{(\mu_i - x)^T \Sigma_i^{-1} (\mu_i - x)}$$

[14] Then, since there are multiple clusters, the distance used will be the $D_{em}(x) = \min_i M(x, C_i)$. For the Kernel K-means algorithm, the assumption is that the results are calculated for normal data in a high dimensional feature space in which the original data is mapped. Therefore, it is necessary to calculate distance in this feature space using the dot product. In addition, the only information used will be information on

the means of the data in these high dimensions as Kernel K-means assumes normal distributions of equivalent variance in this high dimension. Then, there is no need for the inverse covariance to be considered in distance comparisons. Therefore, the distance metric for some high dimensional mapping ϕ and cluster C_i with center c_i is

$$D(x, C_i) = \sqrt{||\phi(x) - c_i)||_2^2} = \sqrt{< x_i, x_i > - 2* < x_i, \mu_j > + < \mu_j, \mu_j >}$$

$$< x_i, c_j >= \sum_{x_k \in C_j} < x_i, x_k >$$

$$< c_j, c_j >= \sum_{x_i \in C_j} \sum_{x_k \in C_j} < x_i, x_k >$$

By taking the minimum of this distance over all clusters, we obtain a new distance for identifying outliers from Kernel K-means. For each of these distances a different set of tolerances can be used. In the case of Kernel K-means, this will range over the minimum and maximum nonzero value of the kernel matrix, which is equivalent to the minimum and maximum nonzero distance in the data set. For the EM algorithm, the tolerance will range over just the minimum and maximum distances between two points for this dataset.

2.3.2 Adjusted Mutual Information

The metric used for identifying the accuracy for clustering the benign dataset against labels, either from traffic data or attack data, was the adjusted mutual information(AMI). The mutual information (MI) between clusters increases with the number of clusters, so AMI is used in cases where there are multiple clusters, where AMI is derived by taking the expected mutual information between two random clusters, and using this as well as the entropy in order to create the AMI, which will be 1 for a perfect clustering, and 0 for a random assignment. AMI is calculated for two clusterings U and V as

$$\frac{MI(U,V) - \mathbb{E}(MI(U,V))}{avg(H(U),H(V)) - \mathbb{E}(MI(U,V))}$$

MI is calculated as

$$MI(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|}$$

Where the probability some object is in cluster C_i is $P(i) = \frac{C_i}{N}$, and the entropy for an entire partitioning C with R clusters is $H(C) = -\sum_{i=1}^{R} P(i)\log(P(i))$

2.3.3 ROC Curves

The Reciever Operating Characteristic (ROC) curve allows for an analysis of the ability to detect outliers over a range of tolerances. In order to identify which tolerance should be used, a training set of both outliers and benign data will be used in order to develop an ROC curve. In order to obtain an ROC curve, all that is needed is a true positive rate and a false positive rate over a range of tolerances. This curve is then plotted and compared to the line $x = y$ on the curve, which is the expectation for the random classifier at different probabilities for each point. The goal is to have the curve for the algorithm to be as removed from this line as possible, and ideally to have a point with perfect detection, $\text{TPR} = 1, \text{FPR} = 1$. Once this curve is plotted, depending on the goal of the algorithm, the optimal tolerance can be selected. In the case of network security, it is desirable to have perfect performance in the true positive rate, with the minimum false positive rate. This is because there is a desire to ensure that even if a few devices are labelled as anomalous which are indeed just demonstrating benign, unobserved behavior, then it is preferable to have to verify this as opposed to having an infected device go unnoticed. Still, this false positive rate must be minimized in order to have a useful detection method, otherwise new methods are required to analyze each point to re-verify the results.

2.3.4 Classification of Point Outliers

The ROC curve is a useful tool for understanding how the outlier detection responds to the change in tolerance. The TPR and FPR can be shown over a variety of tolerances, and therefore a visual examination can identify which tolerance produces the desired rates. In the case where the tolerance needs to be selected without a visual inspection of these ROC curves, there are many options which produce varying results. The objective for this application is to minimize the FPR with a perfect TPR. Therefore, when given the opportunity to tune the algorithm's tolerance with some set of known outliers and benign data, the tolerance could just be selected such that the TPR is 1 and the FPR is minimized. In cases where the tolerance is chosen based on just points and not windows it will often be the case this option will produce very high FPR rates, and potentially artificially low tolerances. It is also a powerful metric to take the TPR − FPR to identify the best tolerance to use. This will give the tolerance that produces the furthest distance from the line $x = y$ on the graph of the ROC curve.

In the case of not being able to have this knowledge of where outliers are in relation to the data, Kernel K-means can provide statistics on the significance of each outlier. This can be done through the identification of the mean and standard deviation of the distance of each point from the center. Since each distance is calculated in order to both assign clusters and provide the total error of the algorithm, this distance can be squared and summed to produce a total squared distance as shown below. Then these two statistics, along with the number of points n can be used to produce a mean

$\bar{\mu}$, and a variance σ_μ^2.

$$\Phi = \min_{c_j} \sum_{j=1}^{m} \sum_{x_j \in C_i} ||x_j - c_i||_2^2$$

$$\Phi_2 = \min_{c_j} \sum_{j=1}^{m} \sum_{x_j \in C_i} (||x_j - c_i||_2^2)^2$$

$$\bar{\mu} = \frac{1}{n}\Phi$$

$$\sigma_\mu^2 = \frac{\Phi_2}{n} - \bar{\mu}^2$$

This equation gives an intuitive tolerance of the average distance plus the standard deviation, $\bar{\mu} + \sigma_\mu$. Or, these can also be used to perform a statistical test on the anomalous data being tested. However, these equations assume that the distances being produced by the Kernel K-means algorithm are normally distributed. This is not the case for the data used, making this specific dataset and distance metric less amenable to this technique.

The histogram in Figure 2.1 below shows exactly why this is not preferable. For the primary dataset being investigated, the set of distances fails to yield a distribution similar to a gaussian. As a result the assumption made by Meidan et al. that the tolerance of $\text{MSE} + \text{sd}(\text{MSE})$ is a difficult one to make in this instance given that the data's distances in this space is visually far from a normal distribution [15].

2.3.5 Classification of Anomalous Devices with Sliding Windows

As opposed to simply detecting whether a point is anomalous, it is necessary to extend this analysis to determining whether a device's entire stream is anomalous. While some streams from benign devices will inevitably have anomalous points, and vice-versa for anomalous devices, it is necessary to identify the entirety of a device's activity as either anomalous or not-anomalous. One simple option is to simply take a

