Intruder Alert: Dimension Reduction and Density-Based Clustering for a Cybersecurity Application

by

Benjamin Burr

A thesis submitted to the
Faculty of Graduate and Post Doctoral Affairs
in partial fulfillment of the requirements for the degree of

**Master of Science in Mathematics
with
Concentration in Statistics**

Carleton University

Ottawa, Ontario

©2020

Benjamin Burr

Abstract

This thesis examines the use of Principal Component Analysis, Robust Principal Component Analysis, and simple autoencoders for dimension reduction on a synthetic cybersecurity dataset. Each is tested as a precursor to Independent Component Analysis. Stable independent components are obtained by iterative randomized starts to FastICA and selecting the centroids of the hierarchically clustered components. A density-based clustering method is then applied to the results with the goal of isolating malicious observations from benign ones using greatest distance between centroids as a heuristic metric of success. The method is then applied to a real-world cybersecurity dataset from an industry partner.

Acknowledgements

No project of this scale happens in a vacuum. To start with, thank you to Dr. Shirley Mills for taking a chance on me. Your kindness has been invaluable and your expertise irreplaceable. Thank you for supervising me and for introducing me to data mining.

Thank you to my committee members, Drs. Dave Campbell and Aaron Smith, for your encouragement, your gentle questions, and your feedback on both the written portion and the defence. It was likely the nicest reviewer feedback anyone in academia could expect to see, and I will cherish that (and likely never publish again so that I can go out on a high note). Thank you to my chair, Dr. Song Cai, for keeping us all on track and overseeing the process. And thank you to our silent Zoom guru Jonathan Lindsay. If things went badly, my wife told me to text her “SOS” and she would unplug the router so that I could claim there were connection issues. I’m glad it all went well so I didn’t have to throw you under the bus.

Thank you to Hazem Soliman, Geoff Salmon, and Rank Software for approaching us with this project. Thank you for the expert guidance, feedback, and support throughout the early stages, and for the generous use of your data. I hope you got something worthwhile out of this work.

Thank you to Shelly Wang for your hard work and support during the first phase of the project as well. I am proud to have a little paper with our names on it and know that you will go on to much bigger things.

And thank you to my loving wife Sabrina, without whose coaching, guidance, support, advice and time management I would still have been writing this thesis in 2030. In fact, you’re looking over my shoulder right now, complaining about how long this is taking.

Contents

Abstract	i
Acknowledgements	ii
List of Figures	vi
List of Tables	ix
1 Introduction	1
1.1 Computer Networks	3
1.2 Intrusion Detection Systems	5
1.3 Thesis Goals	7
2 Literature Review	9
3 Data	14
3.1 Raw Rank Software Data	14
3.1.1 Data Format	15
3.1.2 Network Alerts Data	15
3.1.3 Network Alerts Data - Unnesting Comparison	16
3.1.4 Network Events Data	18
3.2 Public Datasets	18
3.3 CICIDS2017 Dataset Description and Pre-processing	19
3.4 CICIDS2017 Feature Selection	22
4 Methods	27
4.1 PCA for Dimension Reduction	27
4.1.1 Singular Value Decomposition (SVD)	28
4.1.2 Selection of an Appropriate Subspace in Classical PCA	29
4.1.3 Global Dataset Processing for Classical PCA	32
4.2 Robust PCA for Skewness	33
4.3 Autoencoders	34
4.3.1 Choice of Nodes in the Hidden Layer	38
4.4 Independent Component Analysis (ICA)	39

4.4.1	ICA: A Little More Linear Algebra	40
4.4.2	Choice of ICA Algorithms and Parameters	41
4.4.3	Choosing the Number of Independent Components to Estimate	42
4.4.4	Icasso	44
4.5	Word Embeddings for Categorical Traffic Identifiers	46
4.5.1	Word2Vec	47
4.5.2	IP2Vec	48
4.5.3	Evaluating IP2Vec Embeddings	49
4.5.4	Word2Vec Parameter Selection	50
4.6	Density-Based Clustering	51
4.6.1	Density Estimation, Thresholding, and Hierarchy	53
4.6.2	Similarity and Dissimilarity Measures	53
4.6.3	DBSCAN and HDBSCAN	54
4.6.4	Cluster Validation with HDBSCAN	55
5	Results	57
5.1	Summary Statistics	57
5.2	Classical PCA Results: Global Dataset	61
5.3	Classical PCA with Word Vectors on Daily Data	67
5.3.1	PCA: Monday	68
5.3.2	PCA: Tuesday	70
5.3.3	PCA: Wednesday	73
5.3.4	PCA: Thursday	75
5.3.5	PCA: Friday	78
5.4	RPCA: Daily Results	80
5.5	Autoencoder Results	81
5.6	Results: ICA	85
5.7	Clustering Results	88
5.7.1	PCA Clustering	89
5.7.2	RPCA-ICA Clustering	91
5.7.3	Autoencoder-ICA Clustering	93
5.7.4	Pairwise Cluster Distances	94

6	Discussion	98
6.1	PCA, RPCA, and Autoencoder Discussion	98
6.2	IP2Vec Embeddings	104
6.3	ICA and Clustering Discussion	105
6.3.1	A Limitation	111
6.4	The Rank Data: Putting it all Together	111
7	Conclusions, Limitations, and Future Directions	122
	Appendix A: Data Schemata	126
	Alerts Schema	126
	Events Schema	128
	CICIDS2017 Schema	129
	References	133

List of Figures

1	Typical representation of an autoencoder with a single hidden layer. This amateur version based on an image at https://www.jeremyjordan.me/autoencoders/ , used with permission.	36
2	Illustration of a Dendrogram based on hierarchical clustering of independent components over 100 ICA runs on the CICIDS2017 Tuesday data.	53
3	Illustration of an HDBSCAN cluster. The Noise point is not density-reachable or density-connected to the rest of the points. The points encircled in orange are density-connected and are considered core points of the cluster. The points encircled in blue are density-reachable to each other and to the core points, but are considered border points in HDBSCAN. In the HDBSCAN algorithms, the core density-connected points are retained for clusters, whereas the density-reachable ones are counted as noise or outliers. The radii of the circles represents the ϵ -neighbourhood of the points. Image idea based upon an illustration in [78].	54
4	Hierarchical clustering of dissimilarity matrix based on Pearson correlation between variables.	61
5	Screepplot of variation explained for PCs using the classical PCA procedure on 58 numerical variables from the CICIDS2017 dataset, mean-centered and standard deviation-scaled, against the expected variation explained under the Broken Stick model. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.	65
6	Modified biplot (points removed, so more like a uniplot) for PCs using the Classical PCA procedure on 58 numerical variables from the CICIDS2017 dataset, mean-centered and standard deviation-scaled.	65
7	Screepplot of variation explained for PCs against the expected variation explained under the Broken Stick model using classical PCA on the Monday data from the CICIDS2017 dataset. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles. . .	68
8	Screepplot of variation explained for PCs against the expected variation explained under the Broken Stick model using classical PCA on the Tuesday data from the CICIDS2017 dataset. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles. . .	71
9	Screepplot of variation explained for PCs against the expected variation explained under the Broken Stick model using classical PCA on the Wednesday data from the CICIDS2017 dataset. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.	73

10	Screeplot of variation explained for PCs using classical PCA on the Thursday data from the CICIDS2017 dataset, mean-centered and standard deviation scaled, against the expected variation explained under the Broken Stick model. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.	76
11	Screeplot of variation explained for PCs using classical PCA on the Friday data from the CICIDS2017 dataset, mean-centered and standard deviation scaled, against the expected variation explained under the Broken Stick model. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.	78
12	First three dimensions of the 10-node hidden layer of an autoencoder run on the Friday CICIDS2017 dataset for a single hour from 4:00pm to 5:00pm. Malicious traffic is green and benign traffic in red.	84
13	Anomaly scores based on Reconstruction MSE for each observation in the Friday CICIDS2017 dataset.	85
14	First two dimensions of the autoencoder hidden layer for the Monday CICIDS2017 dataset. .	86
15	First two dimensions of the autoencoder hidden layer for the Wednesday CICIDS2017 dataset.	87
16	First two dimensions of the autoencoder hidden layer for the Friday CICIDS2017 dataset. . .	88
17	Multidimensional Scaling plot of independent components for ICA run on the Tuesday Autoencoder subset of 12 dimensions.	89
18	Icasso dendrogram of independent components for ICA run on the Tuesday PCA subset of 12 dimensions.	90
19	Three independent components for averaged ICs based on Icasso on the Tuesday autoencoder dataset.	91
20	Three IC dimensions coloured based on cluster assignment in Figure 11. Red observations are Cluster 1. Green observations correspond to Cluster 2. Blue observations correspond to Cluster 3. Black observations correspond to outliers.	92
21	Clustering for RPCA-ICA data. Red and green represent the four malicious clusters with only one benign point included, while 48 of the black points represent malicious observations that were misclassified as outliers.	93
22	Ridge Plot of the first principal component by traffic type.	100
23	Ridge Plot of the first RPCA component of the Tuesday dataset by traffic type.	100
24	Ridge Plot of the first RPCA component of the Wednesday dataset by traffic type.	101

25	First three dimensions of the central hidden layer of an autoencoder run on global dataset. Red observations correspond to benign traffic while other colours correspond to malicious traffic.	103
26	First two dimensions of the hidden layer of an autoencoder run on Wednesday dataset. Red observations correspond to benign traffic while other colours correspond to malicious traffic.	103
27	First two dimensions of the central hidden layer of a deep autoencoder with 3 hidden layers run on Wednesday dataset. Red observations correspond to benign traffic while other colours correspond to malicious traffic.	104
28	First three PCs from PCA on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Grey observations reflect other clusters. Black observations reflect outliers not contained within a cluster.	106
29	First three PCs from RPCA on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Grey observations reflect other clusters. Black observations reflect outliers not contained within a cluster.	106
30	First three dimensions of the hidden layer of the autoencoder on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Grey observations reflect other clusters. Black observations reflect outliers not contained within a cluster.	107
31	First three PCA-ICA components on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Black observations reflect all other clusters and outliers.	108
32	Ordered distance between cluster centroids from the PCA-ICA clustering in which three clusters were found. Red points represent distance from malicious cluster centroids to other cluster centroids. Grey points represent distance between benign cluster centroids. Black points represent distance to the outliers centroid.	109
33	RPCA-ICA cluster distances. Red points represent distances from malicious cluster centroids to other cluster centroids. Green points represent distances from clusters with centroids nearest to malicious centroids to other clusters. Grey points represent distances between benign cluster centroids.	110
34	Dendrogram of the clustered independent components of the example Rank dataset.	117
35	MDS plot of the clustered independent components of the example Rank dataset.	118
36	Plot of three of the clustered independent components of the example Rank dataset after HDBSCAN is run. The red observations are the three clusters farthest from any other cluster. The green clusters are the nine clusters that were farthest from the three red clusters. Black observations are outliers not included in any cluster.	122

List of Tables

1	CICIDS2017 Sample Estimates for Numeric Variables	58
2	Results of Classical PCA with mean centering, scaled using standard deviation, on Global CICIDS2017 Dataset with 58 Variables	62
3	Results of Classical PCA on the CICIDS2017 Monday Dataset	68
4	Results of Classical PCA on the CICIDS2017 Tuesday Dataset	71
5	Results of Classical PCA on the CICIDS2017 Wednesday Dataset	74
6	Results of Classical PCA on the CICIDS2017 Thursday Dataset	76
7	Results of Classical PCA on the CICIDS2017 Friday Dataset	79
8	Results of Robust PCA on the CICIDS2017 Daily Datasets	81
9	Robust PCA Score Distance (SD) and Orthogonal Distance (OD) Measures on the CICIDS2017 Daily Datasets.	82
10	RPCA H_0 Subset Correspondence to Traffic Types on CICIDS2017 Daily Datasets	82
11	PCA-ICA Cluster Assignment	90
12	PCA-ICA Outlier Scores Above and Below 0.7	90
13	PCA-ICA Probability Scores above and below 0.95	90
14	RPCA-ICA Cluster Assignment	92
15	RPCA-ICA Outlier Scores above and below 0.7	92
16	RPCA-ICA Probability Scores above and below 0.95	92
17	Autoencoder-ICA Cluster Assignment	93
18	RPCA-ICA Outlier Scores above and below 0.7	94
19	Autoencoder-ICA Probability Scores above and below 0.95	94
20	Pairwise Distances Between Clusters: Tuesday Data and PCA	94
21	Pairwise Distances Between Clusters: Tuesday Data and RPCA	95
22	Summary Statistics: Rank Data	113
23	Results of Classical PCA on the first Rank Dataset	114
24	Pairwise Distances Between Clusters	118

1 Introduction

The modern world is increasingly filled with technology, much of it connected to some form of network. Such networks are vulnerable to exploitation and attacks from malicious actors who would like to cause mischief, perform sabotage, or steal information. The field of cybersecurity has been built up over many decades to prevent the unauthorized use of technology. Cybersecurity addresses a broad range of devices and types of information technology, from the pervasive wireless Internet of Things that connects personal devices together to industrial and medical electronic systems to computer networks. We are surrounded by technology in our daily lives, from computers to Bluetooth headphones, to smart homes. Many of the devices we use every day are capable of connecting to a wireless network. Aside from personal privacy concerns that leave us wondering how much of our daily conversations and behaviours are being recorded and stored by these devices, or how much those applications and devices are tracking our movements, we must also be concerned about external actors penetrating the security measures employed by these devices. Furthermore, since we can be certain that many companies are storing our personal data for their own use, we must also be concerned about the defences being used by these companies.

In this thesis, I am interested particularly in the security of computer networks with Internet connectivity, as are typically found in modern business entities, offices, universities, and on a small scale in many homes. Network Intrusion Detection Systems (NIDS) are often used to analyze network traffic flows from network devices, either by comparing them against pre-specified signatures that could indicate an attack or by looking for patterns that deviate from a baseline model that describes the typical behaviour of such traffic. Often, malicious actors will take extra care to conceal their actions when attempting to penetrate a network. An attack may go undetected if the security monitors are not equipped to detect the particular signature of the attack or exploit used, or if the attack is made to blend in with the traffic based on the characteristics used to detect anomalous events. This means that information could be stolen and exploited by a company's competitors or malicious third parties without anyone at the company being aware of the breach; or personal information such as financial details or identifying information could be used fraudulently without a person knowing anything about it.

Security groups hope to find better ways to automatically detect these breaches, to stop them before they become problems, and especially to prevent new and previously-unseen attacks from succeeding by analyzing the activities on the network. Making this more difficult is the ingenuity of malicious actors, the constant evolution of their tools and methods, and the unavailability of public exemplar data containing such attacks out of concern for privacy and security and due to simple lack of availability. Creating such a data set with labelled attacks poses a major challenge in this field. Labelling attacks requires a significant investment

of resources for a company to generate real network traffic and to safeguard it, and a significant investment of time by experts to examine the data and label undesirable events or to distinguish between malicious anomalous traffic and acceptable traffic that only appears anomalous. Couple that with the possibility that attacks may not occur, or that only well-known types will be seen, or that the experts will not identify them based on what they see, and the task of creating the data becomes prohibitively difficult.

Simulating such data also comes with its own problems, since it is not feasible to accurately create a single synthetic representation of a real-world network usage scenario that includes more than a handful of realistic attack scenarios, or that represents the vast range of network environments that could be employed or actions that could be included. Such simulation would also fail to account for the novel and previously-unseen attacks we are most interested in detecting and preventing. Whether simulated or exported from a live network, network security data typically consists of raw network traffic logs that contain information such as packet headers and routing information. Network security data also usually contains output from NIDS that enrich these logs but often does not hold any conclusive identification of an attack. It is unlikely that any real or simulated data for this type of work will be labelled, and if it is labelled it likely will not be made publicly available. Problematically, labelled data is vital for evaluating the strength of a given technique and for training a baseline of normal traffic against which to detect anomalies using many useful available methods.

Network data is received rapidly in real-time and in bulk, often with millions or billions of events being logged every second, depending upon the size of the network. There are vast repositories of network data available to many organizations, but using it to train a supervised detection algorithm is infeasible. To create or to capture such a dataset also means to create something that is immediately out of date, since one could not hope to capture all possible current scenarios and exploits in a world where new and previously-unobserved ones are being developed constantly. Security teams are thus continually trying to keep up. Methods that use static datasets, while useful for forensic analysis, lack the timeliness needed for real-time attack prevention. Static datasets also fail to acknowledge the gradual shifts in data characteristics that can occur over time as businesses change, personnel come and go, and infrastructure is upgraded. The only truly useful solution is to examine this data for abnormalities as it is received and to constantly re-examine it, updating any model being used to account for changes over time, to use the most applicable and up-to-date techniques, and to look for long-term relationships among data points that might indicate slow-moving, periodic, or cyclical malicious activity.

Such activity can indicate persistent compromise of the network alongside the use of high quality obfuscation techniques that allow an attacker to remain active and undetected long after successfully establishing a foothold. This type of activity is known as an Advanced Persistent Threat (APT) and can include several steps

that are often studied in isolation in the literature, but some researchers have attempted to link those steps together to form campaigns of associated malicious activities. Generally, APTs are perpetrated by well funded, well organized, highly effective, and greatly skilled organizations, often sponsored by states and political entities. Some malicious actors have managed to compromise thousands of systems for as much as a decade while avoiding detection[32]. APTs highlight the importance of timely prevention and intervention and retrospective surveillance of past network activity. Detecting such activities is one of the most challenging current problems in cybersecurity.

It is also particularly difficult to quantify APTs in general because each one is unique and may not follow specific patterns. Most cases have been highly specialized to the particular organization beyond a few general steps. Since there is no truly acceptable dataset for APT research available publicly, and a comprehensive attempt to quantify these attacks would require months or years of data instead of the days or weeks of data typically available for general use outside of a security organization, this aspect of the research will be mostly absent in the main body of this thesis. Although APTs were an early motivation for this project, the thesis will focus on anomaly detection for cybersecurity but not on the long-term attack scenario reconstruction necessary to detect APTs. Instead, this thesis will focus on day-to-day detection in an unsupervised manner that leverages the unlabelled data most likely to be available in the real world.

1.1 Computer Networks

For this thesis, networks consisting in general of servers and computer workstations such as desktop and laptop computers will be the primary hosts being considered. Related devices, such as any common peripherals associated with these devices, will be irrelevant in this analysis, mainly for simplicity. What follows is a brief summary of client-server computer network models, the potential vulnerabilities, and the data available to security services for use in safeguarding it. Relevant information is distilled as much as possible without giving more than high-level overview of the considerable complexities of network architecture.

A computer network is a set of electronic devices which are connected together and may share information despite different hardware, software, rules and users. Each device is established on a network domain. Such a domain is controlled by a centralized database, which contains account information and security protocols for the entire network. The centralized database often allows users to connect to their established network account from any authorized device on the network. The network is controlled by a network operating system (e.g. Windows Server 2016, Ubuntu Server, etc.), which performs resource management, controls access, dictates communication rules among devices, and supplies applications and data[89]. All network devices are connected by wired or wireless connections. These connections carry signals that may be interpreted by

the machines as instructions or encoded messages that are passed back and forth.

Devices are often referred to as nodes. Communication between nodes on a network requires shared network protocols. Protocols are rules used to create and terminate connections, address traffic to the correct node on the network, manage data flow to ensure the available resources can handle the speed of the transfer, ensure the order of delivery is correct, detect and correct data corruption, and format the transfer properly for transmission [21]. This thesis mostly considers data that includes Internet Protocol (IP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP). Information is communicated across the Internet in what are called protocol data units, which contain the data to be transmitted as well as metadata such as a header containing the source and destination address, the port number, and the values needed for correct transmission. On the Internet layer of a network, the protocol data units are commonly called packets. Packet headers include source and destination IP addresses, ports, and addressing information, and other information needed to communicate with other network layers and to route packets to the correct destinations as needed. Ports are endpoints at which communicated information may be sent or received. They consist of 16-bit integers ranging from 0 to 99999. Some ports are assigned specific duties by network administrators, while others may be transient and assigned automatically by an operating system.

A host device may wish to send data to another device outside of the network, which is done by routing the data packet through a routing device, such as an Ethernet switch. The routing device will request the proper routing information if it does not have it, then will build a packet with the data and relevant other information. The destination device then unpacks the packet, verifies the destination matches, and extracts the data to pass to the correct application. For example, the network switch may have an application that requests web data from external sources. The reverse series of operations would pass a packet back to the source computer's web navigation application to view a website.

Each node has an IP address, referred to in this thesis as either a host IP address or source IP address for internal nodes connected to a network under consideration, and referred to as a destination IP address for external nodes that are either sending or receiving traffic to our network. A range of valid internal IP addresses is typically made available for a network. Certain ranges of IP addresses for Internet use have been purchased by countries and other large organizations for current and future use, though vast numbers of these reserved addresses are not being used in general. Formerly, IPv4 set all IP addresses as 32-bit numbers of the form xyz.abc.def.ghi. The possible values of each 3-number substring, called an 8-bit subnet, could range from 0 to 255, so 0.0.0.0 is a valid 32-bit IP address, and 255.255.255.255 is also a valid 32-bit IP address. Every value in between these two addresses is valid. These substrings are translated from binary into the form shown. For example, 192.168.0.1 represents the set of four binary octets 11000000.10101000.00000000.00000001.

The first three chunks represent the network ID, while the last chunk represents the host ID, which uniquely identifies the device or host on its specific network. It is standard practice to refer to a network ID by using a 0 for the host ID, so 192.168.0.0 specifies a network, whereas changing the final subnet to a 1 specifies that we are referring to a host ID on that network. A subnet mask hides the network ID from external view[1]. Because there are only 2^{32} possible IPv4 addresses available, all of them have been assigned and there are too few possible addresses for the billions of devices currently in service. As a result, IPv6 was introduced, which uses 128-bit hexadecimal numbers in eight groups of 4 hexadecimal units separated by colons. For example, *fe80 : 0000 : 0000 : 0000 : 897b : 581e : 44b0 : 2057* is an IPv6 address, which translates to the binary string 1111111010000000 : 0000000000000000 : 0000000000000000 : 0000000000000000 : 100010010111011 : 0101100000011110 : 0100010010110000 : 0010000001010111. The empty portions of these strings are often collapsed into two side-by-side colons[21].

The combination of IP addresses, ports, various protocols, and packets, provide a rough description of how data is transferred in a network and on the Internet. A computer network can produce and transmit vast amounts of data across many machines of varying type, location, and technological capability using the methods just described. However, data transfers may be compromised with hidden code, or the packets may be entirely composed of code intended to compromise a host machine by exploiting a vulnerability in the network security protocols, a device's operating system, or a user's credentials. Exploitative activities are the exception rather than the norm, but this does not make them easy to prevent. Even an apparently secure network may be compromised. Wireless and wired signals may be intercepted. Encrypted messages may be decrypted. Even physical locations may be compromised by stealth, carelessness, or poor planning, so a network without external connectivity may still be compromised. It is the job of the network administrators to implement security measures in an effort to minimize the risk of such intrusions occurring and to mitigate the damage a security lapse can cause in terms of unauthorized access to sensitive material, data theft, and data corruption. Early warning and prevention are the key to the tradeoff between allowing the network users enough access to carry out operations while maintaining a secure organization.

1.2 Intrusion Detection Systems

An Intrusion Detection System (IDS) is a security tool meant to monitor a network, detect potential and real external and internal threats and unusual operations, and prevent current and future attacks. Intrusion Detection Systems are often divided into Host-based IDSs (HIDS), Network-based IDSs (NIDS), and Hybrid IDSs. They may also be subcategorized into misuse-based NIDS and anomaly-based or behaviour-based NIDS. Misuse-based or signature-based IDSs are designed to detect attack patterns based on known sig-

natures, rules, and characteristics but suffer from often being unable to handle even minor changes in the patterns. Anomaly-based IDSs are designed to detect deviations or unexpected activity in traffic patterns or behaviour. An IDS based on anomaly detection knows the system it needs to protect and raises an alert when it performs actions it is not supposed to [9]. Using both of these or using a hybrid IDS alongside other security tools is common.

We are interested in anomaly-based NIDS methods for this thesis. Network Intrusion Detection Systems are used to monitor and analyze network traffic for security purposes. They usually consist of a data collector, a manager, and a communication module to receive data for analysis and send back the results[65]. Typically, sensors are located at points on the network that act as bottlenecks for traffic so they can monitor all traffic on the network. For NIDS based on anomalies, comparisons are usually made based on statistical profiles of historical system or user behaviour. The IDS builds a model of the normal behaviour that would be observed on the network, such as frequency or location of logins, data transfer size and frequency, typical sources and destinations, and time-of-day activities occur. Observations of captured packets, headers, traffic volume, and traffic patterns that do not appear to have been generated by the underlying process, signature or behavioural model may be flagged as anomalous based on some threshold or rule put in place by the network administrators, raising an alert. Not all anomalous activity is an intrusion or an attack, although this is the desirable outcome of using an IDS. Instead, four possible scenarios occur:

1. Normal activity does not appear unusual, so it is correctly ignored by the IDS.
2. Approved activity by a non-malicious actor does not match the usual network traffic and the IDS raises an alert (i.e., a false positive).
3. An intrusion may be marked as normal activity by the IDS because the model or system of rules associated with it was not built in such a way that the activity stands out (i.e., a false negative).
4. An intrusion occurs, is marked as anomalous, and an alert is correctly raised by the IDS.

Ideally, we would implement an IDS with which all true intrusions raise alerts and no other traffic is flagged. However, a model cannot capture all nuances and aspects of a dynamic system and we cannot hope to implement a perfect IDS. Normal activities can, and often do, deviate from a model because exceptions occur in situations and in human behaviour. The two incorrect cases, false negatives and false positives, will occur in any NIDS. Security professionals must therefore balance a trade-off between the two. Misuse-based IDSs typically have a low false positive rate but a high false negative rate because they recognize signatures well but cannot recognize anything new or unknown. Anomaly-based IDSs typically have a high false positive rate but a low false negative rate because any deviation or any type of anomalous behaviour, however small or

unobtrusive, may raise an alert. An IDS can allow some deviations from the model to occur to avoid too many false positives, but doing so increases the risk of attacks going undetected if those deviations can be used to the attacker's advantage. In general, we err on the side of too many false positives, since we would rather dismiss alerts that turn out to be acceptable traffic than receive no warning of actual intrusions. However, with so much traffic passing through modern networks, an overwhelmingly large number of alerts may be generated by doing this. A security team cannot keep up with them, leading to what is called alert fatigue[9]. An analyst who is overwhelmed and too tired to deal with all of the alerts may dismiss or simply miss actual intrusions, especially when only a small handful of thousands of alerts are connected to malicious activities.

An anomaly-based NIDS has the potential to allow real-time and novel attack identification while being independent of operating system and not impacting network performance[65]. However, it may also require application-specific protocols, may not be as useful on architecturally fragmented networks, may be overwhelmed by traffic volume, requires a large storage capacity, and cannot analyze encrypted packets. Regardless of such specifics, we want an IDS to not only analyze packet headers of protocols but also to build baseline profiles for the traffic patterns they support so we can look for anomalies in both. This way, if a header exhibits an anomaly it raises an alert; but if the header has been made to appear safe, as is often the case, anomalous use of the communication protocol will also generate an alert[65]. An IDS can read logs of events from routers, firewalls, servers, and other network devices, acting as a focal point for many separate, but potentially connected, pieces of information. It is possible that separately, these sources do not indicate an attack but when taken together they do.

1.3 Thesis Goals

This thesis uses dimension reduction and unsupervised statistical learning methods on network traffic data to locate and separate malicious or intrusive observations from benign ones. To keep the scale of the project feasible, this has been narrowed down to exploring appropriate application of dimension reduction for feature extraction in the presence of malicious events, and using the results of such methods for density-based clustering of those events. To meet the thesis goals, I first implement an exploratory analysis of a synthetic network intrusion dataset, discussed in Section 3.2 to develop an approach that may produce a roughly equivalent outcome on real world data. I use two common decomposition methods, PCA and ICA, and attempt to capture the malicious events in separate clusters using those components. It is hoped that the results will provide another useful approach for security analysts to use in the absence of labelled data. I also consider an embedding method for IP addresses, ports, and protocols to enrich the described methods. The results of this approach on the synthetic data are contrasted with the results obtained on a small portion of data chosen

from a larger dataset of live network traffic provided by an industry partner, with an unknown number of attacks.

Following this introduction, in Section 2 is a review of some of the literature on each of the methods discussed above. In Section 3, the type of data and its context is described, including methods used to validate, transform, and prepare this data for use. Section 4 contains descriptions of the PCA, ICA, Clustering, and Embedding methods used and the methodology used for this project. Section 5 contains the results of applying these methods to the synthetic data and the decisions made throughout the process. Section 6 places these results in context, and Section 7 concludes with remarks on future directions and the improvements that could have been made to this project, deficiencies that came up during the course of the work, and the conclusions that may be drawn. Appendices provide data schemata. Code is available on a private Github repository upon request at <https://github.com/ben-burr/thesis-code/>.

2 Literature Review

There are tens of thousands of papers and chapters in this field covering anything from proposed IDS frameworks to detection of a single type of attack to modelling network traffic and beyond. Trying to define such a model is like trying to hit a moving target that is always slightly out of range and is actively trying to avoid being hit. Regardless, many have tried, so there are hundreds of novel combinations of methods and tools already described in the literature. Network intrusion detection, threat detection, and anomaly detection are all used to describe groups of methods of detecting malicious network events and attacks. Thus, the set of tools that are used to detect anomalies and outliers in other areas may reasonably be applied to the field of intrusion detection, and often are. However, here I only review literature that looks specifically at unsupervised methods of detecting malicious activity in network traffic log data. Of particular interest are methods that allow dimension reduction using PCA, Robust PCA (RPCA), autoencoders, and ICA, and for unsupervised grouping I focus on density-based clustering. There are many other general categories of methods that may be used for intrusion and threat detection, but this is not a comprehensive review.

Principal Component Analysis (PCA) can be considered a latent variable representation of data, where the latent variables capture the uncorrelated components or features that provide an ordered representation of the variation in the raw data. Classically, PCA is linear, but there are nonlinear formulations as well. PCA is discussed more thoroughly in Section 4.1. Using PCA can allow identification of a subspace of this latent representation that allows better identification of groups observations that are of interest, such as malicious traffic events. Davis and Clark[14] identify PCA as a common pre-processing technique to reduce data for intrusion detection, listing several instances where it had been used. Kakavand et al.[38] use PCA for dimension reduction with the average eigenvalue for a dimension-reduction cutoff, a method described further in Section 4.1.2. This is followed by a Mahalanobis Distance Map for classification of HTTP payload anomalies. Ringberg et al.[74] noted that PCA for anomaly detection is sensitive to how traffic measurements are aggregated. Greater aggregation increases smoothing and hides smaller anomalies, while less aggregation increases variability, to which PCA responds. They described an underlying assumption in the network anomaly detection context that the k -subspace relates to traffic trends, such as the work day or week. They also referred to other work that network traffic have low intrinsic dimensionality, so dimension reduction can be a useful first step in analysis.

Bohara and colleagues[8] extracted features that would identify Command and Control and Lateral Movement behaviours from a host communication graph, which represents internal communication between system hosts, then used PCA followed by k-means clustering to identify hosts that were similar to infected hosts. They also used the PCA data to perform analysis on outliers to identify infected hosts. These results were

combined and accuracy was measured by changing a threshold, since the approach was unsupervised. Their approach allowed between 80% and 90% accuracy, although the false positives, despite being few, would be a concern in using this approach. In threat detection, we would prefer to capture all malicious traffic, even if that requires more false positives. Gao and colleagues[22] used adaptive control version of PCA for feature extraction alongside an incremental extreme learning machine as part of an IDS. Li et al. proposed an anomaly detection framework for network traffic that used PCA to extract features and reduce redundancy. The results were passed to a random forest algorithm that also used the weights of the components to inform the branching process of the decision trees[44].

Abdulhammed and colleagues[1] used PCA to reduce the dimension of the CICIDS2017 dataset, which is described in detail in Sections 3, from 81 features to 10 features, then trained various classification algorithms on the reduced data with high accuracy. They also evaluated PCA with only 2 components and applied uniform distribution-based balancing on the data, which increased the accuracy of the classifications. In a review paper, Joshi and Hadi[36] describe the use of PCA for dimension reduction alongside use of a one-class support vector machine (SVM) for classification. Manandhar and Aung[49] similarly use PCA, k-means, then a one-class SVM modelled on the normal traffic to classify new traffic. Platonov and Semenov[67] also used PCA to generate a weight matrix, which could be automatically adjusted and used in the calculation of new parameters to be used in an SVM classifier for an IDS, using one of the DARPA datasets for exploration. They used principal components trained on the attack set alone, which allowed good detection of some types of attacks but not with others. Zhang and colleagues[92] used TCP features and statistical characteristics of network traffic as input to an improved PCA algorithm. Their algorithm reweights the first n principal components to decrease their influence, then carries out dimension reduction. This method is based on approaches used in image processing, but in combination with several classifiers it performed well.

Despite its strengths and widespread use, classical PCA may not be ideal without some modifications. PCA is sensitive to outliers, which can strongly influence the directions of greatest variation that are used to construct the component axes. In the application of network security and threat detection, it might be suspected that malicious events will appear as outliers or introduce greater variation in some dimensions, reducing the effectiveness of classical PCA. Robust PCA helps to address this concern and has been used in several related works. However, RPCA has not been used as extensively as PCA in this application.

An early approach by Shyu and colleagues[80] used a robust estimator of the correlation matrix that relied on trimming the extreme values using Mahalanobis distance first to ensure the estimates of mean and standard deviation were resistant to outliers. Their subsequent classification predictions were around 97% to 99% accurate.

Work by Pascoal and colleagues[64] considered five different approaches to RPCA and chose PCAGRID

and ROBPCA as the best options to deal with data contaminated by univariate or multivariate outliers. They used RPCA for dimension reduction, then classified new observations as anomalies or not based on score distance and orthogonal distance thresholds, with the subspace dimensionality chosen to maximize probability of an observation being accurately classified as an anomaly in the training set. They also found that robust feature selection prior to using RPCA was an essential step to improve performance. Oh and Kwak[60] proposed an alternative to RPCA that uses the generalized mean instead of its special case, the arithmetic mean, and also proposed a generalized sample mean. Their example application was facial recognition, but the two areas overlap in many methodological areas.