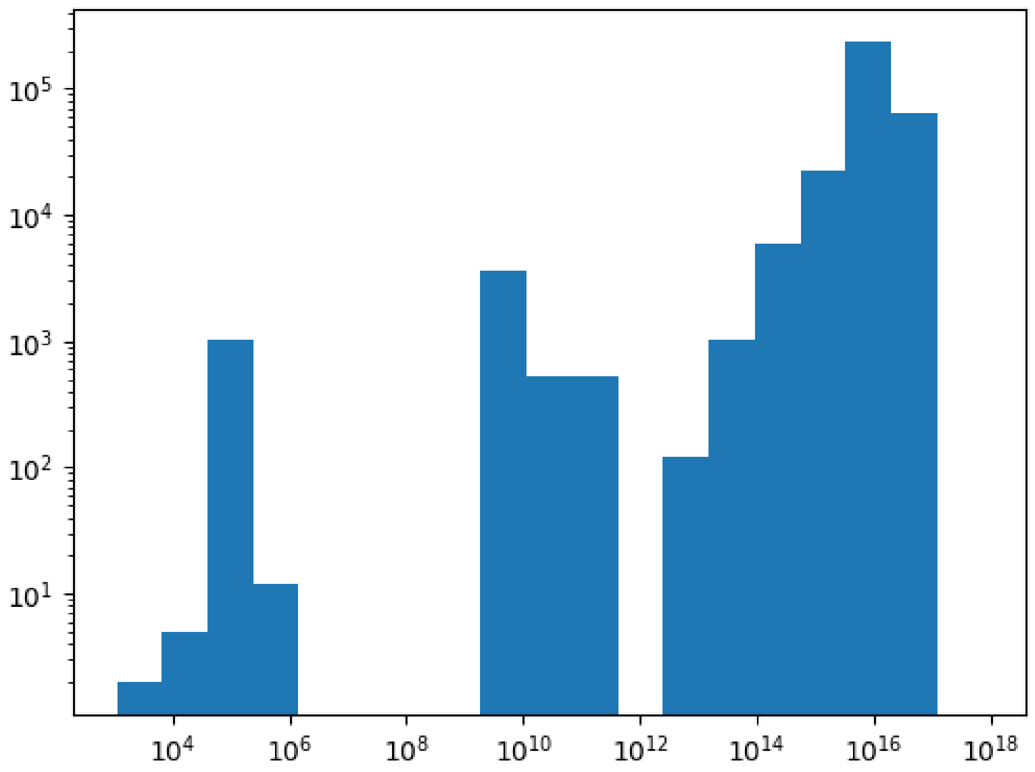

Figure 2.1: Histogram of the Distances Between Points

device as an outlier if the majority of points observed so far are anomalous. However, the question then becomes at what point is it safe to consider the sample anomalous. The approach taken by Meidan et al. is to learn what will not produce false positives by using the benign sample to determine which sliding window produces no false positives on this dataset [15]. Then this sliding window is learned and will mitigate early activity from an anomalous device as labelling the entire device as anomalous prematurely. Then, the minimum value for all sliding windows which produce no false positives when the anomaly detection is performed on the benign dataset is used for the final analysis. Then, when a window is taken to be an anomaly, the same approach is used for determining an anomalous device based on point anomaly detection. Therefore, once a majority of windows are considered anomalous, the device can be considered anomalous. By moving from points to windows, it is easier to ensure that a device will have less erroneous false positives. This is because benign activity determines the window rate such that activity that appears anomalous in the benign dataset should not make a device appear as erroneously anomalous because of the length of the sliding window. However, in the case of the tolerance being too low or too high, this may affect the results of the original sliding window being detected. However, the sliding scale provides a method for counteracting poor tolerance selection. In the case of a high tolerance, the sliding scale will become smaller, as fewer false positives are detected.

2.4 Timing and Scalability Considerations

2.4.1 Kernel K-Means

The main concern in the timing of Kernel K-means for large data is the calculation of each cluster's mean inner product with itself, and the calculation of K, the Kernel

Matrix. The equation for the inner product is

$$< c_j, c_j >= \frac{1}{|C_j|^2} \sum_{x_h \in C_j} \sum_{x_k \in C_j} < \phi(x_h), \phi(x_k) >$$

This has an asymptotic runtime of $O(n^2)$. The calculation of K can also be done naively by looping over all possible combinations of rows of data and filling in the entries one by one. However, each of these are drastically sped up by computing in parallel. In this algorithm, this is done by many of *numpy*'s built in optimized matrix and array functions [18]. Therefore, to make the inner product more efficient, selecting a set of elements with the desired cluster assignment from K can be done by this package. In addition, the sklearn.metrics packages contains the RBF kernel which has an efficient implementation of the algorithm for creating K given some set of data $\mathbb{X}$. It does this by taking for each x_i, x_j,

$$||x_i - x_j||_2 = \sqrt{x_i^T x_i - 2 * x_i^T x_j + x_j^T x_j}$$

[19] This method allows for efficient computation using parallelized array operations, as opposed to utilizing for loops. Therefore, then the computation of the multiplication by the bandwidth parameter as well as exponentiation can be done on the resulting matrix as a whole as well using similar methods. The algorithm requires memory usage of $O(n^2)$ in order to store the Kernel Matrix. This cannot be reduced in this matrix as the rbf kernel is never zero, making it impossible to use many sparse matrix methods of storing matrices.

In addition, once the Kernel K-means has been run on the normal model of the data set, the outlier detection still needs to be run. It is likely that this detection for the entire dataset will need to occur over a number of tolerance values, t. Therefore, the runtime will be increased depending on the breadth and accuracy expected in the search for the optimal tolerance. In order to identify the mean distance and variance of the tolerance, the calculations of distance from each cluster center as well as squared distance are computed as the algorithm runs and can be stored without any additional cost. However, in the case of Kernel K-Means this is not a safe assumption, as using this method works best with a normal assumption on the distribution of distances. Figure 2.1 shows that this is not actually a valid assumption for this data and as a result this method should not be utilized in this instance.

2.4.2 Attack Descriptions

The work by Meidan et al provides data for the following attacks. For two devices, the Mirai botnet attack was unavailable. There were five different attacks given for tis botnet. As described in the paper describing the dataset by Meidan et al. they are as follows: " 1) Scan: Automatic scanning for vulnerable devices 2) Ack: Ack flooding 3) Syn: Syn flooding 4) UDP: UDP flooding 5) UDPplain: UDP flooding with fewer options, optimized for higher PPS (packets per second)" [15].

The other malware utilized was the Gafgyt botnet which goes under a variety of names, commonly BASHLITE, but is also known as Q-Bot, Torlus, LizardStresser, and Lizkebab [15]. The attack descriptions taken from the Meidan article are: "1) Scan: Scanning the network for vulnerable device 2) Junk: Sending spam data 3) UDP: UDP flooding 4) TCP: TCP flooding 5) COMBO: Sending spam data and opening a connection to a specified IP address and port" [15].

2.4.3 Implementation of Outlier Detection

For some point $y \in \mathbb{R}^d$ being evaluated as an outlier, the distance between it and the mean of some cluster center c_i is as such:

$$||\phi(y) - \phi(c)||_2^2 =< \phi(y), \phi(y) > -2 < \phi(y), c > + < c, c >$$

From previous analysis, it is shown that the product $< c, c >$ would take $O(n^2)$ time to compute. Thankfully, this inner product is computed already in the computation of Kernel K-Means and does not need to be repeated. Therefore, the most expensive computation is the calculation of the inner product $< \phi(y), c >$. This value can be computed for some cluster C as $\frac{1}{|C|}\sum_{x \in C} K(x, y)$ which only takes $O(n)$ time to compute at most, where n is the number of rows in the benign dataset. The main limiter in this case will be the number of outliers, and assuming there are more points that need to be tested as outliers (n_0), than the number of points the dataset was trained on (n), the overall time for outlier detection will be $O(n * n_0)$, which is greater than $O(n^2)$ for the running on the benign dataset if $n_0 > n$. In most cases this cost is more acceptable because the number of outliers can be computed over a greater period time. While there are potentially more points to test in total, they do not need to be tested as a batch in the same manner as the benign dataset.

The runtime of the outlier detection can also be mitigated by the parallelization of matrix operations that calculate distance. In the overall process of outlier detection, either running Kernel K-Means on the benign dataset or outlier detection can be the slowest operation, depending on a number of factors. The learning the benign dataset will take longer if the parameter search is done over a wider range and over narrow intervals. It will be beneficial to ensure that the search for the proper tolerance is conducted efficiently, so that the Kernel K-means does not need to be calculated for many different tolerances. In addition, if the initial assignments are close to the final solution, then the Kernel K-means will take less iterations and thus time to converge. This is why the usage of K-Means++ as initialization is valuable, as it has a proven capability by its creators to have this effect for K-Means [2]. Despite these optimizations, the runtime is still asymptotically $O(n^2)$. However this can be again mitigated by performing outlier detection in parallel, over either the tolerances or data rows. The amount of data vectors is usually significantly higher than any amount of tolerances that need to be tested, so it is beneficial to perform it over this portion if both is not possible. However, in most practical instances due to the number of of points in the potential outlier dataset the algorithm is desired to test on, the time will scale with the size of that dataset, and the tolerance range required. With an extremely large dataset of potential outliers, the timing results will be dictated by the efficiency of the outlier detection.