Two more recent approaches include the Robust Graphical Lasso or Rglasso[46] and Adaptive RPCA[48]. Rglasso relies on splitting the covariance into a clean sample covariance matrix and a covariance matrix associated with the outliers in the original covariance matrix. It was found to outperform both RPCA and the minimum covariance determinant in the detection of sparse latent effects in a financial data application. Adaptive RPCA attempts to recover the lowest-rank subspace of clean data from corrupt data while retaining geometric structures both globally and locally. Testing was carried out on denoising images, resulting in lower clustering error than RPCA and other comparable methods. In both cases, graphical models were used, and those have been ignored in this thesis, but constructing a graphical model from the network events was considered as a possible next step.

Autoencoders, a form of neural network that act as a generalized form of PCA that does not require linearity, do not seem to be as popular as some other forms of neural network. However, they are still used in certain applications because they can be useful for anomaly detection. One recent application of deep autoencoders[94] was inspired by RPCA. A denoising autoencoder was developed that did not require noise-free training data, instead relying on the loss function. Experiments were carried out on the MNIST handwritten digit dataset with added corruption. There was a large improvement in reconstruction error over standard autoencoders.

Another proposed approach is the Deep Autoencoding Gaussian Mixture Model, which combines a compression network to perform dimension reduction with an estimation network to make predictions using a Gaussian Mixture Model. Two of the experiments were conducted on the old KDDCUP and KDDCUP99 datasets, both of which have been used extensively in threat-detection research. The results of the autoencoder lead to density estimation, lending weight to the use of density-based clustering as a sensible next step later in this thesis.

A sparse autoencoder with two hidden layers and a sigmoid activation function was used for dimension reduction by Abdulhammed et al.[1] on the global CICIDS2017 dataset. The first hidden layer reduced the dimension to 70 features, while the second reduced it to 64 features. The resulting reduced datasets were then

used with supervised learning methods. A deep autoencoder with four hidden layers of decreasing size and using the ReLU activation function was used on the CICIDS2017 dataset[85] and compared to shallow neural networks. The improvement was not large using deep versus shallow networks. Reducing to 10 features still had over 90% accuracy, but failed on some types of attack. The same problems may be seen in the h -subset from RPCA in Section 4.2.

Fernandez and Xu also used deep autoencoders[19] in both supervised and unsupervised intrusion detection on the same dataset as well as the 2012 version, the ISCXIDS2012 dataset. They used entity embeddings of the Source and Destination IP addresses and Ports, which are similar to the IP2Vec embeddings used in this thesis. The autoencoder had three hidden layers. The results were compared against a large number of other modern machine learning methods, showing the effectiveness of the approach to be greater than any other method attempted, sometimes by a large margin, in terms of both true and false positive rates. The use of the embeddings increased the effectiveness by roughly 3%.

Labonne and colleagues also used a deep autoencoder[43], alongside four other neural network architectures, on the CICIDS2017 dataset, comparing the respective anomaly scores. They note that there is an often-missed distinction between traffic classification and attack detection, so they correlate anomaly scores to estimate the probability of a sequence being an attack. This approach can help connect attack sequences together into campaigns or related incidents, something that is highly desirable for security analysts and was an early consideration at the beginning of this thesis that was ultimately not pursued. The autoencoder was the only one they used that detected the FTP-Patator attack in the Tuesday data, although less sophisticated methods may also be used, as shown later in this thesis. It is likely that the ICA step in this thesis is integral to its success, possibly due to the induced independence of the components helping to separate the attack types, but ICA has not been used extensively in this area of research. Palmieri, Fiore, and Castiglione[62] build a traffic model of basic features, with the goal of representing normal traffic dynamics to improve anomaly detection. They collected traffic data from several sensors that included simple simulated anomalies that mirror common attack types. They ran FastICA to separate the sources and carried out several supervised binary classification on the components, with satisfactory results.

Wang, Yang, and Li used ICA in place of PCA[86] on non-Gaussian distributed network and host performance metric data from virtual machines, then used a multi-class naive Bayesian classifier to detect anomalies. Independent Component Analysis was used for feature extraction, resulting in mutually independent components, which are essential for the Bayesian classifier. Feature extraction using ICA outperformed both PCA and LDA, and ICA with the Bayesian classifier outperformed ICA with a decision tree for classification.

Aside from these two works, there were few easily-located applications of ICA to intrusion detection. Possibly this is because it is difficult to determine a number of sources or because PCA is considered effective

enough to use instead, but most likely it is because neural networks are much more popular. Density-based clustering methods are also not the standard in this area. Many papers[8][40][42] mention k-means clustering for intrusion detection. A 2011 review of machine learning-based clustering techniques used in intrusion detection[91] mentions Self-Organizing Maps, k-means, Support Vector Machines, and hierarchical clustering techniques. Zhang and colleagues[93] used fuzzy clustering as part of attack scenario construction from alert logs.

Two papers do include the use of density-based approaches to outlier detection. The first[82] is a local kernel density-based approach where nearest neighbours, reverse nearest neighbours, and shared nearest neighbours are all considered. The Relative Density Outlier Score was introduced to measure distance from clusters. They tested on simulated and publicly available real-world data with apparent success, but further evaluation is likely needed. The second[41] using Hamming densities, which are based on the ratio of k-nearest neighbour distance to Hamming distance. Local reachability distance was defined using the maximum of the two distances, then a local outlier factor was calculated. The algorithm was tested successfully as an intrusion detection system on the outdated KDD dataset, but more in-depth testing would be required.

Each of these methods therefore has some background in this field of application, though often a much more extensive one in other areas. Principal Component Analysis has been used frequently, so a comparison to other methods like the more recently popular autoencoders and to robust approaches is worthwhile. Independent Component Analysis and density-based clustering, particularly a shape-agnostic method like HDBSCAN, have not been used as much for threat detection, but may prove to be worthwhile techniques to pursue. Embedding methods, which are discussed in more detail in Section 4, have not yet been applied as extensively as they could be given the ubiquitous use of text-based log data in modern cybersecurity. Theoretical implications and considerations will be discussed in Section 4, then applied to the CICIDS2017 dataset in Section 5. Next, in Section 3, I discuss how an appropriate public dataset was selected.

3 Data

In this section I describe the data used in the analyses, the procedure used to process the data, and primary features of the data. The genesis of this project was a grant provided by Rank Software (Rank), a software security company based in Toronto, Ontario. Their intention was to explore cybersecurity solutions involving touchless user and entity behaviour analysis (UEBA) to detect unknown or novel threats. Several projects were proposed and multiple directions were explored, but eventually I focused on using an unlabelled corpus of log data on entities of interest to generate an embedding space. The goal was to generate similarities between host machines that could be used to capture further insight into security threats, in light of which they provided us with real-world network security logs from a network they monitor.

The data for the first part of this project originated from a network monitored by Rank. This network has roughly 50 employees using it. The majority of the network hosts use Windows systems. There are several internal servers. Rank uses several network monitoring systems and IDS implementations, such as Zeek¹ and Suricata² that capture network traffic as it travels between each of these hosts and servers, and between these hosts and the internet gateway or between the gateway and these hosts. The traffic captured is bi-directional rather than outgoing or incoming alone.

3.1 Raw Rank Software Data

Rank uses Zeek³, a well-known open source network security monitoring sensor. Zeek monitors network traffic and produces connection event logs. Each connection event represents a bi-directional TCP, UDP, or ICMP connection. These connections capture packet flow, where the origin and the destination each have an identifying IP address and port. The source port is not particularly meaningful for this network because source ports were automatically assigned by the operating system based on availability and were subject to change. However, destination ports often have specific values assigned by the network administrators.

In addition to the Zeek logs, Rank's other sensors capture host-based events and other network events. Suricata⁴ is an open sourced intrusion and threat detection system (IDS), which, alongside other IDSs, is used by Rank's system to analyze the captured events and to generate alerts based on their customized rules.

The data provided by Rank included files that mapped network alerts, files that provided network alert summaries, network alert logs, and network connection event logs. Eventual work was completed on data from September 1, 2019 to October 31, 2019. There was a time gap in the Alerts and Events data between September 20, 2019 to September 25, 2019. The Events data included the first 11 hours of September 21,

¹<https://zeek.org/>

²<https://suricata-ids.org/>

³<https://zeek.org/>

⁴<https://suricata-ids.org/>

2019 and hours 10 to 23 of September 25, 2019, with nothing in between. The Alerts data did not include any data files for September 21, 2019 and September 22, 2019, but did include data for the first part of September 20, 2019 and the last part of September 23, 2019.

3.1.1 Data Format

Data was provided in JavaScript Object Notation (JSON)⁵ and Apache Parquet (Parquet)⁶ formats. These two data formats are common open standard formats, which claim to be easy for machines to parse, storage-friendly, and human readable. Typical output from Zeek and Suricata appears to lend itself well to these data formats. JavaScript Object Notation was originally meant for JavaScript processing for use in data transfer between web browsers and back-end applications, but is now used for data transfer services[21]. It consists of a list of one or more key-value pairs. This format is ill-suited to data analysis due to the potentially deeply nested structure, which I discuss in more detail below.

3.1.2 Network Alerts Data

Network Alerts data for the time period was broken down into a single daily JSON file, for a total of 59 JSON files of 111MB in total size. The alerts for September 21 and September 22 were not present in the data.

A schema can be used to describe the memory organization of this type of data. The main schema for this set of Alerts data can be found in Appendix A.1, and shows the deeply-nested structure of this sort of data. The schema for these alerts is deeply nested to begin with. Items at the deepest level like the “interesting” object include even deeper nesting, as shown in Appendix A.1. While these features can provide rich augmentation for an alert that could help a security analyst evaluate the threat to the network or a particular host, they make it much more difficult to work with this kind of data due to inconsistent length and difficulty extracting useful information for analysis. Regardless of how “human-readable” the data might be, we want it to be human-usable instead, so we want to unnest it to avoid having to do Olympic-level gymnastics to do any analysis.

Most functions designed to read a JSON file into R do not handle deeply nested JSON files well. Thus, a streaming function is required to deal with multiple lines of JSON per file. One solution that is manageable in this case due to relatively small data size, but is not generally a good idea, is to manually copy and paste all of the data into a single JSON file using a text editor, resulting in a 111MB text file with a single JSON object per line. Following that, the *jsonlite* package⁷ was able to stream in the single unified data file with

⁵<https://www.json.org/json-en.html>

⁶https://en.wikipedia.org/wiki/Apache_Parquet

⁷<https://cran.r-project.org/web/packages/jsonlite/index.html>

the correct number of JSON lines, which was 37082. When iteratively reading the separated files, it tended to fail at various points due to line breaks and trailing garbage.

Alternatively, Bob Rudis, the Chief Security Data Scientist at Rapid7, and his collaborators released the *ndjson* package on CRAN⁸, which is intended specifically to deal with JSON files that have more than one record per file, such as occurs in streaming data of this type. The *ndjson* package addresses the need to flatten newline delimited JSON data structures so they may be worked with in a rectangular format. In comparison to the other packages, it simply works by recursively checking for another nested level of data until it runs out of levels. Using the *ndjson stream_in()* function allowed reading in of all of the Alerts data files quickly and efficiently.

3.1.3 Network Alerts Data - Unnesting Comparison

The two options provide parallel pictures of how unnesting the data will look. Using the *ndjson stream_in()* function resulted in a data.frame with 37,082 observations and 11,981 features or columns, which is 3.3GB to work with in-memory. Using the *jsonlite::stream_in()* function resulted in a data.frame with 37,082 observations and 5 features, which is 349MB to work with in-memory, or one-tenth the size at this stage. One of these features (the “Source” list) may be expanded, resulting in 81 features instead. However, 13 of those features are still the deeply nested lists of varying sizes and data structures.

When unnested by JSON line, which corresponds at this stage to an alert ID, not necessarily unique, a typical one of these rows contains thousands of sparsely populated features. When streaming and unnesting using the *ndjson* package, 97.4% of the table is empty. This is because any particular Alert entry might have an extremely variable number of machines, users, IP addresses, ports, or connections associated with the alert that may be collected by the IDS. For example, on September 7, 2019, the first alert ID had approximately 3,150 Destination IP addresses associated with a single Command&Control Behaviour alert. This is common in something like a denial of service attack, which may have thousands of incoming IP addresses from a botnet attempting to overwhelm a site or server with more traffic than it is equipped to handle. In comparison, when using the *jsonlite::stream_in()* function, if we extract only the features that can expand easily due to fairly shallow nesting (approximately 68 features), only 36% of the table is empty. It is the expansion of the remaining 13 features into thousands more features that results in the extreme sparsity. Indeed, I found that many of these features were not useful for the purposes of this thesis.

I introduce here several comparisons among the deepest nesting layers to ensure the *ndjson* package unnests the same elements as may be extracted manually. The R code may be found online⁹ or can be

⁸<https://cran.r-project.org/web/packages/ndjson/index.html>

⁹<https://github.com/ben-burr/thesis-code/>

provided upon request.

1. **Source: Data: Rank.Alert: Structured Message:** When manually extracted from the *jsonlite* data frame, this feature is empty in this particular dataset and in the *ndjson* data frame.
2. **Source: Data: Rank.Alert: Justification:** When manually extracted from the *jsonlite* data frame, this feature is empty in this particular dataset and in the *ndjson* data frame.
3. **Source: Source: Reputation:** When manually extracted from the *jsonlite* data frame, this feature contains 8,115 key-value pairs of Sources and Reputations (e.g. "alien-vault":"Malicious Host"). This exactly corresponds to 8,115 key-value pairs in two columns with matching names in the *ndjson* version.
4. **Source: CorrelationIds:** When manually extracted from the *jsonlite* data frame, this feature has up to 8,735 entries for a given alert. Extracted on its own from *jsonlite* package, the first row has 7,091 correlation IDs, which corresponds exactly to the first column of the *ndjson* version. The *jsonlite* version ought to be transposed, as the `data.table` unnesting operation results in the alerts being spread left to right and the correlation IDs for each alert acting as the rows. When unnested using the *jsonlite* package, the columns are sorted in the wrong order (e.g. 1, 10, 100, ... 2, 20,...). The *ndjson* package does not place all of the CorrelationID columns together either, as the unnesting occurs in stages and columns spread out across the entire dataframe. Direct comparison may be carried out by extracting the related columns from each and sorting the *jsonlite* columns by column number. Once the columns are named identically and in the same order, 10 columns that were randomly sampled and compared were found to be identical.
5. **Source: Data: Rank.Alert: Interesting:** Alongside CorrelationID, this portion of the data proves to be the most challenging to compare. Each row in this column was a dataframe of a variable number of columns. Each of those columns was a list or a dataframe itself. Some of those dataframes were key-value pairs. This makes intuitive sense, but is difficult to work with. For example, row 32,161 has 203 IP addresses associated with a single alert. Each IP address is identified as being an IP address, the IP addresses are listed, and the details of the connection are included. This includes latitude, longitude, country of origin, city of origin, region, reputation, and so on. A direct comparison of any row in the two versions of the dataset confirms that any value in one can be found in a corresponding entry in the other.

These comparisons confirm that the *jsonlite* version of the data matches the *ndjson* version. We can now be confident that the automation carried out by the *ndjson* package is trustworthy and will rely on this version

of the data. However, the Network Events data for the concurrent time period was provided in Apache parquet format, so a different approach is required to format and analyze it.

3.1.4 Network Events Data

Network Event logs were provided from the beginning of September 2019 to the end of October 2019. Each day's data was provided in its own folder. Within each of these folders, the hourly data was also separated into separate folders by hour. The hourly data is divided into smaller parquet files, with anything from a single file to around 50 files for a given hour. In total, 19,708 files were provided over the 61 days of the relevant period. One file for September 21, 2019 in hour 10 was an empty temporary file rather than a parquet file.

Each parquet file is small, so this amounts to only 2.94GB of storage space on disk. However, in-memory, this expands considerably, since parquet is a highly compressed format. For example, for a file that is roughly 1MB on disk we obtain an 8.1MB data frame in R, with 35 columns and 32,204 rows. This is not prohibitively large, but decompression makes the given data slightly difficult to deal with in-memory. The 2.94GB of parquet data expanded to 13.5GB in 8 csv files following reformatting and export in R. Code may be found online at <https://github.com/ben-burr/thesis-code/>. The 35 features found in these datasets are listed in Appendix A.2.

3.2 Public Datasets

In order to carry out a reasonable evaluation of the success of clustering approaches used on the Rank dataset, I considered five possible publicly available datasets. There are several criteria for a comparable dataset. Since the Rank data was generated from raw network logs and exported using their network monitoring and IDS architecture, I first needed to ensure that the public dataset was also either generated from network logs or had a similar set of features available. I also wanted the network data to be aimed at intrusion detection research or anomaly detection research at a minimum. If possible the flows needed to be bi-directional, but this did not play into the analysis so it was not essential. The Rank network infrastructure has around 50 machines, including servers, mostly running Windows. A similarly-sized network with a comparable infrastructure would be helpful but is not required. The Rank data covered two months of time, so ideally the public data set would cover at least a few days to a few months. Something modern would also be preferred.

It is difficult to fulfil all of these criteria given the lack of truly comprehensive, modern public datasets available for this application, but several recent datasets provided some possibilities. Five datasets were

considered, and ultimately the CICIDS2017¹⁰ dataset from the University of New Brunswick (UNB) was chosen. It contains several attack scenarios and covers 5 days between July 3, 2017 and July 7, 2017. It has an extensive feature set, which includes the source and destination IP addresses. The flows are bi-directional. It covers a shorter timeframe and has fewer machines in the network environment than would be ideal, but is otherwise promising. The CICIDS2017 dataset is the easiest to work with in terms of data format, and is a good candidate based on characteristics and content.

3.3 CICIDS2017 Dataset Description and Pre-processing

Sharafaldin and colleagues[79] described the methodology for generating this dataset in their related paper. The data capture began at 9:00am on Monday, July 3, 2017 and ended at 5:00pm on Friday, July 7, 2017. The first day contains only benign network traffic, while attacks were generated for either the morning or evening period of the subsequent days. Some periods experienced multiple attacks. Each of the 14 attack types is described briefly in their paper and covered the most common attacks as of 2016. The dataset was generated using a network topology that includes modem, firewall, switch, router, and all three of the major operating systems. A user profiling agent was used to generate 10 PCs, 3 servers, a firewall, and 2 switches in the victim network and 4 PCs, a router, and a switch in the attacker network. All available protocols, which include HTTP, HTTPS, FTP, SSH, and email protocols, were a part of the dataset.

A total of 84 features were extracted from the packet captures using the CICFlow-Meter. Each flow was augmented with identifying information of Source and Destination IPs and Ports. Each flow was labelled as benign or as a specific attack. The authors also proposed the four best features that could be used to detect each type of attack. Two different datasets in csv format were provided, one with the identifying features as columns and one without. PCAP files were also available for researchers to extract their own features.

Certain characteristics of the dataset are problematic. There are 288602 instances of empty rows in the overall dataset. For some reason, these rows have only entries of 0 in the Forward Header Length column, and otherwise are NAs throughout. As such, nearly any variable but that one may be used to eliminate them by using a negative filter.

In addition, certain features had negative values that should not exist. For example, Flow Duration has negative values, but a negative duration is impossible. Similarly, the flow inter arrival time maximum is a negative value for some observations, but there can be no negative time. These values likely correspond to some sort of defined indicator. However, it is unclear what they might mean. A list of duration or count related variables with negative-valued observations follows:

¹⁰Hosted at <https://www.unb.ca/cic/datasets/ids-2017.html>

1. Flow Duration
2. Flow IAT Max
3. Flow IAT Mean
4. Flow IAT Min
5. Fwd IAT Min
6. Fwd Header Length
7. Bwd Header Length
8. Init Win Bytes Forward
9. Init Win Bytes Backward
10. Min Seg Size Forward

Simple frequencies on the labels associated with each of these show us that the Flow Duration and Flow IAT Maximum and Flow IAT Mean are all associated with 115 benign cases. This suggests it is likely safe to set these equal to zero rather than the negative values. This may miss some nuance of the data definition that is unknown to us, but seems appropriate. The Forward and Backward Header lengths and the Minimum Segment Size Forward are associated with 35, 22, and 35 benign cases respectively. I felt that these could also be set equal to zero safely.

The remaining features have attack cases associated with their negative-valued observations. The Flow IAT Minimum has 2,891 negative-valued cases total, of which 6.7% are attack cases. The negative values of Forward IAT Minimum are associated with 17 DoS attack cases. The Initial Win Bytes Forward feature's negative values have a very small proportion associated with PortScans with only 7 of the 1,001,189 cases being attacks (0.0007%). The Initial Win Bytes Backward feature has around half (1,441,552 of 2,830,743 or 51%) of its values below zero. These are associated with 10 types of attack, with 8.5% of the negative-valued observations being attack observations.

There are also several constant columns, which do not provide additional information for clustering or anomaly detection. These are as follow:

1. Bwd Psh Flags
2. Bwd Urg Flags
3. Fwd Avg Bytes Bulk

4. Fwd Avg Packets Bulk
5. Fwd Avg Bulk Rate
6. Bwd Avg Bytes Bulk
7. Bwd Avg Packets Bulk
8. Bwd Avg Bulk Rate

There is the option of removing the negative values, setting them equal to zero, or leaving them alone. Similarly, it is likely safe to remove the constant columns, so I elected to do so. The distribution of class labels for the negative-valued cases versus the non-negative-valued cases is different enough that it may not be acceptable to correct these values to zero safely, and it seemed questionable to remove them entirely. As such, I left them alone and trusted that standardization techniques later would deal with them appropriately.

Two other columns present minor issues. Two columns have infinite values. They could be set equal to the maximum of the same feature, the maximum possible value, or a slightly higher value than the feature's maximum. These observations could also be removed or set them equal to zero, but it seems questionable to set them equal to zero. Infinite values might imply some kind of lost connection or a connection timing out, or they may correspond to a certain type of traffic behaviour. If enough of these values are present, analysis of differences between the largest non-infinite values and the infinite values might help determine whether the extreme values might also correspond to connections timing out. In the current analysis, the infinite values have been set equal to my machine's maximum, but I acknowledge that this is likely not the ideal option. For example, if these values correspond to dropped connections, it would be preferable to replace these values with an indicator function.

In the Flow Packets/s feature, these infinite values correspond to 2,867 cases. Of these, 1,777 (62%) are benign. Most of the rest correspond to DoS Hulk (33%) attacks and PortScan (4.4%) attacks, with a small handful being Bot (0.3%), DDoS (0.07%), and FTP-Patator (0.1%) attacks. There are no missing values in this column, but enough of them correspond to attacks that they cannot simply be removed. The maximum non-infinite value on my machine is $1.797693e+308$, so that is what was used. In the Flow Bytes/s feature, there are 1,358 missing observations and some infinite values. In fact, we find that there are 1,368 benign traffic cases where the Flow Bytes/s is infinite, and we find that there are 949 of the 1,358 cases (69.8%) where Flow Bytes/s has values of NA and the label is DoS Hulk. This suggests that missing Flow Bytes/s values here may be associated with loss of connection that could be related to an attack. On the other hand, infinite values are more likely to be benign (90.6%). Furthermore, all 31 cases where Flow Bytes/s is the maximum value of $2.071e+09$ are benign. However, by setting the infinite values and the missing values equal to the

machine maximum, we find that we now have an exact correspondence in counts between the two variables where infinite values were present. This seems acceptable, but an expert on network administration could point out some flaw in this approach.

3.4 CICIDS2017 Feature Selection

Anomaly detection methods network data often rely on packet header fields, which include flags, port numbers, and IP addresses[88]. The packet contents are often encrypted, so deep packet inspection, which would often reveal anomalous patterns more readily, is not feasible in most circumstances[51]. However, if there is nothing anomalous about the packet header fields, other information may be needed to detect an attack. This might include statistics on packet arrival times, or packet sizes or frequencies. This is where hybrid IDSs could help, as they can work at the host level to examine the packet contents for malicious code embedded in the payload. However, I do not have access to host level data, so I focus on only those features that may be accessible from the network log data.

Despite its relative youth, the CICIDS2017 dataset has been examined extensively since being released. I consider which features work well for clustering this data with the intention of identifying outliers or anomalies that might indicate attacks. Despite the fact that the dataset is labelled I do so using unsupervised techniques. As mentioned previously, it is unlikely that labelled data will be available in the majority of cybersecurity application settings, so relying on results from supervised approaches is fairly meaningless. That said, I still consider the features selected in such applications, as they may prove meaningful in this application in more general terms.

Sharafaldin and colleagues[79] at the CIC and UNB generated this dataset to make up for previous deficiencies in publicly available intrusion detection datasets, including traffic diversity, a variety of modern attacks, and a strong feature set and metadata. They note that Source and Destination IPs and Ports were included to identify the flows. In the real world, these might not only identify the flows, but also provide useful characteristics of benign or anomalous traffic by identifying particular ports or devices that have been targetted or are already compromised. They used the Random Forest Regressor algorithm to obtain the importance of each feature, then the average standardized mean value of each feature was multiplied by each class to obtain the feature's importance value by class. The benign traffic was best identified by the backward packet length minimum, subflow forward bytes, total length of forward packets, and the forward packet length mean. The backward packet length standard deviation was useful for identifying DoS GoldenEye, Heartbleed, DoS Hulk, and DDoS attacks. The flow duration was useful for identifying Heartbleed, DoS Hulk, DoS Slowhttp, DoS Slowloris, Infiltration, and DDoS attacks. Interarrival Time minimum, mean, and

standard deviation were each useful for identifying several of the attacks. In general, the maximum and total statistics for any particular flow measurement were not considered useful. However, there was enough variation in the most useful features that most could likely not be eliminated without perhaps damaging our ability to capture particular attack behaviours. Precision, recall, and F-measure were used to examine the results of various classification algorithms, with random forest being the most accurate and one of the fastest. Quadratic Discriminant Analysis was four times faster and only slightly less accurate, while ID3 was four times slower but just as accurate.

A paper by Abdulhammed and colleagues[1] examined the use of autoencoders and PCA for dimension reduction, then used the reduced feature spaces to run various classifiers. They obtained a high degree of accuracy and good visual separation between malicious events and benign ones with a combination of principal components and a random forest classifier alongside uniform distribution based balancing to deal with class imbalance. Their autoencoder's two layers reduced the number of features to 64, which exactly matches the number of features remaining in the data after removing binary flag features, the timestamp, and the ID feature, suggesting that autoencoders may work better with continuous features. Accuracy did not improve between 10 and 70 principal components using random forest, and that linear and quadratic discriminant analysis grew slightly less effective overall as the dimensionality was reduced. With the autoencoder, the accuracy was slightly improved by reducing from 81 to 59 features for random forest and Bayesian network approaches, and slightly reduced when using LDA and QDA.

Another analysis by Abdulraheem and Ibraheem[2] looks at shortcomings in the dataset. These would include null values, suitable features for detecting attacks, highly correlated and redundant features, and balance. They considered the Source and Destination IP addresses and the Source Port to be improper features to use in model training because they are not constant, while [1] disagreed and converted the IP addresses to integers to use them. They disagreed with a previous study[63] that merged attack classes, as they found that these classes could be discriminated. Previous works had also used either minimax or z-score scaling for data normalization, whereas [2] used a quantile scaling. They found several features had no values and removed them. Then they used random forest to evaluate the relative feature importance, removing 14 that had an importance of less than 0.001. These included the previously-mentioned binary flag features along with several statistical features. They then used a correlation threshold of 0.95 to remove another 20 redundant features, mostly packet statistics that had a strong relationship between forward and backward size or length. This left them with 36 features. On the remaining features, the three normalization methods were compared using a multi-layer perceptron model with three hidden layers. A comparison was also made to the 23 features chosen by the original authors of the dataset[79]. They found a small increase in accuracy and decrease in loss in using the 36 features, primarily with specific types of attack: Bot, DoS Slowhttptest, FTP-Patator,

SSH-Patator, and Brute Force web attacks.

The dataset was used by Ahmim et al.[4] to test their hierarchical model composed of three classifiers. Each level of the model used different features, with the third using all available features. Features with Flow Packets/s with values of infinity or NAN and features with constant values were removed beforehand. Standardization was done using the minimax method. Accuracy was good on benign traffic and most types of attack. However, DoS GoldenEye, Bot, and Web Attacks were poorly detected by most of the approaches they tried, suggesting either that those attacks are harder to detect or their feature set or models were not tuned well enough.

Aksu et al.[5] used Fisher Score feature selection, again with supervised learning algorithms. They proceeded from 80 features down to 10 by reducing 10 at a time and calculating accuracy over 100 iterations. They found that 30 features worked best for KNN and decision tree, with the extra 50 features reducing KNN's accuracy by 5% by adding noise and not affecting decision tree's accuracy. Their SVM approach had an accuracy rate roughly in line with a coin flip throughout, suggesting it may not be an appropriate discriminator on this particular feature set. The ten most effective features using Fisher Scoring were:

1. Fwd Packet Length Mean
2. Avg Fwd Segment Size
3. Fwd Packet Length Max
4. Bwd IAT Min
5. Total Length of Fwd Packets
6. Subflow Fwd Bytes
7. Fwd Packet Length Std
8. Destination Port
9. Protocol
10. Fwd PSH Flags

This is interesting because one of the flags is present, and the destination port and the protocol were not considered by most other approaches so far.

Patil, Pachghare, and Kshirsagar[66] used the CICIDS2017 dataset to detect DDoS LOIC attacks, selecting the 18 most relevant features using performance-based feature selection using JRip and 10-fold CV.

This method ranks the features on each iteration by accuracy and time to build the model, then comparatively updates the overall feature set ranking. The selected features were the following:

1. Source Port
2. Fwd Packet Length Max
3. Fwd Packet Length Min
4. Flow Packets/s
5. Fwd IAT Std
6. Bwd IAT Std
7. Bwd Packets/s
8. Packet Length Std
9. Packet Length Variance
10. FIN Flag Count
11. SYN Flag Count
12. RST Flag Count
13. PSH Flag Count
14. URG Flag Count
15. Init Win bytes backward
16. Act Data Pkt Fwd
17. External IP
18. Label

This is clearly problematic, since they included the label that indicates an attack as one of their most prominent features for detecting an attack, which must be a mistake. Furthermore, it appears they used the CSV files rather than the PCAP files, yet they list feature 86, External IP. This is not a feature in the CSV dataset, which only has 85 features. Still, the features listed may be relevant in detecting DDoS LOIC attacks and perhaps could be useful to consider as part of the feature selection process for this dataset. The Init Win bytes backward feature was also in the top 10-20 features in almost all attack types.

Prasad, Tripathi, and Dahal[68] use Bayes' theorem and rough sets to probabilistically select the highest ranked features. They used minimax standardization on the data and removal of imbalance in the data, then the Feature Probability Estimation algorithm to compute the probability of importance for each feature. The important features varied by day, which is reasonable because attacks varied by morning/afternoon and particular day. Based on the top 20 features for each attack type, one of the inter arrival time (IAT) statistics was an important feature in most attack types. The protocol was another important feature. The header length in at least one direction was important. The number of bytes or packets in either direction was important. The destination port was important. The overall packet statistics, which include the packet length overall and in either direction and either the maximum, the minimum, the mean, or the standard deviation, were all considered important for different types of attacks. The amount of time a connection was active, which included either the maximum, the minimum, the idle maximum or minimum, or the mean of active or idle time, was also important. From this list, we should be able to determine roughly 10-15 features that are likely the most important overall, as several that varied depending on attack type may be approximately equivalent to others. It should be noted, again, that their analysis failed to account for columns of all zeros, some of which were included in the top 20 important features for particular attack types.

Mathur et al. attempted to use supervised methods to detect botnets by mining network traffic from the CTU-13 and ISOT datasets, using flow start and end times and duration, source and destination IP addresses and ports, protocol, some type of flag, input packets, and input bytes. These were selected using the Cfs-SubsetEval filter method to choose the features with the highest predictive power, regardless of algorithm or domain. Logistic regression and multiclass classifiers were more accurate than their other approaches.

Overall, no public dataset will perfectly match the Rank dataset. However, based on the criteria outlined above and the findings from previous studies, the CICIDS2017 dataset is an appropriate choice for the goals of this thesis. In the next section, I use this dataset first to explore dimension reduction methods to locate a smaller subspace that allows separation of the malicious and benign observations as much as possible, then to test density-based clustering methods to isolate the malicious observations in their own clusters.

4 Methods

One goal of this thesis is to find a way to cluster network traffic data so that, if possible, malicious traffic is separated from benign traffic. The methods used in this thesis are all considered, either alone or in combination, with that goal in mind. In broad terms, these methods seek to reduce the size of the feature space in order to simplify clustering and analysis of the data by focusing on a few of the most important features while still providing a good description of the data. In particular, I want to find a way to make any outliers or anomalies in the data appear more pronounced or easier to detect. To accomplish this, there are several possible tools available.

Rather than simply selecting one or more variables that is hoped to be most relevant to this problem based on domain knowledge but which may not have the best characteristics to genuinely help, I could instead define certain elements to have a greater importance by weighting them. One of the most common ways to do this is Principal Component Analysis (PCA). Another is Independent Component Analysis (ICA). A more recent and modern approach is the use of neural networks, by way of an autoencoder, to perform non-linear PCA. An additional method to be considered is word embedding, which will be used to embed categorical features such as IP addresses, ports, and protocols, into continuous numerical vectors upon which traditional distance measures may be used. These vectors will be used together with the rest of the data when working with the above methods.

Clustering also generates a summary of the data in terms of certain characteristics, such as the density of a group of observations or the distance between density centres. Following dimension reduction, density-based clustering is used to attempt to locate useful groups of data. Ideally, the clustering reveals separability in the data that may be used to extract malicious traffic from the otherwise benign network flows. Each possible method aims to transform or project the data into a new space, or, equivalently to rotate and transform the space around the data to provide a more useful representation of the data. It is hoped that a combination of these approaches will yield meaningful clustering results.