Chapter 3

Network Data Testing

3.1 Data Processing and Analysis

The problems involved with data representation are taking network data and giving a vector of $\{X_i\}_{i=1}^{n}$ which the algorithm can effectively detect all possible harmful utliers on.This data must have certain properties. Specifically, it would be ideal for each X_i to be formed in a manner where each X_i contains an event which can be labeled as anomalous or not. In this data featurization, the method used allows for a variety of different types of activity to be represented through different parameter values.

For the majority of the analysis in this project, the dataset obtained from Meidan et al., available in the UC Irvine dataset repository, was utilized [15] [13]. For this particular dataset, the original data was created from a model network. Nine devices were connected to this network, and then benign network data was collected. To provide anomalous data, the devices were infected with two different botnets, and data from each of the attacks was captured. For each of these devices and attacks, .pcap data was captured of different streams which created a new point of data for each packet of data coming in or out of the device. From this raw network data, incremental statistics were then calculated. The statistics were incremental because any sliding window statistics would have taken more time and memory [15]. The statistics that were calculated are for the packet size, packet count, the jitter from the stream, and inbound and outbound packet size. For each type of information the sum, the squared sum, and number of packets is recorded as each packet arrives. If a statistic is not applicable for a stream it is entered as simply zero.

These statistics are calculated on a dampened exponential window so that before each new packet's info is added to the statistics, there is an exponential factor of $e^{-\lambda * t}$ multiplied to the statistic so far, where t is the time passed since the last observation and λ is a tunable parameter. These statistics are calculated for different λ, and for the aggregations over everything leaving the source IP, the source MAC-IP, channel, and the socket connection. Then the additional statistics of mean, covariance, magnitude, radius, and weight when they are respectively applicable are calculated [15]. This occurs for five different values of λ and results in 23 different stream statistics, resulting in 115 statistics in total being produced that summarize the traffic for the different streams. A number of these statistics are entirely zero if they are not applicable to that specific stream. These values are calculated with a number of different decay values of λ as well [15]. This allows for the vectors produced to represent a variety of time frames therefore giving a better chance of capturing anomalous activity over different time frames. Using these different values and variety of statistics allows for a summary of network data into vectors X_i that capture an array of information that can be clustered.

3.2 Testing Kernel K-means on Benign Network Data

3.2.1 Testing Methods Overview

The ability of clustering algorithms to capture the activity within a network can be evaluated in a number of ways. The goal will be to identify different devices, activity, and unusual activity using clustering. While this method will not be used in the outlier detection, it will demonstrate the possibilities of Kernel K-means in modeling activity for these datasets. Two different methods are used to separate the data and test how this separation matches given labels for the data in order to understand how accurately the clustering separates these different types of activity. In the data from UC Irvine,there are nine different types of malicious activity and one class of benign activity [15]. The data is split into nine attacks carried by two botnets, and one class of normal activity for all the devices. For the types of devices, there are six different webcams, two smart doorbells, and one smart thermostat [15]. Each of the possible comparisons for the different tests will be done over 1,500 data points. The testing will occur over the range of values present in the euclidean distances between points in the dataset. There arefferent values of γ tested for each dataset, with two clusters to fit the given number of unique labels.

First, the testing will try to identify clusters based on the nine different types of malicious activity compared to the respective device each attack is launched from. Documentation indicates that the greatest varying activity will be between attacks and benign data, so comparing malicious activity to benign activity is expected to yield higher AMI values [15].

Second, the clustering will use activity to identify clusters based on different device types, and checking if the normal state of the network can be separated based on the clusters for the webcams, doorbells, and thermostat. These should have different activity in duration of use, so the statistics with strong weighting on earlier packets should show strong differences. However, identifying benign device activity is expected to be more difficult than the last task as the benign network data does not have as significant a difference across devices. While the clustering may detect distinct clusters, it is possible that these clusters will actually be different types of activity that the devices share.

3.2.2 Cluster Analysis I

The results for identifying a botnet's activity on an infected device and the same device while it is operating normally are shown below. This analysis was performed on the first device in the group, the Danmini Doorbell, and the Gafgyt udp flood attack. The results over a variety of bandwidth parameters are given in Figure 3.1 to show the increase and decrease of the AMI as the bandwidth parameter γ approaches the optimal range. The Kernel K-Means was run with both the implementation used and a reference implementation given by Mathieu Blonel on github to show the accuracy of the program [3]. These results demonstrate both the accuracy of Kernel K-means on these more complex datasets, as well as the difficulty of clustering IoT data.

Tables 3.1, 3.2, 3.3, 3.4 demonstrate this maximum AMI for each of the device's benign data against each attack. It is averaged for each different device for Tables 3.1 and 3.2, each different Mirai attack in Table 3.3, and each Gafgyt attack in Table 3.4. Therefore, the drastic differences in the ability to use Kernel K-Means to distinguish certain attacks is evident from this analysis. The variety of values in the attack is far greater than the difference of values present in each device average. As a whole, the data indicates that while there will be difficulty in clustering certain values, the Kernel K-Means has the capability to differentiate this complex data.

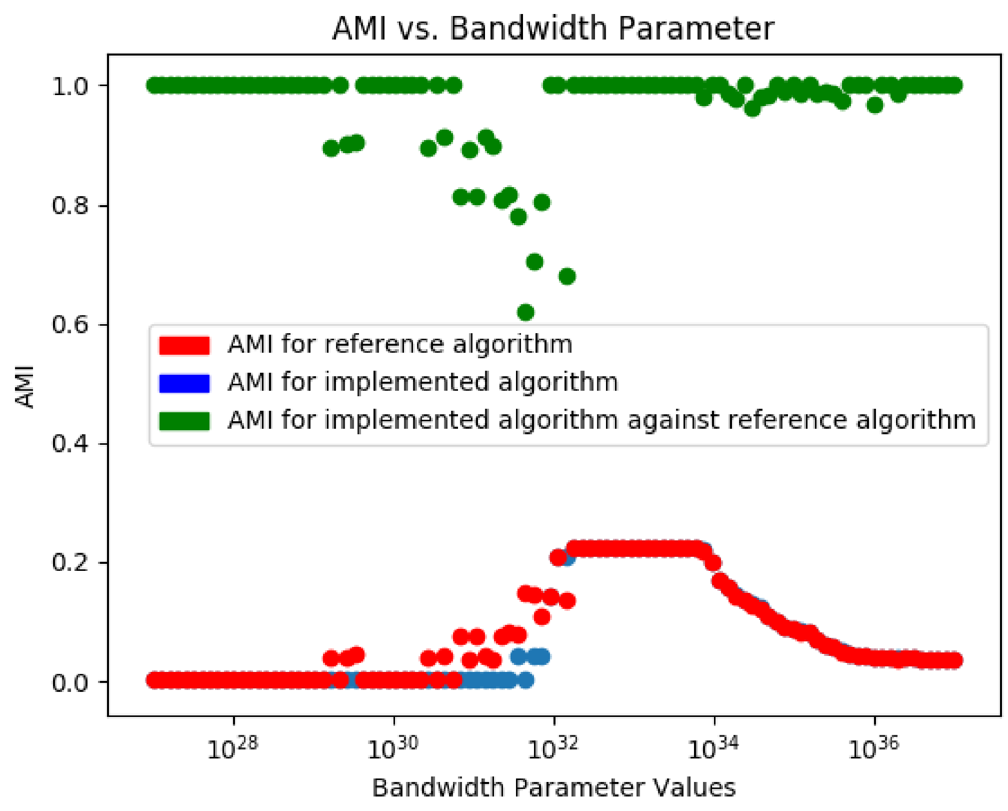

Figure 3.1: AMI for reference implementation: red, AMI for implementation: blue, AMI for implementation given reference as truth: green. X axis is logscale. It records the value of $1/\gamma$, for 200 different values of $1/\gamma$.