4.1 PCA for Dimension Reduction

Traditional PCA transforms a set of variables through weighted linear combinations into a new set of variables that are orthogonal to each other and therefore are uncorrelated. This technique allows us to explain the variance-covariance structure of a set of variables[34]. All of the principal components together explain all of the variation in the original data. The amount of variation explained by each component is listed in descending order from greatest to least, so the first principal component captures the most variation in the data.

If a principal component representation of the data in a few dimensions that explains most of the variation can be obtained then it may be possible to greatly reduce how many dimensions must be considered and proceed with analysis on this subset of the components. The reduced set of components may then be used as input to cluster analysis or to provide better insight into a problem. Principal Component Analysis is a useful descriptive tool, but care must be taken if using it for inference, as inference relies on the assumption of the data having a multivariate normal distribution, which often does not hold in practice[35].

Below I discuss the method of carrying out Principal Components Analysis for the classical approach using Singular Value Decomposition (SVD). Robust PCA is also considered. Both PCA and RPCA are tested using classical standardization and robust standardization of the dataset. The pre-processing methods used for the dataset are described. In addition, possible methods of evaluating the principal components to determine a suitable subspace are considered.

4.1.1 Singular Value Decomposition (SVD)

Suppose we have a data matrix X with n rows of observations and p columns of variables. Principal Component Analysis relies on a method of matrix factorization called singular value decomposition (SVD). Consider first the spectral decomposition of a square, diagonalizable matrix by decomposing it as $X = PDP^{-1}$. The spectral factorization for nonzero, non-symmetric, possibly non-square rectangular matrices is the SVD. We formulate the SVD as $X = P\Sigma Q^T$. The matrix P has orthonormal columns, so $P^T P = I$. The matrix Σ is a diagonal matrix with the singular values of X as the diagonal entries. The matrix Q^T has orthonormal rows, so $Q^T Q = I$. In contrast to the spectral decomposition PDP^{-1} , the P and Q matrices need not be related. Assuming that X has distinct singular values, the SVD is unique aside from possibly changing the signs of the columns of the P and Q matrices. Proof of these properties may be found in most standard linear algebra textbooks. I have referred to Olver and Shakiban[61] because it is my favourite.

The transformations given by the matrix multiplication $P\Sigma Q^T$ represent rotations in R^n and R^p for P and Q^T , and scaling for Σ , transforming the data into a new space. This transformation maximizes the amount of variation in the data in each orthogonal direction iteratively. The component with the maximum amount of explained variation in the data is found in the first basis dimension and the component with the least amount of explained variation is found in the last basis dimension, with the total number of components equal to the original number of variables in the data. The new basis dimensions correspond to eigenvalue-eigenvector pairs where the maximum variation corresponds to the first eigenvalue in $\Sigma_{1,1}$ and the values decrease by ranked variation down the diagonal. The variance of each component corresponds to the eigenvalue, so $\Sigma_{ii} = \lambda_i$. By transforming the data into principal components, the covariance matrix is diagonalized so that

the variances of the components are found along the diagonal and the correlations between variables are zero in these directions, or $cov(y_i, y_j) = E(y_i y_j) - E(y_i)E(y_j) = 0$ for all $i \neq j$.

Given a singular value decomposition of a data matrix with rank r , and $1 \leq k \leq r$, we may select an upper left matrix with k columns from the matrix Σ that contains the first k eigenvalues along its diagonal. We may also select the first k columns of P and Q^T . In doing so, it turns out that $B = A_k = P_k \Sigma_k Q_k^T$ is the rank k matrix for which the Euclidean matrix norm $\|A - B\|$ is minimized, so B is a lower rank approximation of A that captures the essential features of a dataset in data compression[61].

The SVD method described above allows us to obtain the principal components of a matrix efficiently. However, since data may be measured on different scales, standardization may be needed before carrying out SVD. If the data is on the same scale, the covariance matrix may be used in the decomposition. When the scales vary, standardization leads to the correlation matrix being used instead since the standardized covariance matrix becomes the correlation matrix; however, there is no straightforward relationship between the principal components obtained from one and those obtained from the other. Furthermore, it is more difficult to interpret the components obtained from the correlation matrix because they refer to standardized variables, and it is more difficult to base inferences on them[35].

With our network traffic data, the scales include the number of forward and backward packets, flow durations, times between packet arrivals (inter arrival time), means, maxima, minima, and several binary flags. For simplicity, it is here assumed that there are no missing values. In some cases, a handful of columns fully populated by NA and NaN values have been removed. The data is first centered and scaled so that each variable has a mean of zero and a variance of one. Normalization prevents the differing scales of the original data from interfering with analysis, since one or more of the unscaled variables could end up dominating the variation, and hence the first few principal components, if we were to ignore the normalization step. After normalization the same variable or variables might prove to be responsible for a considerably smaller, possibly relatively unimportant, proportion of the variation.

4.1.2 Selection of an Appropriate Subspace in Classical PCA

Having carried out PCA, I wanted to select a smaller rank k subset of components that would allow me to explain enough of the variation in the data that I might distinguish useful or interesting characteristics. The goal is to select a subset of components that will allow us to separate clusters of benign and malicious traffic. Everitt and Hothorn[18], Jolliffe[35], and Rencher and Christensen[71] describe a number of methods, both formal and informal, that may be used to make this decision based on the literature on PCA. Many of the formal methods devolve to heuristic rules when placed in a real-world context.

1. Set a percentage of variance that must be accounted for. Keep the number of components that achieves this. Everitt and Hothorn suggest 70 – 90% is a common choice; Rencher and Christensen mention 80% as a possibility.
2. Calculate the mean of the eigenvalues (equivalent to the mean variance of the original variables) and exclude those that are less than the mean. This is also known as the Kaiser Rule[37]. If the correlation matrix is being used, the average variance will be 1, so any eigenvalue less than 1 would be excluded. An alternative suggested Jolliffe and Everitt is to exclude eigenvalues less than 0.7 instead, which serves to account for sampling variation. This alternative also applies when using the correlation matrix. The idea of excluding eigenvalues that explain less than the mean is that there is no gain from employing a linear combination that explains less than including the average original variable would explain.
3. Generate a scree plot of the eigenvalues against the number of components. An “elbow” bend or visible change in the slope of the plot beyond which the plot is roughly a straight line suggests a potential cut-off k . This is because the amount of variance being explained levels off, so adding more components does not provide as much additional information. A modification of this approach is to use the log-eigenvalue diagram, which plots $\log(\lambda_i)$ against i instead[18].
4. The Broken Stick model uses a randomization procedure to decide on the number of PCs retained. Random points are selected between 0 and 1 based on the specified number of dimensions. The average length is obtained of the longest segment, the second-longest segment, the third-longest segment, and so on. The probability that segments will satisfy the triangle inequality sets a threshold for the size of the eigenvalues. For example, the probability that the first longest segment will satisfy the triangle inequality is around 0.25. Once the expected value from the Broken Stick model overtakes the observed eigenvalue, the cut-off is made[83][33]. The Broken Stick model guides selection of the number of PCs to retain based on the expected value of the k^{th} -longest piece of a unit-length stick that has been broken into a total number of pieces equal to the number of variables[33]. A cut-off is selected at the last component before the component where the variation explained comes within some local neighbourhood of the expected variation explained by the Broken Stick model for the number of exponents in the data.
5. Jolliffe discusses that in some circumstances the last few components may be the most important ones for an analysis rather than the first few; however, it does not appear as though the noise is of particular interest in the applications related to this thesis.
6. Bartlett’s test may be used with $H_{0,k} : \lambda_{k+1} = \lambda_{k+2} = \dots = \lambda_p$ against the alternative hypothesis at least two of the last $(p - k)$ eigenvalues are unequal. The idea is that applying this test backward sequen-

tially for values of k can allow us to determine how many principal components contribute substantial amounts of variation and how many are simply noise[35]. This approach may be problematic, as the distributional assumptions of carrying out such a test may not necessarily be fulfilled, the significance levels are usually unknown, it tends to fail in cases where variables have little difference in variance, and it tends to retain more PCs than are necessary. For example, Jolliffe found that using this as a cutoff rule resulted in a cut-off of about 0.1 or 0.2 rather than 0.7 in the correlation matrix case.

7. Jolliffe[35] and Forkman and colleagues[20] describe several computationally intensive cross-validation methods that test the quality of the approximation given by B_k . These include subdivision of the data matrix, resampling methods, bootstrapping the residuals or the data, using a jackknife estimate or an approximation of one, Bayesian procedures, parallel analysis, and statistical tests that assume non-standardized variables. These methods are beyond the scope of this thesis, but would be worthwhile considerations in an extension.
8. Jolliffe describes using the average of the squared partial correlations, conditional on the use of the first k PCs.
9. The statistical significance of the components to be retained after carrying out PCA may be assessed using parametric bootstrap tests proposed by Forkman and colleagues[20]. This method is intended to deal with data that has first been standardized, and does not rely on asymptotic properties. A non-parametric version has not been developed at present, but would be a useful addition. The parametric version was also intended for small samples and tends to vastly overestimate significance on large samples.

Such methods of data reduction rely on the assumption that choosing a few of the PCs that describe a large proportion of the variation in the data will be a meaningful approach to data reduction. An alternative interpretation offered by Jolliffe is that if a certain number of PCs account for most of the variation in the original data the basis dimensions of those PCs may describe a more realistic dimensionality for the original data, allowing us to eliminate some of the original variables without much loss of information. The underlying assumption is that the excluded dimensions are primarily noise.

Since typically an intrusion detection system will require a simple and widely-applicable approach, and furthermore relies on timeliness and computational efficiency, we will ignore some of the more time-intensive and computationally-intensive methods described. Additionally, it must be remembered that intrusion detection is not a static problem, even if generally treated as though it is in this thesis. The data will change, either gradually or dramatically, over the course of a given day, week, or month. The types of attacks and

their characteristics are also constantly evolving. The variables that are important may therefore change over time, so automatic or built-in elimination of particular variables or continual selection of the same number of PCs would be ill-advised. The changing characteristics of the data, the changing behaviour of users, and the changing threat landscape implies that selection must be dynamic. Using the smallest index above a threshold, as in the average eigenvalue threshold, is one way of allowing such adaptability.

4.1.3 Global Dataset Processing for Classical PCA

With the global CICIDS2017 dataset (84 variables, all 5 days of data) I use only the numeric data columns since standard PCA does not allow for categorical data. This causes problems with the ID column, the IP addresses, and the date column. A method to try to include the information provided by some of this information, particularly the IP addresses and ports, is presented in Section 4.5.1. Ten binary “flag” variables are removed as well. The blank rows of the dataset were removed. Several of the flow duration and count columns had negative values. It is unclear what these values might mean in this context, but a negative duration or count is not possible in the real world, so those entries could have been set equal to zero. In the interests of not making changes to the data without reason, these values have been retained as they are, but should be noted as potential data quality issues.

In addition, the Flow Packets Per Second and the Flow Bytes Per Second columns have infinite values in them. These have been reduced to match the largest non-infinite value in each column, $4e+06$ and $2.071e+09$ respectively. The constant columns were subsequently removed as well. The time variables, which included a datetime feature that had been deconstructed into a YYYY-MM-DD feature, a two-digit hour feature ranging from 00 to 23, and a timestamp feature. The hours actually included in this dataset do not range from midnight to midnight, but instead only include some of the typical business hours or roughly ten hours per day.

Two standardization choices were considered. In the first, the data columns were centered and scaled to have a mean of 0 and a variance of 1. Then PCA was run on the entire dataset using the *prcomp* function from the base *stats* package in R[69]. In the second standardization approach, the data columns were centered using the median, and scaled using the Median Absolute Deviation. The same classical PCA method was used again on the resulting data. The Forward Header Length variable tended to be unstable during standardization, which is likely a floating-point or rounding error due to some of the standardized variables having 15 or more decimal places. As such, at times this variable was removed to allow algorithms to work. It was retained whenever possible.

4.2 Robust PCA for Skewness

Classical PCA is intended to be applied to multidimensional elliptical, symmetric data, which is why the data is transformed prior to performing PCA. Classical PCA is, however, sensitive to outliers. Because PCA relies on decomposition of the variance-covariance matrix, which is sensitive to anomalies, the first several components tend to be pulled in the direction of any outliers, which tend to be the directions of greatest variation. Hubert, Rousseeuw, and Verdonck proposed a method in 2009 to handle asymmetric data without the transformation of the original data[30]. The modified algorithm for skewed data presented here was based on the *ROBPCA* algorithm developed by Hubert, Rousseeuw, and Vanden Branden in 2005[29].

Taking a dataset X , with p variables and n observations, the principal components are defined as $t_i = X * p_i = X * \operatorname{argmax}_a(\operatorname{Var}(Xa))$, with constraints $\|a\| = 1$ and a is orthogonal to each p_i . The algorithm proceeds in the following steps, described in more detail in the 2009 paper[30].

1. Perform SVD (not strictly necessary).
2. Choose a value for α between 0.5 and 1. A smaller α is more robust but less accurate.
3. Calculate the adjusted outlyingness of each row of the data. Then the mean and covariance matrices are calculated for the set of data points with the smallest outlyingness, defined using the IQR of the projected data and the medcouple as a robust measure of skew. Fewer points are flagged as outliers due to the skewed distribution[28] than with classical PCA or standard Robust PCA. In the skew-adjusted version, the adjusted outlyingness is calculated using the adjusted boxplot based on [28].
4. Project the data onto the k -dimensional subspace spanned by the first k eigenvectors of the robust covariance estimator. The value of k is chosen by screeplot or a robust PRESS algorithm[28].
5. In the skew-adjusted version, the orthogonal distance of each point from the projection is calculated to give a robust estimate of the subspace spanned by the first k eigenvectors of the covariance matrix of observations for which the orthogonal distance is less than a cutoff defined empirically. The orthogonal distances to a power, $OD^{2/3}$, are approximately normally distributed, so the cutoff c is determined by $c = (\hat{\mu} + \hat{\sigma}z_{0.975})^{2/3}$.
6. Compute a robust center and covariance matrix in the k -dimensional subspace with the lowest adjusted outlyingness. The reweighted robust covariance estimator is applied to the data projected onto the k -dimensional subspace.

This algorithm identifies and distinguishes between three types of outliers. Good leverage points fall near the principal component subspace while remaining outlying to regular observations. Orthogonal outliers

do not project well onto the subspace because they lie above the point cloud and are projected onto the principal component axes. This means they may be projected into the point cloud and not appear outlying after rotation. Bad leverage outliers are both orthogonally distant from the PC subspace and are farther away from the regular data when projected onto the subspace[30]. In the application of classical PCA above, the primary problem appears have been inadequacy of the procedure to handle orthogonal outliers, leading to the attack traffic being projected onto the point cloud so that it overlaps or mixes with the benign traffic. The Robust PCA method allows the use of projection pursuit to attempt to find an interesting and useful subspace upon which to project the data.

The 2009 version of the algorithm follows the same basic steps as the 2005 algorithm, but with three changes. The first change is that in the third step, the adjusted outlyingness measure based on the adjusted boxplot for skewed distributions from [28] is used. The adjusted outlyingness flags fewer outliers in a skewed distribution, using the median and first and third quartiles of the projected data as robust measures of skew. The second change is that the cutoff value for orthogonal distances in Step 5 is set to be the largest orthogonal distance smaller than the absolute orthogonal distance based on the empirical data rather than on theoretical distributions as in the original algorithm[30]. The third change uses the lowest adjusted outlyingness in Step 6 when computing a robust center and covariance matrix.

The number of components to retain, k , may be selected using Hubert's criterion, may be specified by the analyst, or may be set to the maximum, as desired. Hubert's criterion[29][30] uses an initial covariance matrix for projection, then robustly estimates the location and scatter, and selects k non-zero eigenvalues as the robust PCs. In this thesis the maximum has been used. The number of directions used in projection pursuit was 1000 thanks to increased computing power available. More directions may yield more components, but 1000 may have performed slightly better compared to lower numbers like 250 or 500 directions. The Robust PCA procedure with the 58 non-binary, non-constant variables was tested using mean centering and standard deviation scaling. The *rospca* package in R[72] has an implementation of the *ROBPCA* algorithm from Hubert's 2005 paper, along with the extension for skewed data[30] and a sparse version that was not needed here. Classical standardization was carried out using subtraction of the column means and division by the standard deviations. Robust scaling used the median to center the data and the median absolute deviation to scale the data.

4.3 Autoencoders

Autoencoders are a form of artificial neural network, but the input layer and output layer are the same. On one side the data is compressed, encoded, or transformed using an activation function into code vectors that

result in a lower dimensional representation in the middle. Then the data is reconstructed or decoded on the other side. The original idea is that if one were sending a large piece of data to someone else and wanted to do so while minimizing the size of the data being transferred, this encoding-decoding scheme should allow compression and reconstruction of the original data while minimizing the error between the original data and its reconstruction. It is assumed that the data is not linearly separable, and based on preliminary exploration, the CICIDS2017 data is not easily linearly separable. The hope in the context of this thesis is that making a nonlinear transformation to the data will allow linear separability afterward so that a complex decision boundary in the original space becomes a simpler one in the transformed space. If that is the case, a neural network is not really needed, though may still be used. The number of hidden layers is generally left at one, because performance increases from adding additional layers, i.e. deep learning, are somewhat context-specific; however, Goodfellow, Bengio, and Courville suggest that greater depth results in better generalization [24]. If we have one hidden layer, the neural network may approximate any function that contains a continuous mapping from one finite space to another[25]. Two hidden layers allow representation of arbitrary decision boundaries to arbitrary accuracy with rational activation functions, and can approximate any mapping to any accuracy.