Table 3.1: Average AMI Devices For 1-5 All Attacks

Device Names	Danmini Doorbell	Ecobee Thermo-stat	Ennio Doorbell	Philips B120N10 Baby Monitor	Provision PT 737E Security Camera
Avg AMI	0.14	0.14	0.18	0.13	0.15
# of devices	9	9	4	9	9

Table 3.2: Average AMI For Devices 6-9 All Attacks

Device Names	Provision PT 838 Security Camera	Samsung SNH 1011 N Webcam	SimpleHome XCS7 1002 WHT Security Camera	SimpleHome XCS7 1003 WHT Security Camera
Avg AMI	0.15	0.21	0.13	0.15
Device Count	9	4	9	9

Table 3.3: Average AMI for Mirai attacks

Attack Names	Mirai udp	Mirai syn	Mirai ack	Mirai scan	Mirai udpplain
Avg AMI	0.04	0.04	0.04	0.30	0.04
Attack Count	7	7	7	7	7

Table 3.4: Average AMI for Gafgyt Attacks

Attack Names	Gafgyt junk	Gafgyt scan	Gafgyt udp	Gafgyt tcp
Avg AMI	0.05	0.15	0.29	0.30
# of attacks	9	9	9	9

3.2.3 Cluster Analysis II

The results for these comparisons are drastically different than the results discovered for comparing attack data to the different benign datasets. The overall low AMI across all of the averages indicates a very weak ability for Kernel K-Means to identify device labels for any set of comparisons. In fact, the maximum AMI for any of the device comparisons was .08, which was for the SimpleHome XCS7 1003 WHT Security Camera and the Provision PT 737E Security Camera. The fact that these two devices serve the same function and are similar products shows that the ability to cluster IoT data based on purely device is difficult. The expected variance in different device classes as shown by Sivanathan et al. is confirmed in this research [22]. However, the low AMI across devices indicates very similar activity between the devices. The ability to cluster between some attacks relative to this maximum AMI between devices points to the conclusion that the device data cannot be effectively aggregated for outlier detection, as some of the subtle differences necessary to perform outlier detection require the benign data to model the cluster data separately.

Table 3.5: Average AMI by Device 1-5 For Clustering on All Pairs of Devices

	Danmini Doorbell	Ecobee Thermostat	Ennio Doorbell	Philips B120N10 Baby Monitor	Provision PT 737E Security Camera
Avg AMI	0.03272	0.022856	0.019112	0.050927	0.034443

Table 3.6: Average AMI by Device 6-9 For Clustering on All Pairs of Devices

	Provision PT 838 Security Camera	Samsung SNH 1011 N Webcam	SimpleHome XCS7 1002 WHT Security Camera	SimpleHome XCS7 1003 WHT Security Camera
Avg Ami	0.019701	0.014498	0.014476	0.038829

3.3 Testing results of Outlier Detection

3.3.1 Testing Outlier Detection For Toy Datasets

Testing the outlier detection capabilities of both the EM algorithm and Kernel K-means was done first on small datasets from scikit-learn [19]. The two datasets used were the “make blobs”, and the “make circles” datsets. The expected clusters are visualized along with the additional simulated outlier and non-outlier data on which the outlier detection was performed.

The first dataset tested is shown in Figure 3.2. For this dataset, the ROC curve after using the EM algorithm for outlier detection with the Mahalanobis distance is presented in Figure 3.3. This ROC curve demonstrates there exists a tolerance for which the algorithm is able to classify all of the outliers with no false positives.

For the Kernel K-means algorithm, the ROC curve is shown on Figure 3.3. This shows that for any of the different tolerances tested, the outlier detector was never able to have false positives, and so with a high enough tolerance was able to also perfectly predict outliers.

Next, for the “make circles” dataset, the data is visualized in Figure 3.4. The outliers are actually within the two circles; created to make detection more difficult, and the more sophisticated clustering will have an advantage. The ROC for the EM algorithm on this dataset is in Figure 3.5. It shows that this algorithm only achieves perfect accuracy on all outliers when approximately half or more of the points that

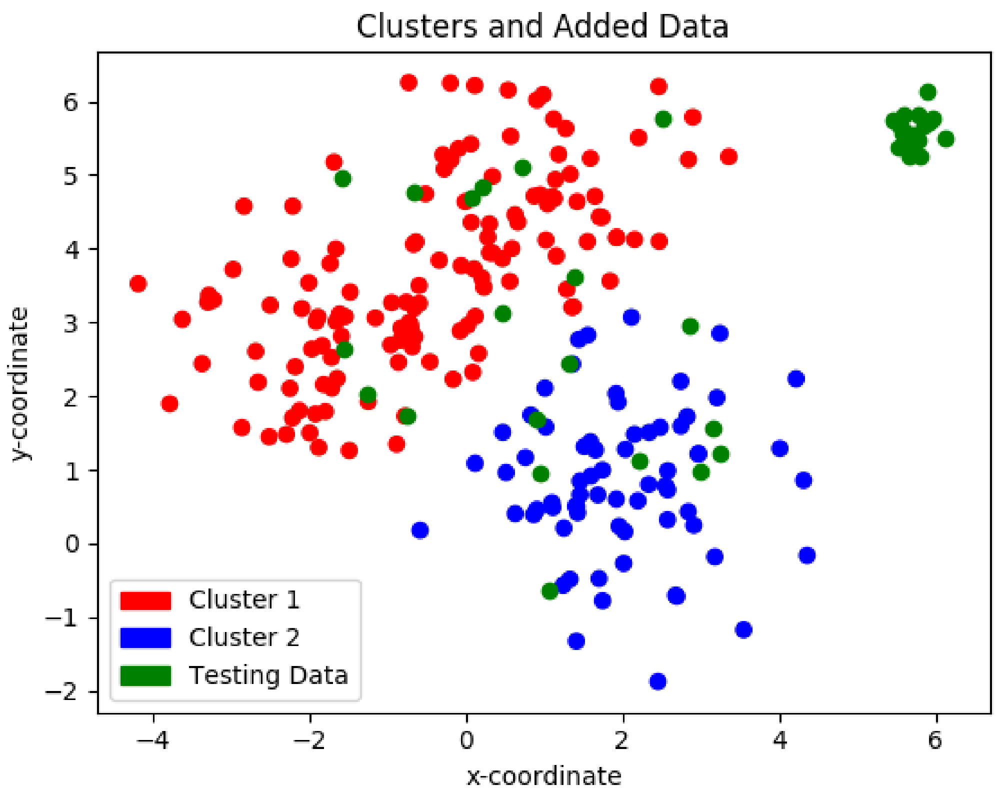

Figure 3.2: Shows toy dataset used for the initial outlier detection test comparisons between EM and Kernel K-means

are given as part of the clusters of activity are identified as outliers as well. However, Figure 3.5 shows that there is a tolerance that achieves perfect outlier detection with no false positives.

3.3.2 Testing Analysis for Toy Datasets

On the first dataset, "make blobs", the EM algorithm performs well since the simulated data used is close to or derived from a mixture of gaussians. This is exactly the underlying assumption that the EM algorithm depends on and therefore it can achieve a high rate of accuracy on the outliers. However, even in this dataset, Kernel K-means with a properly tuned bandwidth parameter is able to identify outliers on a wider range of tolerances at a high rate without incurring any false positives. This lack

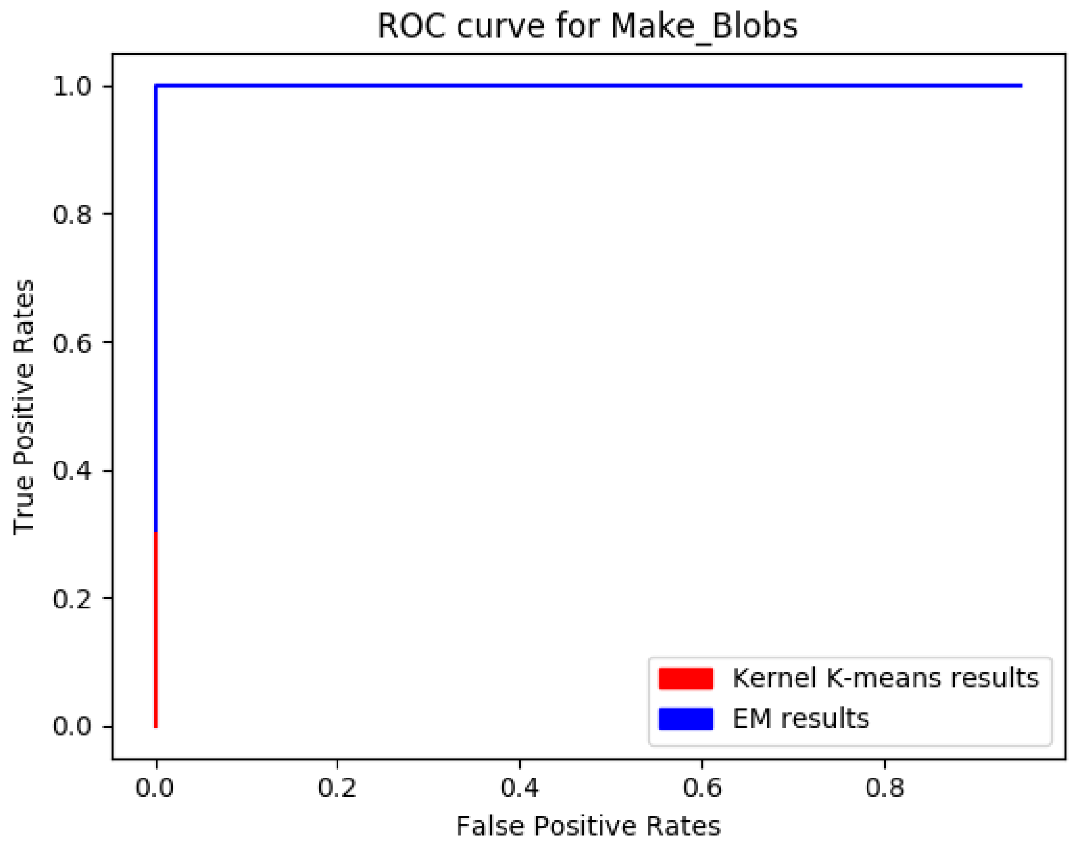

Figure 3.3: ROC for EM algorithm and Kernel K-Means on the “make blobs” dataset

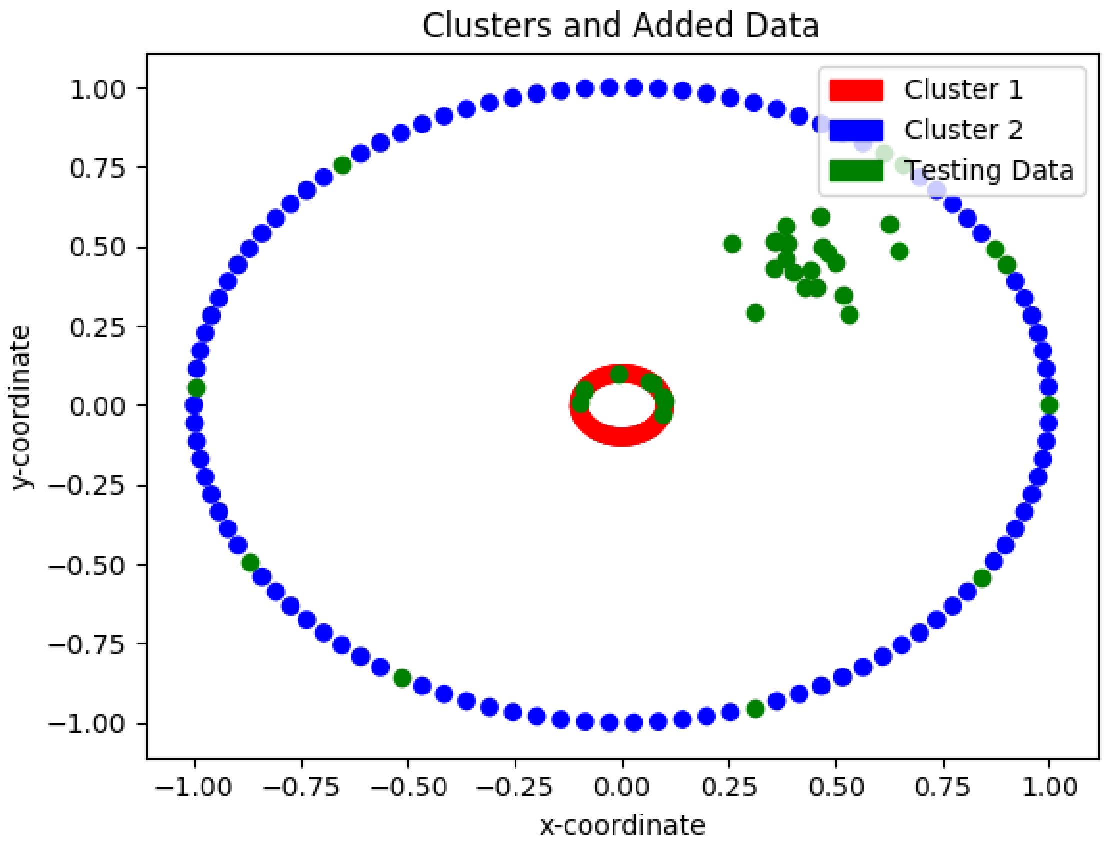

Figure 3.4: Shows toy dataset "make circles" used for the initial outlier detection test comparisons between EM and Kernel K-means