In an unsupervised context, the input and output layers have the same size. An autoencoder with lower-dimensional hidden layers in the middle is called *undercomplete*. The number of nodes in such a layer should be set up in such a way that the weightings cannot simply capture the data and must learn intelligent features to reconstruct the data on the other side. The code representation summarizes the input by way of a latent space representation. The success of the reconstruction is gauged by a loss function's measurement of the error rate. In an autoencoder, the loss function is either the MSE or the binary crossentropy. In trying to classify the CICIDS2017 data, the MSE might be useful if classifying all classes of malicious traffic, or the binary crossentropy might be useful if attempting a more realistic benign-malicious dichotomy.

In a simple autoencoder with one hidden layer, as is visually represented in Figure 1, the input layer takes the original data and passes it through an encoder function onto the first - possibly only - hidden layer. Then the output of the final hidden layer is passed through a decoder function to reconstruct it in the output layer. This process can be described using basic function composition of the form $g(f(X))$, where f is the function between the input and the hidden layer and g is the function between the hidden layer and the output layer. The neural network is usually trained using gradient descent on a loss function that describes the information lost during reconstruction. The loss function has the general form $L(X, \hat{X}) = |X - g(f(X))|^2$.

Principal Component Analysis may be thought of as a linear autoencoder[15] with a single hidden layer that has k nodes and whose weights span the same subspace as the first k principal components. The average squared error between the data X and its reconstruction $\hat{X}$ is minimized, or, in other words,

Figure 1: Typical representation of an autoencoder with a single hidden layer. This amateur version based on an image at <https://www.jeremyjordan.me/autoencoders/>, used with permission.

$\mathcal{L}(X, \hat{X}) = \frac{1}{N} \sum_{n=1}^N \|X_n - \hat{X}_n\|^2$ is minimized. Equivalence with classical PCA occurs when linear activation functions are used to carry out the encoding-decoding steps, but one of the strengths of autoencoders is that they do not need to fit a hyperplane like classical PCA does. Instead, a nonlinear, continuous, non-intersecting surface may be used, so autoencoders may be thought of as a generalized version of PCA.

The hidden layer is one of the most important aspects of the autoencoder for this thesis, along with the choice of activation function. If the hidden layer is a lower-dimensional space than the input and output layers, the autoencoder is called *undercomplete*. The central hidden layer may be extracted and used in a similar way to selecting the first few principal components as a lower-dimensional subspace containing the data. By projecting the data onto this subspace, the autoencoder is forced to learn a representation of the data that captures what is hoped to be the most important latent characteristics for reconstruction. Once the data is encoded, the move from the hidden layer to the output layer usually suffers some amount of loss

of information. This is a key aspect of using autoencoders for anomaly detection. It is assumed that the neural network will learn and reconstruct normal data best, so abnormal data will typically result in greater reconstruction error. Thus, the reconstruction error can be used as an anomaly score.

Making sure the trained model does not overfit the data and allows enough generalization to be useful must be balanced with ensuring the model can accurately reconstruct the data. The loss function is minimized to ensure a close approximation of the original data is obtained during reconstruction. To prevent overfitting, the hidden layer can be used to ensure the hidden layer has few enough nodes to prevent overfitting, introducing an information bottleneck; however, using a lower dimension in the hidden layer does not guarantee prevention of overfitting. The quality of the output of the autoencoder does not depend on what happens in the hidden layers[6], so the representation of the data may not be particularly meaningful. This is best illustrated by the pathological case of using a single node in the hidden layer. In that case, the network would likely learn an index mapping that captures none of the latent relationships between data points but could accurately map the input back to the output.

To counteract this problem, reconstruction loss cannot be the only constraint used. Data manifolds can be used to characterize a topological surface on which the data tends to be concentrated. On such a surface, some directions are necessary to explain the variability in the data. Data that is located farther away from that surface is less probable, so it is desirable that the network learns the directions that explain the shape of this surface and considers the characteristics that are farther away from the surface to be more anomalous.

A regularization parameter or penalty may be used in addition to the loss function to prevent overfitting and improve generalization. This can be done by penalizing the activation function so only some of the hidden nodes are used or “fire” at each step, forcing the neural network to reweight in a way that captures important features in the data. Another method is a probabilistic approach using a variational autoencoder, which encodes statistics from a prior probability distribution, samples from them in the hidden layer, and decodes into a posterior distribution, using the Kullback-Leibler divergence to measure the distance between the two distributions[6]. Adding noise and having the network learn the manifold that removes the noise, called a denoising autoencoder, is another option that forces the decoding function to resist small variations in the input data. Contractive autoencoders resist small variations in the encoding function by contracting local neighbourhoods through constraints on the derivative; the two are essentially the same thing from two different directions.

4.3.1 Choice of Nodes in the Hidden Layer

As suggested above, a single hidden layer is the simplest case. Deep or stacked autoencoders use more than one hidden layer between encoding and decoding. Additional layers allow representation of more complicated nonlinear relationships in the data[7]. On the other hand, Reed suggests that the network must have sufficiently many nodes, and that “sufficient” is more of a moving target than a well-defined number[70]. Goodfellow, Bengio, and Courville imply that better generalizability results from deeper networks[6]. Hidden layers are typically added symmetrically, so if three hidden layers are used, the two outer hidden layers will have an equal number of nodes to each other and the middle layer is often the smallest number of nodes. To set the number of nodes in a hidden layer, there is no overarching theoretically grounded rule, as was also the case in PCA and ICA. A few suggested rules with no theoretical backing are to set the size of the hidden layer no greater than half the size of the outer layers, or to use a number of neurons equal to the square root of the product of the number of nodes in the input and output layers[50]. This second suggestion does not apply to the type of autoencoder discussed here, which has an equal number of nodes in each of the two layers. The result is simply a hidden layer of the same size. However, since an autoencoder in this context is similar to PCA, we might consider similar ideas.

A grid search can be carried out on a range of options to determine a good choice for the number of nodes in the hidden layer and the number of layers. An automated search strategy could try random configurations of layers and nodes. Alternatively, some type of systematic approach could be used, or a directed search might be useful. An exhaustive search tries all possible combinations. The possibilities are extensive, so a exhaustive grid search may be infeasible and is not recommended on data with a large number of features, as it is only feasible in small networks and datasets. However, a range of values for single layer and multiple layer combinations could be selected and a grid search run on these to calculate which of them returns the smallest mean square error. I carried out a grid search using a single hidden layer and a number of nodes ranging across the entire number of dimensions. In addition, the grid search includes three hidden layers and five hidden layers, with symmetric exterior layers, as discussed above, and including various reasonable combinations of values based on the dimension reduction work done using PCA and ICA above. This is discussed in more detail in Section 5 and Section 6.

It must be remembered that using too few nodes will result in underfitting while using too many will result in overfitting. The danger of overfitting for this thesis is that the malicious observations will also be fit and the analysis approach will not generalize well to unknown attacks. Too many nodes will also mean that some of the nodes will not be useful in the weighting. One method of addressing this is pruning, in which nodes that do not noticeably affect performance are removed. These may be determined by looking at the

post-training weight matrix and finding weights very close to zero to remove. A pruning algorithm may often be used to accomplish this pruning by beginning with too many nodes and allowing the excess to be chopped as it proceeds. This should not be an issue with large data.

Since the autoencoder learns the Day-of-Week or Date feature very well, one preliminary step to take is to again subset the data based on day of occurrence prior to proceeding. As such, additional steps from this point forward should be considered to have been carried out on daily rather than global data. Additional granularity exists in the data that is not here explored, but an hourly breakdown is also possible. It is not as immediately clear that this would be useful, but other exploratory work not included here suggests that it may provide meaningful feedback in the synthetic data. Whether this translates to the real-world remains to be seen. It would be beneficial to explore time series approaches at this stage, but this is, as usual, beyond the scope of this thesis.

Following subsetting, the grid search can be used to inform the number of nodes selected for the autoencoder's hidden layer. The reconstruction error for each of the grid search possibilities may be compared to determine the optimal model. A manual pruning method may also be used.

4.4 Independent Component Analysis (ICA)

Independent Component Analysis is similar to PCA, but ICA decomposes data into statistically independent components rather than components that are only uncorrelated as in PCA. A standard example of ICA is the cocktail party problem. Imagine several conversations are occurring simultaneously in a room alongside shuffling feet, clinking glasses, music being played, and the ambient noises of the room and the areas beyond. Imagine these sounds are recorded by multiple microphones. Each microphone most strongly picks up the source sounds closest to it, but background noise interferes with the clarity of the signal. In blind source separation, which is closely related to ICA, the goal is to separate such mixtures into sources from each part of the room. Mathematically speaking, the goal of ICA is to find a linear and statistically independent data representation for de-mixed signals that are primarily non-Gaussian. The number of sources may not be equivalent to the number of variables available, so some method of dimension reduction on the ICA data representation can also be considered, assuming that the number of sources does not exceed the number of variables available. It is also possible to perform nonlinear ICA, but that is beyond the scope of this thesis. Since the CICIDS2017 data appears to be primarily non-Gaussian, ICA may be appropriate in that it could allow separation of the signals of the attack data from the benign traffic. It should be noted that it would be proper here to test the Gaussianity of the data prior to proceeding, but that this has been passed over in this analysis.

4.4.1 ICA: A Little More Linear Algebra

The ICA model, as described by Hyvärinen[31], is presented as follows. Suppose there is a matrix S of source signals and a set of n random variables describing linear combinations of those source signals. Represent these sources as $X = AS + V$, where each row represents a mixture as a weighted sum of the source signals. The V matrix represents random noise. The matrix A is such that the entries are real numbers that act as weighting coefficients for the mixture. Given this setup, the goal is to extract the matrix S , representing the independent source signals, or to estimate that matrix accurately.

This poses a challenge because the mixing matrix A and the sources S are both unknown, and the noise matrix V cannot easily be determined. It follows that the first step is to find a way to estimate the mixing components and the source components. Palmieri, Fiore, and Castiglione assume that network traffic is recorded in a noise-free environment[62], an assumption that Hyvärinen suggests works well in many situations. Hence, the model is first simplified to $X = AS$ for this application and the noise is ignored. The process of unmixing is an attempt to project the mixture matrix X onto the space of S so that the original signals may be recovered as much as possible. Approximation of the source signals is defined as $\hat{S} = WX$. Here W is a matrix of weights, which acts as a pseudoinverse of A .

It is assumed that the signals are statistically independent, so $p(s_1, s_2, \dots, s_n) = p_1(s_1)p_2(s_2)\dots p_n(s_n)$, and that the signals mix linearly, though this assumption is not strictly required since nonlinear methods may also be used. It is also not required that A be a square, invertible matrix. When it is not, the number of mixtures can be different from the number of components. It is usually required, however, that at most one of the source signals is Gaussian, although Woods, Hansen, and Strother[90] proposed a solution that allows separation of Gaussian signals using PCA after the fact. If two or more variables are jointly Gaussian, then they will not be separable by ICA and will result in arbitrary linear combinations of these components[31]. This is because uncorrelated and jointly Gaussian variables are independent, so unmixing results in an identical distribution and ICA fails to separate the signals.

The sum of two independent random variables is more Gaussian than the original variables[31], but the Gaussianity can be minimized when the sum is roughly equal to one of the source signals, implying that they have nearly identical distributions. At the same time, Gaussian variables have the largest entropy of all variables with equal variance[31]. Thus, if we have a set of variables with unit variance, Gaussian variables will have the largest entropy. This occurs by design, then, when we standardize the data, and the size of the entropy can be used to decide which variables are the most Gaussian.

We then want to maximize the non-Gaussianity, which is measured using negentropy. Computationally, negentropy is approximated using sample skew and kurtosis. Kurtosis, a special case of negentropy, is con-

ventionally defined as $kurt(x) = Ex^4 - 3(Ex^2)^2$, but when x is standardized this simplifies to $kurt(x) = Ex^4 - 3$. Hence kurtosis is a normalized version of Ex^4 , the fourth moment. A Gaussian random variable has a fourth moment of $3(Ex^2)^2$. Since $Ex^2 = 1$, this gives us a fourth moment of zero. Normalized negentropy is defined as $J(x) = H(x_{gauss}) - H(x)$, where $H(x) = -\int p_x(\eta) \log p_x(\eta) d\eta$ is the differential entropy of a random vector x with density $p_x(\eta)$ and x_{gauss} is a Gaussian random vector that has the same correlation matrix as x .

When optimized, there are two local optima for each of the independent components, since the sources may be estimated only up to a positive or negative scalar of the source, as in PCA. Geometrically, this is an optimization problem on the unit hyper-sphere with its optima occurring exactly when one of the independent components occurs and corresponding to projections that maximize the absolute value of the kurtosis.

4.4.2 Choice of ICA Algorithms and Parameters

The data is centered and sphered, as in PCA, since it simplifies the algorithm[31]. Then non-Gaussianity is maximized using either a fixed point method or gradient descent method. Mutual information measures how much information one variable has on another variable in a set of variables, so it can be used to measure dependence between variables. Minimization of mutual information is equivalent to maximization of entropy or of the sum of non-Gaussianity in the data and results in maximization of the independence between components. This approach does not assume anything about the data[31], so it can be useful even when an ICA model is unrealistic.

Define w to be a matrix of weights. Define z to be the whitened and centered data. Define G to be a generalized nonquadratic index function, and g as a nonlinear derivative function of G such as a hyperbolic tangent function, a scaled exponential function, or a cubic function. Define v to be a standardized Gaussian random variable. Define $\gamma = E\{G(w^T Z)\} - E\{G(v)\}$.

Hyvärinen recommends using symmetric decorrelation rather than estimating the components sequentially. He also recommends using robust, nonpolynomial nonlinear algorithms for g , the simplest of which is the hyperbolic tangent function; however, a second function may be needed when using gradient algorithms. He also notes that statistical properties will depend on the choice of nonquadratic objective function G . A general-purpose choice is the *logcosh* function; for highly super-Gaussian components or when robustness is important, *-exp* may be better; kurtosis may be a good choice for sub-Gaussian components with no outliers[31]. Random variables with negative kurtosis are sub-Gaussian (more uniform) and with positive kurtosis are super-Gaussian (with sharper peaks). The CICIDS2017 data is highly super-Gaussian as well as skewed, and based on the extreme values may require a robust approach, so the objective function used will be $-\exp(-\frac{x^2}{2})$.

The algorithm for the *fastICA* algorithm from the *fastICA* package maximizes the negentropy approximation described above using the fixed-point method. It gives the choice of $-\exp(\frac{-u^2}{2})$ and either a deflation or a parallel estimation procedure. The algorithm proceeds as follows.

Result: Maximization of Non-Gaussianity

1. Center the data to mean zero;
 2. Whiten the data by projecting it onto its principal component directions with the number of component directions specified;
 3. Estimate the weight matrix W that maximizes the negentropy approximation so that $S = XKW$ and W is orthonormal;
-

4.4.3 Choosing the Number of Independent Components to Estimate

Principal Component Analysis returns the same number of components as there are original dimensions in the data, but some of those components may explain little of the variance and can often be safely discarded as noise. Some ICA approaches[45][59] utilize PCA for dimension reduction, with the number of estimated ICs equal to the number of PCs chosen at the outset, or perhaps a combination of the PCs and ICs[16]. Independent components are considered by some to be more interpretable than PCs because they are independent sources rather than merely uncorrelated. However, different initial approximations can result in different components because the solutions to the optimization problem may locate a local rather than a global optimum and the resulting components are not ordered[37] as in PCA. Without a way of determining the sources represent something meaningful, the interpretation is still difficult to make. It is even more difficult to determine how many independent components to use, which remains an open problem, so trial and error is one of the accepted approaches[31]. The number of components selected or separated could be equal to the original number of dimensions in the data or could be fewer.

Kairov et al.[37] describe the problem as a two-part one of dimension reduction first and feature extraction second. Because the CICIDS2017 dataset includes labels, a reasonable guess at the number of sources is easier to make and can be used to explore potential numbers of components to extract. However, it is unlikely this number will be meaningful outside of the specific dataset. In real world situations related to network threats, it is unlikely the number or type of attacks afflicting a network at a given time will ever be fully known, so a subjective choice of effective data dimension could easily be incorrect. If the effective dimension is chosen to be too small, components will be fused together, reflecting lost sources. If the effective dimension is chosen to be too large, ICA will over-decompose and reflect more sources than are present[37]. Aside from the methods of determining the effective data dimension that descend from PCA, Kairov and colleagues describe theoretical approaches that utilize Minimal Description Length or AIC, use data structure

organization, or employ cross-validation. A Bayesian formulation of ICA can employ the BIC[37]. In their paper, they examine component stability over multiple runs of ICA on the same dataset measured by varying the algorithm initialization and evaluating internal compactness of clusters of matched ICs across multiple runs. They also consider component stability across varying choice of effective dimension by measuring correlation between matched components from components from decompositions of the different orders of effective dimension. Reproducibility across independent datasets measured by average correlation between matched components using the same parameters but different datasets is also considered.