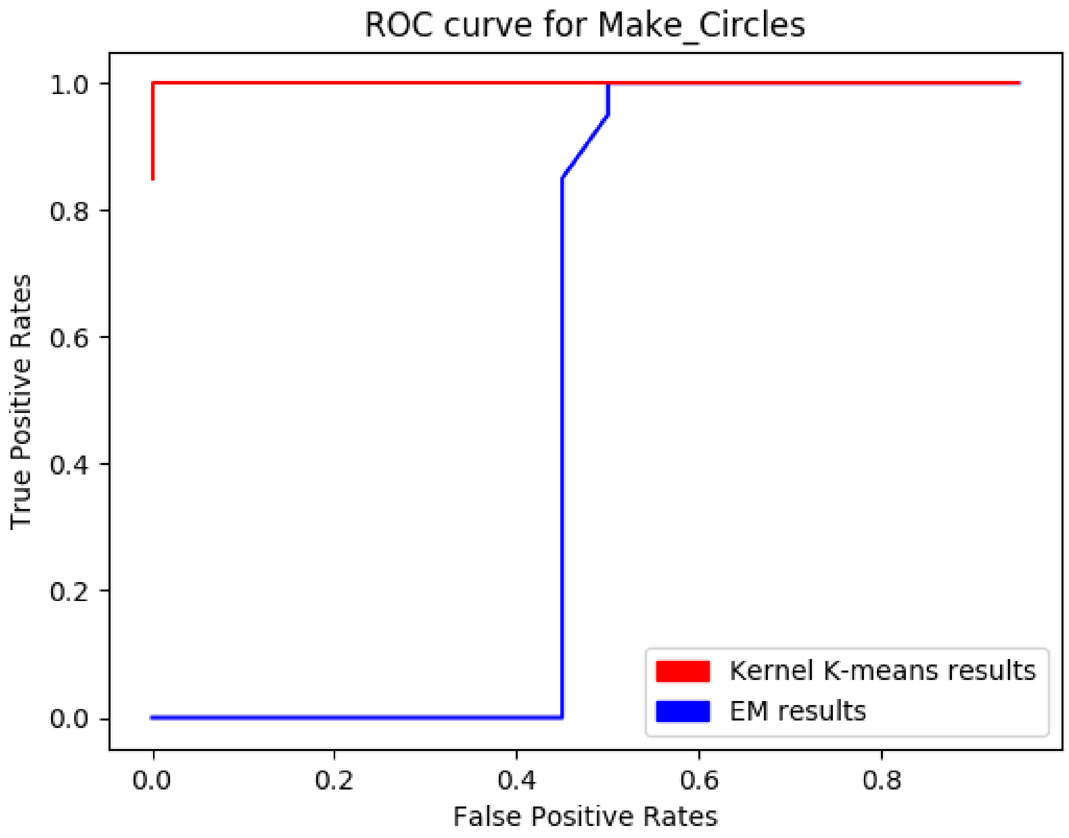

Figure 3.5: ROC for EM algorithm and Kernel K-means on the “make circles” dataset

of sensitivity in tolerance makes the Kernel K-means algorithm easier to use in this application. On the second dataset, Kernel K-means shows improved performance due the need for fewer assumptions on its inputs to be effective. This is because the EM algorithm cannot capture the unusual structure of data effectively, due to its limiting assumption that the data is just a mixture of gaussians. In this case because the data is not a mixture of gaussians or even remotely similar, the EM algorithm is relatively inept at identifying outliers. It only identifies outliers with a relatively low ratio of false positives to true positives at best. Here, Kernel K-means still shows that even in the range of tolerances tested, it is highly successful at detecting outliers, and can detect outliers perfectly at some tolerances. However, the range of tolerances is definitely lower than with the "make blobs" dataset indicating the relative difficulty of finding the outliers in the second dataset for any algorithm. The performance with a properly tuned bandwidth parameter is the same between the datasets, showing the Kernel K-Means outlier detection should be far more generalizable.

3.4 Outlier Detection on Network Data

3.4.1 Detection of Point Anomalies

This portion of the research was performed with the purpose of being able to understand how successful the implementation is on the data and what types of anomaly detection processes will be most effective. The benign dataset was trained on 1,500 points over 15 different γ parameters in order to model the benign traffic for each device. Tables 3.7, 3.8, 3.9, and 3.10 contain the results averaged over each device and Then a testing set of 3,000 points of half anomalous and half benign data were used to test the anomaly detection capabilities. Multiple numbers of clusters were tested, but just two clusters produced the best results on anomaly detection. The results indicate that outlier detection can be done under high certainty over a long period of time, and the potential for further detection is promising. The ability of the algorithm to detect at worst an average difference of .629 TPR-FPR for any attack shows that using simple techniques for anomaly detection will create powerful results. This conclusion follows because with this difference, it is possible to use simple rules to perform anomaly detection on devices as a whole. The concern regarding the premature detection of a device as anomalous is not as great a concern based on this analysis as much of the traffic in the beginning of a stream is not determined as anomalous in either benign or anomalous streams. Therefore, once the majority of the traffic is shown to be anomalous, the stream can be safely considered anomalous.

The concerns with these results are with the specific devices which do not respond well to point anomaly detection with Kernel K-means. The fact that certain devices had such low scores indicates that the benign data clustering algorithm was unsuccessful on specific types of data. Specifically, either the Kernel K-Means was unsuccessful at reproducing the geometry exhibited by the data, or the parameter selection techniques failed at detecting specific clusters. However, the ability to detect well across all such attacks indicates that the techniques are unexpectedly relatively independent of attacks contrary to what was indicated by the AMI data. This is likely because the anomaly detection method employed is able to pick up small differences within the data points, given an effective clustering. In the comparisons where the AMI for the first cluster tests were relatively low, the anomaly detection for many of these instances is able to achieve effective detection. The consistency across attacks compared to inconsistency across devices indicates the main limitation is the parameter selection.

Figure 3.6 shows the ROC curves for all different combinations of devices and attacks. This collection of curves shows different categories of effects produced by the algorithm. The collection which is concerning is the selection of curves which are at or a little below the $x = y$ line. This shows areas where the anomalous activity and benign data are identical with reference to the model created from the benign data. The other sections of the graph show more potential. All other lines on the graph have some point where the max of TPR-FPR is greater than .5. Therefore, for each of these lines the anomaly detection is highly successful at a certain tolerance.

3.4.2 Parameter Selection Results

The function $f(P)$ on some partition of points P defining inter- and intra- cluster differences averaged over all clusters was used for parameter selection in this test. The ability of this method to identify the optimal γ for anomaly detection results were identical to methods using portions of the testing data in training. Compared to locating the value of γ which maximizes the resulting TPR-FPR from anomaly detection, the maximization of $f(P)$ was able to give the same final results with no information from the testing data.

Table 3.7: Average Max TPR-FPR Per Devices 1-5

	Danmini Doorbell	Ecobee Thermostat	Ennio Doorbell	Philips B120N10 Baby Monitor
Device Num	9	9	4	9
Device Avg	0.0713	0.9967	0.996	0.9967

Table 3.8: Average Max TPR-FPR Per Devices 6-9

	Provision PT 838 Security Camera	Samsung SNH 1011 N Webcam	SimpleHome XCS7 1002 WHT Security Camera	SimpleHome XCS7 1003 WHT Security Camera
Device Num	9	4	9	9
Device Avg	0.994	0	0.6753	0.9964

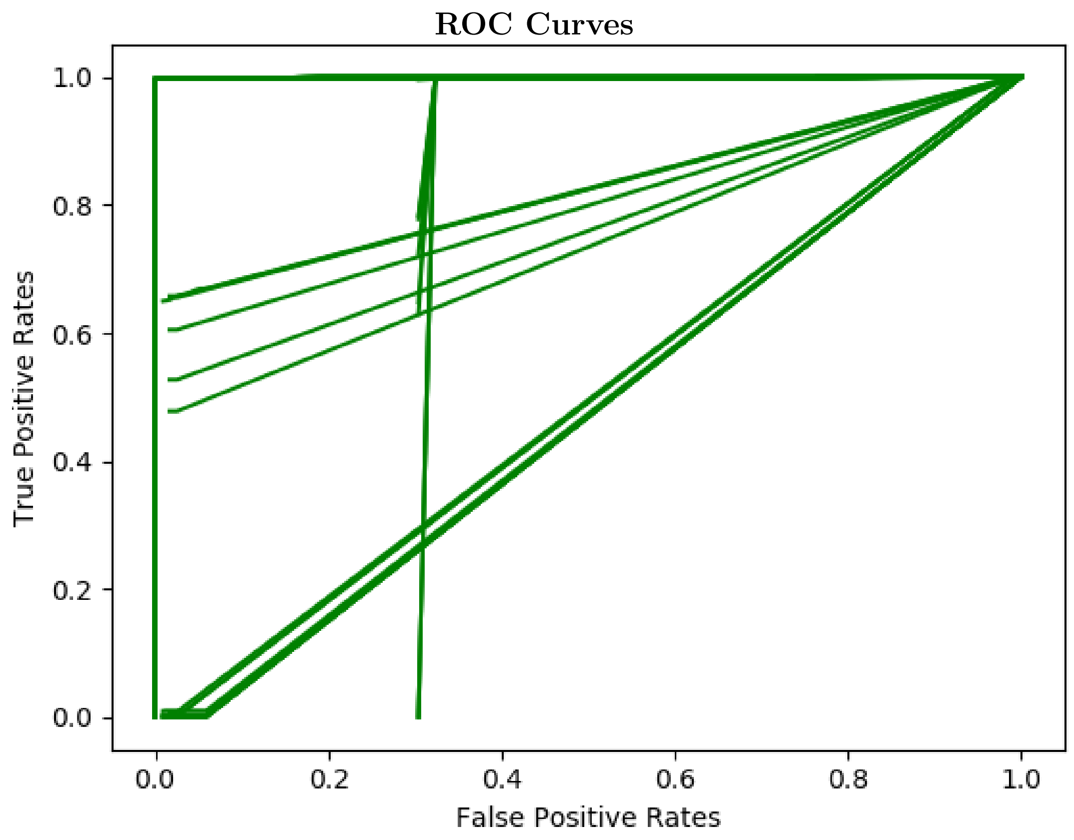

Figure 3.6: ROC Curves for all device and attack Combinations

Table 3.9: Average Max TPR-FPR Per Mirai Attacks

	Mirai udp	Mirai syn	Mirai ack	Mirai scan	Mirai udpplain
Attack Num	7	7	7	7	7
Attack Avg	0.7494	0.7383	0.757	0.6666	0.7312381

Table 3.10: Average Max TPR-FPR Per Gafgyt Attacks

	Gafgyt junk	Gafgyt scan	Gafgyt udp	Gafgyt tcp
Attack Num	9	9	9	9
Attack Avg	0.739	0.7702	0.629	0.629

Chapter 4

Conclusion

4.1 Application Recommendations

The Kernel K-Means based outlier detection has shown high levels of effectiveness over a variety of attacks and devices. As a result, the primary practical implementation for it will be in large-scale networks which are being newly implemented. Since these new networks will not have been infected by botnets or attackers the data will create the benign dataset. The addition of new devices later in the network's lifetime can be done by collecting data on each device and using this data to create a new model for each device as added. Therefore, the method becomes scalable for larger networks, as each device's addition only scales linearly to the overall cost. Compared to other methods, the results obtained by the techniques are shown to be competitive over different attacks. The results lagged a little per device, for both this project and other research. They each struggled on different devices in this dataset, indicating that the techniques from this project and from other papers should be combined depending on the device. In addition, if there is any capability to perform testing or fake botnet attacks on the network to provide some small subset of testing data, this can be harnessed in order to provide the guaranteed best possible results over all devices.

The parameter selection techniques employed for the γ parameter were shown to be as empirically effective but access to the supervised data will always provide better guarantees for results.

4.2 Summary of Results

4.2.1 Clustering Techniques

This project identified a number of advances in using Kernel K-Means in IoT network analysis. First, it established the ability of Kernel K-means to differentiate between the activity of certain devices and attacks. While Kernel K-Means demonstrated a limited capability to identify between different types of benign traffic, this provided further information on the structure and similarities of different traffic data. The AMI were uniformly low, it indicates that the clusters obtained by Kernel K-Means were entirely independent of the traffic data labels, and that the partition of this data in activity is far stronger than the labels being tested for.

This analysis carried into the results obtained for different attacks and traffic data. Where there were still attacks for which this clustering was not particularly successful, it still indicated a much stronger ability to use clustering to detect this anomalous data from benign data as opposed to the ability to determine different types of benign data. The overall results were still varying, ranging from a high AMI of .324 for Mirai botnet tcp flood attack on the SimpleHome XCS7 1003 WHT Security Camera to low results of .0179 for the Mirai botnet's udpplain flood attack on the same device. From this analysis, Table 3.3 and 3.4 depicts the average AMI for each attack, again indicating the relative difficulty and ease of conducting anomaly detection on different attacks. The similarity in these results across devices indicates the ability to detect anomalies at a high rate over a wide range of devices through Kernel K-Means, as the results are relatively invariant to device, and depend far more on type of attack.

Specifically, there was a distinct difference between the ability to identify data from the two botnets tested. The results were far more inconsistent across different attacks for the Mirai botnet. While the scanning attack for this botnet had one of the highest results across all the attacks, the ability to cluster for the Gafgyt botnet was overall much higher. The most elusive attack from the Gafgyt botnet was the junk attack. Despite the other attacks for this botnet having a relatively high AMI this attack was still difficult to manage. The important conclusion from this is indicating which attacks and devices will be difficult to use. In general it can be expected Mirai will be more difficult to perform this analysis on, as the initial discovery suggests.

This is promising in extending this method to larger networks with a wider variety of devices, and with the introduction of new devices to the market. However, it also indicates the difficulty of using tests based on the correct detection of different benign traffic patterns as a method for unsupervised parameter detection. While if there were a strong correlation between the clusters Kernel K-Means selects in the data and the device labels, it would be beneficial to use these labels to determine which γ values produced beneficial results, the actual results indicate a different direction for parameter selection is needed. For supervised learning, the γ values which produced high AMI values for clustering the benign data can be used for anomaly detection and give results that can have better guaranteed results than the parameter selection based on intra- and inter-cluster distances. However, the ability of this method to identify the correct parameters is not as effective as maximizing TPR-FPR for a simulated anomaly detection with attack data and empirically the intra- inter- cluster distances provide the same results with less information. Therefore, it is more of an indicator that the Kernel K-Means can be used in this setting as opposed to being effective parameter selection or anomaly detection.

4.2.2 Anomaly Detection

The use of Kernel K-Means as the basis for anomaly detection provides a number of challenges in implementation through managing both the runtime needs of the algorithm and its parameter selection. This project provided manageable solutions to these problems which allow for the Kernel K-Means to be viable as the basis for an anomaly detection method. The results on the anomaly detection confirm this. While they are not uniformly effective across all devices, they still maintain high levels of effectiveness across attacks. This provides confidence for specific devices that Kernel K-Means and the parameter selection are effective. In addition, the parameter selection heuristic presented gives results in anomaly detection without knowledge of anomalous data that are extremely promising. In fact, the heuristic identified for evaluating clusters has the ability to identify γ just as well as the methods based on having the additional knowledge of the attack data in the anomaly detection process. Being able to identify this solution as an unsupervised method which is competitive with supervised learning methods for this parameter value is invaluable, as it gives further promise to use Kernel K-Means effectively in a variety of contexts.

The overall results for using the point outlier detection technique on this specific data was successful based on the high results in TPR-FPR. This value was over 0.5 on average for all attacks, which shows the point anomaly detection works extremely well. Most importantly, it works well enough to be able to differentiate between the anomalous data and non-anomalous data to identify devices as a whole. Working to tune the algorithm on the devices that the method was unable to cluster would still require additional research, most likely to develop a more fine-tuned parameter search in γ or number of clusters.

4.3 Impact of Results

The two main components of this project, the analysis of network data clustering as well as the subsequent anomaly detection, both provided novel analysis in the realm of IoT network analysis. The results from clustering the benign and anomalous datasets with labels showed the weaknesses in this approach as well as the prevalent structures in the IoT data. While many of the attacks displayed that they could effectively be clustered when combined with devices, there were many others that did not share this capability, which was unexpected. The results obtained for the anomaly detection portion also displayed a more general capability of anomaly detection compared to clustering. This finding reinforces the methodology used as opposed to attempting outlier detection based on clustering the potentially anomalous dataset. However, in the case of IoT networks the difference between anomalous data and benign traffic is better uncovered by performing the anomaly detection following the clustering. This further validates the importance of obtaining the benign dataset in network anomaly detection.