These approaches appear to be more useful in high-dimensional problems when sparsity can be used to help distinguish between the effective data dimension and additional dimensions that have limited utility for source separation, but the overall ideas may help identify effective dimension for ICA. The stability approach utilizes the inflection point between sets of stability values, where the stability values of stable components are approximately uniformly distributed and the stability values of unstable components are not. The conservation approach is the most intuitive, as it is essentially trial and error with careful comparison. Comparing ICs across independent datasets is infeasible here. The general ideas presented there resemble those proposed by Himberg and Hyvärinen[27], who attempted to address the problem using a method they called Icasso much earlier. Using PCA for potential dimension reduction first, either a bootstrap, random initializations, or both together are used to repeatedly run ICA on the same data. The resulting components are then correlated to determine their nearness to each other. Similar components are clustered together using any available clustering algorithm (they used agglomerative hierarchical clustering). The tighter together a cluster of similar components is, the more reliable that component is considered. Components that tend to generate wider variations appear less likely to be meaningful or useful and may be considered for discarding. This method seems to be one of the best available.

Liu and colleagues[47] believed that the most reliable known method of component selection is based on manual inspection of ICs based on component loading, distribution of the mixing matrix, visualization, and clustering techniques. They concede, however, that it is impractical to analyze all of these each time and that the best components do not necessarily give the best subspace since components might work well with other components but not alone. For classification, some of the ICs that appear less important end up being useful. All components are equal candidates as a result.

Wang and colleagues prove that ranking the columns of the mixing matrix by MSE from greatest to least and using traditional PCA dimension-reduction methods can be tenable[87]. The same general idea of the sum of the explained variation and the process errors is used to evaluate the results. Hendrikse, Veldhuis, and Spreuwiers[26] similarly derive an ordering method for the components. The data power is defined by the variance contributions of the components, with each making an independent contribution similar to ranking

eigenvalues in SVD. This method turns out to be the same as SVD in some mixtures, so it provides a good description of the data in terms of least squares. However, they suggest an alternative of performing ICA on dimensional-clusters. Since dimension reduction and source separation occur on all dimensions at once, this alternative focuses on performing ICA on subsets of the dimensions individually. This can be done by dividing the data into equally-sized blocks and performing ICA on each block. Then the strongest sources in each block are selected and the others are discarded. This begins at the highest granularity, at the discretion of the analyst. At the next step, the neighbouring blocks are merged into larger blocks and the process is repeated over and over until only one large block remains. This reduces overtraining in local sources, and performs better than global PCA or ICA in identifying the strongest sources. It also avoids the problem of discarding small principal components as is done in PCA, which may eliminate the ability to distinguish weak sources using ICA so that they can no longer easily be identified, recovered, or separated. The same problem is addressed differently by Risk, Matteson, and Ruppert[75], who order components by marginal likelihood rather than by variance. A 2017 dissertation by Nethery[58] at UNC Chapel Hill proposes a bootstrapping method to retain correlation structures in EEG data, allowing computation of uncertainty and associated hypothesis testing of the parameter estimates in ICA. Both of these methods seem promising but are beyond the scope of this thesis and thus are not explored further.

There were three viable options for selecting reliable components for this thesis. The first is to use PCA to reduce dimensionality, then use ICA at the same dimensionality. The second is to use column ranking by MSE on the mixing matrix, as per Wang and colleagues[87], then to use the PCA dimension-reduction methods such as the broken stick model on the ICs. The third option is the Icasto algorithm. The method was implemented in Python (available on Github¹¹).

4.4.4 Icasto

The Icasto method for ICA relies on running multiple iterations of ICA on the same data with random initialization and possible bootstrapping at the beginning of each run. The absolute value of the mutual correlation coefficients is used to derive similarities between components in different iterations and a dissimilarity matrix is constructed. The matched components are clustered using any desired clustering algorithm, though agglomerative hierarchical clustering with average linkage was used by the authors. The results define cluster centroids. The compactness of the clusters is examined, with compactness defined as the case when similar component estimates repeatedly occur despite randomized initialization. A tighter cluster implies a stable component. If there are mostly tighter clusters in general this implies there are mostly stable components overall. A quality index can be derived using the difference between the average intra-cluster similarities and

¹¹https://github.com/Teekuningas/icasso/blob/master/icasso/_icasso.py

the average inter-cluster similarities. Stable estimates are selected as the final component estimates. An average of the estimated components can be chosen as a set of reliable components generated by multiple runs and use that as the final component. Such a choice means that the final ICs are not strictly independent but should be more reliable than using estimates from a single run. These centroids may be used on the original data to determine groups of observations that are not captured by the components, which may indicate outlying points or may indicate an incorrect choice of number of components that would lead to revision before proceeding.

In the original paper introducing this concept, each estimated component gives a single point in space, and each run provides a new estimate. The more runs of the algorithm that produce a close point, the stronger the assumed reliability of the component. Unreliable components correspond to points that do not belong to a cluster. It is assumed that in practice the demixing matrices will be estimated, so the demixing matrices are combined into a single matrix $\hat{W} = [\hat{W}_1^T \dots \hat{W}_M^T]$ for M runs of ICA with n_i ICs for each run.

The similarity measure is defined as the absolute value of mutual correlation coefficients between components, r_{ij} , where $i, j = 1, \dots, K$. These are obtained by taking the covariance matrix Σ for the data matrix X and calculating $R = \hat{W}\Sigma\hat{W}^T$. The assumption must then be that the number of estimated components matches the dimension of the weight matrix to avoid nonconformity issues. Hence, either the X matrix must be reduced beforehand so the supplied Σ matrix represents the reduced dimension or the full complement of components must be estimated. The elements of R give the similarities $\sigma_{ij} = |r_{ij}|$. The similarities are transformed into dissimilarities $d_{ij} = 1 - \sigma_{ij}$.

The set of all estimated components is partitioned into a specified number of disjoint clusters. Any clustering algorithm may be used on the dissimilarity matrix to achieve this. Hierarchical clustering is considered a good choice because it allows easy evaluation of what represents a reasonable number of clusters at different levels of the dendrogram. Hence, the specified number of clusters may be selected based on cutting the dendrogram at a level that allows one to obtain that number of clusters.

A cluster quality index may also be used to determine a natural number of clusters, but this is subjective, as is the selection of a cutoff level of the dendrogram in this scenario. Group average linkage was used in the original implementation of Icasso because it was less sensitive to noise than single linkage. Average linkage also joined clusters more consistently than single linkage and complete linkage when the number of clusters chosen was less than the number of components overall. The index I_q is given by the difference between intracluster and intercluster similarity. It measures the compactness and isolation of a cluster and is defined as

$$I_q(C_m) = \frac{1}{|C_m|^2} \sum_{i,j \in C_m} \sigma_{ij} - \frac{1}{|C_m||C_{-m}|} \sum_{i \in C_m} \sum_{j \in C_{-m}} \sigma_{ij}$$

Here, $C_{-m} = C - C_m$. An ideal cluster returns an I_q of 1 and results in lower values for less compact or less isolated clusters. The Icasto authors recommend investigating values of I_q in increasing rank order with I_q plotted on the x-axis and the cluster number on the y-axis.

4.5 Word Embeddings for Categorical Traffic Identifiers

Until now I have ignored the presence of IP addresses, ports, and protocols in the dataset. These features are ubiquitous in network traffic data, but they are difficult to deal with. Each is categorical, but not easily used alongside mostly continuous data of the sort we have considered so far. IP addresses have a vast number of possible values. Local IP addresses may not have particular meanings due to automated assignment by the network, so it may be difficult to quantify distance or relationships among them. External IP addresses can originate anywhere in the world, and can easily be masked to prevent us from locating them. There are tens of thousands of possible ports, with some more commonly used while broad ranges of them lying unassigned in a local network.

It would be useful to be able to analyze the relationships between the host IP addresses, the ports they use, and the protocols present in network traffic logs and to maintain the context in which such traffic occurs so that two IP addresses or ports that exhibit similar behaviour and relationships in the network will appear to be near each other while those that act differently from each other will appear farther away from each other. To do this requires the introduction of a transformation from categorical values to some form of numerical one that has a distance measure built into it.

In section 1.1 the type of metadata associated with such traffic was discussed. In the context of the CIDS2017 dataset and the Rank dataset, it can be known roughly which host machines are close to each other in terms of the typical behaviour expected and the type of user that would generally use the machine. A network administrator will typically know which machines are outward-facing or may communicate with external devices and features, are associated with certain ports or protocols, or have particular security measures implemented. These may have specific IP addresses associated with them, whereas perhaps internal workstation IP addresses are automatically assigned an address from a pool of available IP addresses in a specified range on demand. This makes it difficult to build firm rules, but gives a general sense of how particular machines should act, so the behaviour of a server can be distinguished from the behaviour of a workstation; however, if simply examining generic network traffic logs from an outside perspective, no such intuition is available, posing additional challenges to this work.

Furthermore, IP addresses, described in Section 1.1, have no encoded relationship to each other. For example, the address 192.168.0.1 may not be closely related to the address 192.168.0.2, despite the apparent

nearness of the bitstrings. In fact, either address may be available but not in use in a particular network. The goal is to encode a meaningful distance measure that will account for the metadata and network infrastructure context so that when two distinct IP addresses that exist in the logs are compared, those that exhibit similar behaviour and context to each other will be near each other, while those that exhibit different behaviour and context from each other will be farther apart so that embeddings properly reflect the real-life topology of a given host network. The shortcoming of this approach is that it is not widely generalizable to other networks.

4.5.1 Word2Vec

This section relies heavily upon a paper that discusses three overarching methods of handling IP addresses and other categorical network features in network flow data[73]. The first method is ignore the IP addresses in any distance calculation. The second method is to extract some meaningful features from the IP addresses. For example, the IP address may be converted into its binary form and up to 32 or 128 features can be extracted from it. The source IP addresses may also be used to aggregate network flows, possibly within a time window, and perform further analysis to detect anomalies. The third method is to define a metric on the IP addresses, such as the distance between 32-bit integers, the geographical distances between the places from which they originate, or the hierarchical distance in the network topology. The second of these methods takes some behaviour into consideration during aggregation, but otherwise these approaches do not account for relationships among the categorical variables. Instead, it seems reasonable to consider embedding the categorical variables within a useful space using generated relationships between flows in such a way that a meaningful distance measure is defined.

The most basic approach to embedding words into vectors is called one-hot encoding. Encoding in this manner represents each unique word by a dummy variable, generating an extremely sparse high-dimensional space. This approach rests on the assumption that there are no relationships between words or categories being embedded. Using the example of IP addresses, the “word” 192.168.0.1 is equally different from 192.168.0.2 and 255.255.255.255. It is easy to see why such a method could be problematic because there are over four billion (or 2^{32}) IPv4 addresses and 3.4×10^{38} IPv6 addresses, making the space prohibitively large. To counteract this, each IP address could be converted to binary 32-bit or 128-bit form or an integer value prior to embedding. Alternatively, the common representation as a 12-digit number connects parts of the IP address to particular aspects of the network topology, such as the subnet, which allows them to be related to expected behaviours in that part of the network and from that type of machine; however, as was mentioned above, no true context is attached to the addresses in this scenario, so any distance measure is essentially meaningless. The Word2Vec algorithm was introduced by a team at Google led by Mikolov in around 2013[54]. The

patent[53] describes the general procedure of training data, comprising sequences of words, on a classifier and an embedding function using various parameters and generating a high-dimensional distributed numerical representations of each word in the training vocabulary as output. Such numeric representations can reflect either semantic or syntactic similarities between words, depending on the model used. Learning algorithms in NLP are thought to perform better with this type of distributed representation.

The Word2Vec procedure uses a fully-connected neural network and begins with a one-hot representation of the vocabulary. The hidden layer is much smaller than the vocabulary. The Google team used 300 neurons in the hidden layer, which may have been related to the number of features in the data, since often one neuron per feature will be used in the hidden layer. The output layer has one neuron per vocabulary word, with values in the range of 0, 1, representing probabilities that a word appears in a context, using softmax for output classification[73]. The representations encode patterns in the words in such a way that formulations like “Madrid” + “Spain” = “capital” might be a valid derived meaning, while “Madrid” - “Spain” + “France” results in nearness in meaning to “Paris.” Word representations can also include phrases, which helps generate context for words that mean one thing on their own and another when combined with another word or group of words.

The Skip-gram model used in Word2Vec takes in a sequence of words $w_1, \dots, w_T$ and maximizes the average log probability

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log(p(w_{t+j}|w_t))$$

with the probability representing the likelihood of a word appearing in a particular context within a specified training window of size c . The probability utilizes the hierarchical softmax because it is more computationally efficient. The training window allows construction of the distance metric. For a given word w_i , distance of other words w_{j_k} away from the w_i within the window and frequency of occurrence of each w_{j_k} together with w_i build the importance of the relationship between any two words in the vocabulary. Words that occur frequently enough do not improve the context much, so Mikolov and colleagues introduced up-sampling of rare words, which is intended to rebalance the training process. They also introduced negative sampling, which speeds up the training of the neural network by only updating the weights of the correct output and a random sample of other neurons for each training sample[73].

4.5.2 IP2Vec

A related formulation of the Word2Vec approach was introduced in the 2017 IEEE conference on Data Mining by Ring, Landes, Dallmann, and Hotho[73]. Since IP addresses appear in most network flow data it would be useful to be able to use them to enrich contextual information around each network flow. However, as

mentioned, there is no easy distance measure that can be used to help build relationships among flows. It was similar to previous work¹² online that had generated similarities between network flows that included IP addresses. Ring and colleagues' method, called IP2Vec, builds similarities between IP addresses (rather than network flows) based on context from traffic flows. Hereafter, IP2Vec is used to refer to this work and its authors.

Since IP addresses, ports, and protocols are all categorical features, IP2Vec attempts to embed them in such a way that they are turned into continuous vector representations based on the context information representing behaviour that is included in flow data. The IP addresses, ports, and protocols are used as the vocabulary input to the Word2Vec algorithm. The individual network flows comprising this information make up the text corpus. A fully-connected neural network with one much smaller hidden layer is used. The sizes of the input and output layers are identical. A softmax classifier is used for the output layer probabilities that a particular word will appear in a context. The final weight matrix between the input and hidden layer is extracted to be used as the features of the IP addresses, since these weights have been iteratively updated based on the data and should represent a measure of similarity between words. Each flow describes a single network connection, with some unknown aggregation possibly occurring behind the scenes. Each flow becomes a sentence for the purposes of Word2Vec input. IP2Vec generates similarities between hosts, hence choosing features to use that would aid in this goal. Source port was ignored, since it is fairly meaningless in most contexts. It should be noted, as pointed out in the IP2Vec work, that word similarities in IP2Vec may change over time because network connections are dynamic and new ones may be made at any time. As in their work, this thesis considers only an offline scenario where they remain constant.

IP2Vec lists the typical components of a network flow as including source and destination IP addresses and ports, protocol, bytes, packets, TCP-flags, timestamp, and duration. Source IP address, destination port, and protocol were used as input words. When Source IP address was used as input, destination IP address, destination port, and protocol were used as context words. When destination port was the input word, only destination IP address was used as context. They used unidirectional flows, but in both the CICIDS2017 data and the Rank data the flows are bidirectional. Because of this, fewer restrictions were placed on the input and context words in our work[10] than in the original work. This creates duplicate training samples, as indicated in IP2Vec, but we considered this to be acceptable because of the directional differences.

4.5.3 Evaluating IP2Vec Embeddings

The IP2Vec authors used T-SNE to visualize and qualitatively evaluate clustering output from a similarity measure. In our follow-up work[10] we used the embeddings to generate similarity scores that then informed

¹²<https://edhenry.github.io/2016/12/21/Netflow-flow2vec/>

the edges in a graph-based model. Although this generated a useful visual representation, in addition to T-SNE on the embeddings, that allowed us to see some connections between malicious flows and host machines, the graph-based approach will be ignored in this thesis. Instead, the embeddings will be appended to the data and used as part of the input to the dimension-reduction and clustering algorithms used in the remainder of this thesis. The hope is that the embeddings will allow location of better subspace representations of the network flow data for clustering of the malicious flows.

One problem with the approach in this thesis is that the Monday data is free of attack in the CICIDS2017 dataset, but since the approach is unsupervised this ought to be treated as unknown. There is little guarantee that captured network packets in the real world will be entirely free of malicious activity. However, one possible approach would be to use the Monday data to generate the embeddings that are later used in the remainder of the work on the CICIDS2017 dataset. The problem with doing so is that we have no knowledge of whether undetected attacks occurred on the Rank dataset during the timeframe captured, so there is no guaranteed clean data that can be used to generate the embeddings. The Alerts data can be used to indicate possible malicious activity, but the network environment is not safely controlled like the synthetic data's environment is, and could have been attacked by something outside of the purview of the defense system that went undetected.

In IP2Vec, a set distance between categories of IP address was defined, with categories including private, public, multicast, link local, and default network. The distance between addresses in a category was given by the Hamming distance on the binary representation of the IP addresses. In [10], we used cosine distance on the embeddings and ignored categories, relying instead on expert knowledge of the particular network topology of the network being used to consider particular hosts and their relative distances. Work by Schnabel et al.[76] suggests that typical cosine similarity is not necessarily a useful evaluation method because of variability across embedding and evaluation methods. They propose evaluation by using individual queries, as done in our work, although we still used cosine similarity in this individual evaluation.