In addition, for determining characteristics of IoT network traffic patterns, this project provided evidence that activity patterns are quite strong across devices. Especially compared to the benign dataset, the AMI scores across devices were nowhere near as high as the scores obtained for comparisons between the attack and benign data. As a result, the ability to identify different types of benign traffic data based on flat clustering methods seems to be a difficult problem. The range of γ tested for each comparison was extremely large, and due to the lack of results achieved it can be concluded that there are many similarities between the data activity. Specifically, by showing the primary partition of the data is not between labels of the devices this indicates there are very general patterns within IoT data as a whole.

Finally, being able to determine that anomalies can be effectively detected by this algorithm was the main conclusion of the testing. The results are very promising for further work in employing Kernel K-Means in this field, just not for using it as a sole means of outlier detection. Kernel K-Means is extremely effective when paired with a given benign dataset and anomaly detection method. Many of the computational complexity costs are more manageable if using parallelized packages and can be formulated with mostly parallelizable operations. This allows for the results to scale better than the $O(n^2)$ asymptotic runtime.

4.4 Further Research

4.4.1 Feature Selection

The implementation of feature selection dramatically affects the impact of the clustering algorithm and outlier detection. Different attacks and data leaks can vary greatly in attributes, so being able to define a set of such statistics for the different streams which can capture the necessary variety of attacks is difficult. The approach taken by Meidan et al. to create the data used in this project is to take any statistics which can be computed incrementally [15]. This approach allowed for the feature extraction to be efficient, and enabled the computation of a variety of valuable statistics from the network data. However, there are other approaches. One such possibility is using different window models, a set of approaches which still demand more computation but the runtime and memory is bounded by the length of the window. If the maximum time it takes for a specific attack to reveal itself is known, then that time becomes the maximum time that is required to compute the sliding window. Similarly to how the value of λ is adapted for the exponential window model, the ability to change the size of the sliding window has a similar effect, and could produce different results.

The network flow can also be viewed as a time series, so it is also possible to use entirely different families of techniques to determine the attributes of the flow. This different feature extraction can be utilized to identify different aspects of the raw data, with extremely different implications than the statistics collected by using the exponential window model. This method would give features which would represent the changes in the raw benign traffic data and attack traffic data in a manner which would not necessarily be as dependent upon window size or λ selection. The features would have very different meaning, and on their own would provide their own novel intuition about the data.

4.4.2 Clustering Techniques

The most important advancement in this paper beyond the current scope of techniques is improving the Kernel K-Means clustering algorithm's efficiency. The most evident technique for this would be the ability to find efficient approximations for the bottleneck of computing the cluster center inner product with itself, or being able to find a low dimensional representation of the gram or distance matrix as a whole. This has been done for sparse Kernel PCA, and while those techniques cannot be directly translated, there is work that can be done towards a similar goal since such representations and approximations are abundant [9]. Especially in identifying and minimizing the representation of the cluster centers, it would improve the potential runtime of the algorithm. Being able to approximate this would drastically change the outleir detection method. The reason it would have to be a different approach than the one used in the PCA papers is that these approximations are designed to efficiently find important eigenvalues, where the approximations needed for Kernel K-Means would be designed to enable an accurate sum of different elements. This is an area with less mathematical background than approximating eigenvalues, making the solutions more heuristics and with fewer mathematical guarantees.

4.4.3 Parameter Selection

In addition to the algorithm improvements, there can be further research into effective methods of unsupervised detection of the optimal parameter for clustering the benign data. The heuristic used in this project is one of many, and the problem of identifying the correct parameter for γ can be further researched for Kernel K-Means. The correct parameter has a dramatic effect on the final clustering, therefore if this approach can be narrowed beyond a grid search it would drastically cut down on the runtime of the overall algorithm. This would also require a greater understanding of the relationship between the values in the gram matrix K and the expected result of the gamma value. This understanding would also drastically affect the breadth of the grid search, however, the optimal γ parameter still depends on the structure of the data. Therefore, there would be some expected similar structure across different types of network data, making this analysis important for further use of Kernel K-Means in IoT data.

4.4.4 IoT Anomaly Detection

One of the important factors in applying these datasets to the IoT is the time over which the algorithm is applied. In this project, the algorithm was run on a fairly small network which was simulated. Ideally, the algorithm could work on much larger networks and eventually be tested en masse. The capability to do this would require the data for a much larger network assuming no anomalies or just relatively few for the techniques discussed to be effective. In addition, being able to determine the impact of each outlier on the effectiveness of the model for its implementation on real world data sets would be helpful to understand. However, while these checks would confirm the validity of this algorithm for larger network instances, the results obtained for the smaller version indicate that the method is flexible and robust for when expanded.

In addition, for each specific device, benign traffic data should be collected as they are added to the network. While the devices were sure to be clean of any botnets, then the new model on this traffic data would be introduced and would test for outliers related to the new device. Making the anomaly detection more efficient using knowledge from the original benign data clustering would also be beneficial. This information would also require knowledge on how quickly devices can be infected once they enter a network, and the employment of methods that ensure the initial traffic collected is entirely benign.

Finally, by testing on real network data and being able to analyze the anomalies, it would also be interesting to identify other events that are labeled as anomalous. In the case that any of these events are not actually botnets, it is valuable to understand what other activities are similar to the activity of different botnets, data leakages, or other cybersecurity vulnerabilities. In the case that a larger portion of anomalous activity is being detected that should be included in the model, it can be added in the same manner as adding an additional device to the network.

Bibliography

[1] Yousef Amar, Hamed Haddadi, Richard Mortier, Anthony Brown, James A. Colley, and Andy Crabtree. An analysis of home iot network traffic and behaviour. *CoRR*, abs/1803.05368, 2018.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[3] Mathieu Blondel. kernel_kmeans.py. GitHubGist, 2014.

[4] Ismail Butun, Burak Kantarci, and Melike Erol Kantarci. Anomaly detection and privacy preservation in cloud-centric internet of things. In *IEEE International Conference on Communications*, 06 2015.

[5] Colin Campbell. An introduction to kernel methods. *Studies in Fuzziness and Soft Computing*, 66:155–192, 2001.

[6] Sanjay Chawla and Aristides Gionis. k-means–: A unified approach to clustering and outlier detection. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 189–197. SIAM, 2013.

[7] Corero. Corero ddos trends report q2–q3 2017. *Corero*, 2017.

[8] Earl Crane. Is the internet of things becoming the internet of risk? *Emergent*, July 2017.

[9] Rudrajit Das, Aditya Golatkar, and Suyash P. Awate. Sparse kernel PCA for outlier detection. *CoRR*, abs/1809.02497, 2018.

[10] Ernesto De Vito, Veronica Umanità, and Silvia Villa. An extension of mercer theorem to matrix-valued measurable kernels. *Applied and Computational Harmonic Analysis*, 34(3):339–351, 2013.

[11] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556. ACM, 2004.

[12] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE, 2018.

[13] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[14] Michiel Hazewinkel et al. Mahalanobis distance. *Encyclopedia of Mathematics*, 3, 2001.

[15] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Dominik Breitenbacher, Asaf Shabtai, and Yuval Elovici. N-baiot: Network-based detection of iot botnet attacks using deep autoencoders. *CoRR*, abs/1805.03409, 2018.

[16] Rob van der Meulen. Gartner says 8.4 billion connected "things" will be in use in 2017 up 31 percent from 2016. *Gartner. Letzte Aktualisierung*, 7:2017, 2017.

[17] Hamid Mirvaziri. A new method to reduce the effects of http-get flood attack. *Future Computing and Informatics Journal*, 2(2):87 – 93, 2017.

[18] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–.

[19] 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, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Piyush Rai. Clustering: K-means and kernel k-means. Machine Learning (CS771A), August 2016.

[21] Amnon Shashua. Introduction to machine learning: Class notes 67577. *CoRR*, abs/0904.3664, 2009.

[22] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Characterizing and classifying iot traffic in smart cities and campuses. In *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*, pages 559–564. IEEE, 2017.