4.5.4 Word2Vec Parameter Selection

Aside from those addressed immediately below, the default hyperparameters were selected for Word2Vec. This thesis used a negative sampling skip-gram Word2Vec model with Python 3.7 and the Gensim 3.8.1¹³ implementation of Word2Vec. Negative sampling used 5 words. The minimum vocabulary count was 1 word. The maximum window size was 3 words on either side of the target word. Ten epochs were enough for convergence. Various embedding sizes were tested, but 32-dimensional embeddings (or 32 nodes in the hidden layer) were settled upon. Higher dimensional embeddings tended to decrease the resulting similarity

¹³<https://radimrehurek.com/gensim/>

measures between hosts that were expected to be near to each other, whereas lower dimensional embeddings tended to increase the same measures. With 32 dimensions, the cosine similarities of closely-related hosts appeared to fall roughly within the 60-70% cosine similarity range, whereas 10 dimensions resulted in many embedded words approaching 100% similarity, which introduced difficulty in distinguishing between them. While 32 dimensional embeddings are likely not ideal, the choice was reasonable based on IPv4 addresses. The ports and protocols may be better served by fewer features in the embeddings.

4.6 Density-Based Clustering

The goal of clustering data is to assign similar observations to the same cluster and differing observations to different clusters[57]. However, it is generally unknown how many groups of observations there are and to which group each observation belongs. Cluster analysis aims to use data to develop rules with which to assign these observations to a group. In the case of network security, it is desirable that characteristics can be found to distinguish malicious activities from benign ones in such a way that malicious activities are not assigned to common clusters with benign activities. Ideally, malicious activities will form their own standalone clusters that may be isolated from the rest of the traffic, allowing the security analyst to eliminate the threat and plan defenses against future ones. However, evaluation of the quality of the clusters is difficult in unsupervised data since there is no objective measure of how successful the clustering has been or whether the cluster assignment rules are correct. The Rand index, for example, gives the fraction of correct clustering decisions, but it is impossible to know the correct clustering on unlabelled data.

Density-based clustering has the following general definition from Silverman[81]: Take a set of points X_i in a dataset X in a p -dimensional space. The peaks or modes in a density estimate $\hat{f}$ constructed from these points will correspond roughly to a cluster. Generally, density-based methods use the number of points in a volume or area of the space or a density estimate based on a kernel smoother[3]. Relationships among the points can be constructed using a hierarchy, where a distance measure $d_{i,j}$ between two points X_i and X_j is defined. Among the points within a threshold distance t_i (or an ϵ -neighbourhood), the one that is the farthest “uphill” from a point is named its parent. This ensures that the centroids of a cluster can be found at its highest point or peak, which corresponds to the densest point. Hence, the function $\frac{\hat{f}(X_j) - \hat{f}(X_i)}{d_{i,j}}$ is maximized for j for all points within t_i distance from X_i and for which the density estimate of $\hat{f}(X_j)$ is larger than the density estimate of $\hat{f}(X_i)$. If there are no points with a larger density within the threshold distance of a point, it is a root node of a cluster, or, in other words, a local maximum. The clusters are generally divided by the valleys, low points, or local minima, in the density.

Silverman notes that there are multivariate consequences that had not, at the time, been addressed. He

also refers to older work by Cox and others that multiple “bumps” in a probability density likely indicates mixing of components[81], where a bump is defined as f being concave over an interval $[a, b]$ but not over a larger interval. Clearly if a cluster represents a probability density and there are multiple clusters there must be a mixture of densities. The standard starting point to deal with such mixtures is the use of mixture models for clustering. It can be difficult to determine the generating distribution of real-world data, though. Ideally, a distribution-free density method that can handle many dimensions and large samples could be used to locate dense areas and sparse areas and allow discrimination between them without knowing underlying distributions. Density-based clustering methods “make no assumptions about the number of clusters or their distribution”[3], so this is the approach attempted in this thesis.

Density-based clustering approaches often rely instead on calculating the density of the data at a high granularity, such as sectors of a single dimension, then connecting the areas of high density together with similar areas in other dimensions and sectors to form clusters of dense areas[3]. Thus, the clusters are connected together by density of points and do not need to have a particular shape to be clustered properly. The immediately-apparent downside is that sparsity is treated as noise so if malicious traffic is, or can be made to appear, less dense it will likely remain undetected by this type of algorithm. On the other hand, perhaps benign traffic will not be treated as noise, so only the malicious traffic would stand out. Neither scenario is entirely likely.

Another problem that has already been dealt with in the literature is that noise points between clusters might be dense enough to chain together dense regions when they should be separated. The solution cited by Aggarwal[3] and by Campello[11], who attributes the idea to Wishart, is to eliminate non-dense points first. These are defined by Aggarwal as those having too few neighbours. One way of envisioning this is to think of the dense region as a mound or hill on a flat surface. The bottom of the hill spreads in all directions, but is not particularly thick. As one travels up the hill, the ground beneath becomes thicker and thicker - or more dense. Now imagine trying to separate the dense parts of the hill from the less dense parts of the hill, using some threshold for what is considered to be dense. The intuitive thought is to slice through the hill in a plane parallel to the flat surface. An alternative way of thinking is to imagine raising the flat plane in the direction of the top of the hill. As the surface rises through the hill, the bottom parts of the slope are eliminated from the cluster. The result is that the “thinner” or sparser parts of the hill that are found in the low-lying areas get removed. This analogy is deficient in that raising the flat surface up through a hill would remove points straight down from the peak of the hill, which represent points at the centre, or most dense area, of the cluster. The 2-dimensional projection of such an action would show points throughout the point cloud being eliminated. The better analogy might be something akin to using a cookie cutter on the hill, slicing through the lower slope vertically to remove the less dense regions there, leaving a short cliff face at

the edge of the cluster.

4.6.1 Density Estimation, Thresholding, and Hierarchy

It is assumed that clusters are high density areas of an unknown probability density from which the dataset has been sample. Nearest neighbour and kernel estimators have often been used to find local density estimates[11]. Density level sets are constructed based on a density threshold and how closely connected the points are to each other to form density level sets, with points that are not part of such a level set defined as noise. Global or local density thresholds can be set. A single threshold partitions the data, whereas a hierarchical structure allows consideration of a range of threshold values.

Lower- and higher-density clusters can also be allowed by varying a threshold regionally. This can be particularly important because density levels may vary by data region, and some lower-density clusters could even have other higher-density clusters nested within them[11]. Hierarchical clustering has traditionally allowed these variations to be captured in the tree structure. Hierarchical algorithms output a bottom-up or top-down breakdown of data into groups, as in Figure 2, so that a cut at a particular level gives more- or less-dense clusters, which can be identified by following the branches of the tree.

Figure 2: Illustration of a Dendrogram based on hierarchical clustering of independent components over 100 ICA runs on the CICIDS2017 Tuesday data.

4.6.2 Similarity and Dissimilarity Measures

Dissimilarity is a way to measure the “closeness” or distance between two objects, i and j . Such a method of measurement may be open to subjectivity. Observations may be clustered using how near they are to each other, whereas variables are typically clustered based on correlation or association measurements[34]. Distance is always positive, so $d_{i,j} \geq 0$. The distance between an object and itself is zero, so $d_{i,j} = 0$. The squared distance in Euclidean space is a common dissimilarity function, but others such as the L_1 norm may be used to derive similarities, and thus dissimilarities, between objects.

Measures of distance include the Euclidean distance, Minkowski distance, Canberra distance, Czekanowski distance, Hamming distance, and so on. Some of these only apply to nonnegative values. Others may not have definitions for categorical data. Most satisfy the triangle inequality, but data may not be as cooperative. Binary variables, for example, can be used to compare presence or absence of a characteristic.

4.6.3 DBSCAN and HDBSCAN

Density-Based Spatial Clustering of Applications with Noise, or DBSCAN was originally introduced in 1996[17]. It is intended to apply to spaces of any number of dimensions. Clusters are defined based on the neighbourhood around each point containing at least a specified minimum number of points within ϵ distance. An ϵ -neighbourhood of a point is defined for point p as $N_\epsilon(p) = \{q \in D | \text{dist}(p, q) \leq \epsilon\}$ DBSCAN works with any distance function.

To deal with boundary values of a cluster, for every point p in a cluster C there is required to exist another point q such that p is within ϵ of q and the ϵ -neighbourhood of q contains at least the minimum number of points. This means that p may not have enough points in its ϵ -neighbourhood, but since q does, and q can reach p , p is within the cluster C . These points are density-reachable with respect to each other. Points that may not be directly density-reachable with respect to each other, but that can be chained together with other density-reachable points until they connect, are considered density-connected. Clusters are defined as the maximal set of density-connected points. Noise points are defined as those not belonging to a cluster, meaning they are not density-connected to the cluster[17]. The full definitions and algorithms are available in [17].

Figure 3: Illustration of an HDBSCAN cluster. The Noise point is not density-reachable or density-connected to the rest of the points. The points encircled in orange are density-connected and are considered core points of the cluster. The points encircled in blue are density-reachable to each other and to the core points, but are considered border points in HDBSCAN. In the HDBSCAN algorithms, the core density-connected points are retained for clusters, whereas the density-reachable ones are counted as noise or outliers. The radii of the circles represents the ϵ -neighbourhood of the points. Image idea based upon an illustration in [78].

In the time since DBSCAN was introduced, it has experienced considerable popularity and additional development. The Hierarchical Density-Based Spatial Clustering of Applications with Noise, or HDBSCAN, algorithm, first demonstrated by Campello and colleagues in 2013[12][13], and accelerated by McInnes and Healy[52] as HDBSCAN*, makes some improvements to the algorithm based on a density-contour cluster and tree models by Hartigan. The newer algorithm removes the use of border points, making these into noise or outlier points instead, and allows clusters of varying densities. The algorithm relies on a transformation into pairwise dissimilarities (mutual reachability distances) based on computing all possible clustering solutions hierarchically using a minimum spanning tree[11]. The accelerated HDBSCAN* algorithm offers the benefit of only requiring the minimum number of points without ϵ being necessary and fulfills most of the desired characteristics for this thesis, so it will be the first choice of clustering algorithms used here. Hereafter, HDBSCAN will be used to mean the accelerated HDBSCAN* of McInnes and Healy. The package is available on github and online for Python¹⁴.

4.6.4 Cluster Validation with HDBSCAN

Campello and colleagues note that standard cluster validity measures rely on hyperspherical or globular clusters rather than arbitrary shapes to judge within-cluster compactness or variance and between-cluster separation, making them intractable in this type of problem[11]. They also note that most previous methods intended to address this type of clustering have failed in some way, aside from Moulavi and colleagues (one of whom is Campello) in 2014[56], who developed a measure called the Density-Based Clustering Validation Index, or DBCV.

Density-Based Clustering Validation relies first on the all-points-core-distance, notated as $a_{pts}coredist$. They define the core distance of an object as the inverse of the density. For object o in cluster C_i and $n_i - 1$ other objects, $a_{pts}coredist(o) = \left(\frac{\sum_{i=2}^{n_i} \left(\frac{1}{KNN(o,j)} \right)^d}{n_i - 1} \right)^{\frac{1}{d}}$.

DBCV is therefore nonparametric, relying only on a cluster having been chosen and having access to each object within the cluster. The core distance of each object falls between the second and the last nearest neighbour distance for the object, so $KNN(o, 2) \leq a_{pts}coredist(o) \leq KNN(o, n)$. This means the core distance of o is approximately the same as a uniformly distributed $KNN(o, j)$, with $j \approx \frac{n}{\ln(n)}$, so if most objects are more densely-packed around o the core distance is smaller[56]. The dissimilarity measure used here is either Euclidean or Squared Euclidean distance. Based on core distance, the symmetric mutual reachability distance (MDS) between o_i and o_j can be defined as $d_{mreach}(o_i, o_j) = \max\{a_{pts}coredist(o_i), a_{pts}coredist(o_j), d(o_i, o_j)\}$. Based on the MDS, a complete MDS Graph between objects as vertices and MDS as edge weights may be

¹⁴<https://github.com/scikit-learn-contrib/hdbscan/blob/master/docs/api.rst> and <https://hdbscan.readthedocs.io/en/latest/api.html>

constructed, and a Minimum Spanning Tree (MST) derived.

These core ideas may be summarized into the procedure used to calculate DBCV. Beginning with a cluster and the observations or objects that comprise it, the core distance of each object is first calculated, then the MRD is calculated for each pair of objects. This is used to generate the MST for the cluster. The process repeats for each cluster. The density sparseness (DSC) of a cluster is the maximum edge of the MST, which reaches the area of lowest density in the cluster. The density separation (DSPC) between clusters is the minimum MRD between the objects of the two clusters. The validity index of a cluster combines both measures. The validity index of the whole clustering solution is a weighted average of the validity indices of all clusters in the solution. Index values fall between -1 and 1 , with larger values corresponding to better clustering solutions[56]. This validation method was tested with both DBSCAN and HDBSCAN* with some success. In this thesis it will be used with HDBSCAN* and considered for one further clustering method.

In the next section, the methods discussed in this section are implemented using the public dataset discussed in Section 3. At each step, the usefulness of the results is evaluated. Based on the efficacy of the results, particular methods are selected for use on the Rank dataset, which is discussed in Section 6.

5 Results

Below are the results for each method tested on the CICIDS2017 dataset. As the results progress, they are informed by the previous result in such a way that the size of the reported results may be reduced. Hence, if a method does not appear particularly promising, it is not continued to the next step. The methods build on each other progressively, with discussion to follow in Section 6. The final selection of methods will be used on the real-world Rank Software dataset in the section following that. The methods used are Classical PCA, Robust PCA (hereafter referred to as RPCA), IP2Vec, ICA, Autoencoder, and Density-Based Clustering.

5.1 Summary Statistics

First, a table of summary statistics of the non-binary, numerical 58 variables is included in Table 1. All values above 1000 have been rounded to the nearest integer for presentation purposes. It should be noted that Flow Bytes/s, Flow Packets/s, and Forward Header Length were modified by reducing Infinite-valued entries to match the maximum non-infinite value. It is also worth noting that throughout this analysis, R had trouble with the Forward Header Length variable, which often resulted in failure during decomposition (usually SVD) due to NA or infinite values. This instability meant that generally this variable was excluded. It is unclear which step of the decomposition caused this, but it appears to be an artifact of rounding error during normalization. It similarly failed during simple pairwise correlation analysis.

Since the values of skewness are rather extreme in general, it should be safe to assume that the variables are asymmetrical. Since the kurtosis is also generally extreme, the peaks are also apparently non-Gaussian. By testing the pairwise differences of medians between each group of traffic (e.g. Benign, and each of the 14 classes of malicious), it becomes apparent that the distributions are not easily distinguished as different from each other based on median difference. Each group has a different number of observations. Due to these unavailable assumptions, in particular the lack of symmetry and the presence of non-Normality, a non-parametric Mann-Whitney test for a difference of medians was used. This assumes independence of the samples and ordinality. It is not clear that the assumption of independence among these groups is well-founded, but assume for now that this is the case. The null hypothesis of this test is H_0 : The distribution of each variable $x_{i,j}$ is not different from the distribution of $x_{i,k}$ for all $i = 1, \dots, 58$ variables, with $j \neq k$ and $j, k = 1, \dots, 15$ being the traffic groups. Among 6,090 pairwise tests, all tests resulted in strong failure to reject the null hypothesis ($p \approx 1$ in all cases). Hence, for now we conclude that there is no evidence of different distributions among these groups.

Table 1: CICIDS2017 Sample Estimates for Numeric Variables

Variable	Mean	Variance	Median	MAD	Skewness	Kurtosis
Flow Duration	14785664	1.13E+15	31316	46394	2.16	6.02
Total Fwd Packets	9.36	562009	2	1.48	244.38	61757
Total Bwd Packets	10.39	994783	2	1.48	244.68	62021
Total Length Fwd Packets	549.30	99871826	62	83.03	805.57	988554
Total Length Bwd Packets	16162	5.12E+12	123	182.36	244.33	61605
Fwd Packet Length Max	207.60	514354	37	45.96	9.85	139.75
Fwd Packet Length Min	18.71	3641	2	2.97	20.14	479.56
Fwd Packet Length Mean	58.20	34630	34	39.73	9.14	97.93
Fwd Packet Length Std	68.91	79066	0	0	10.53	142.95
Bwd Packet Length Max	870.85	3788346	79	117.13	2.94	11.91
Bwd Packet Length Min	41.05	4742	0	0	4.84	75.99
Bwd Packet Length Mean	305.95	366336	72	106.75	2.52	8.47
Bwd Packet Length Std	335.36	705085	0	0	3.41	16.12
Flow Bytes/s	2468998	2.99E+15	4587	6801	34.41	1263
Flow Packets/s	74834	8.03E+10	110.67	163.83	6.65	62.34
Flow IAT Mean	1298449	2.03E+13	11439	16953.30	8.97	126.05
Flow IAT Std	2919271	6.47E+13	137.18	203.38	3.61	17.98
Flow IAT Max	9182475	5.98E+14	30865	45702.63	2.90	10.04
Flow IAT Min	162380	8.70E+12	4	4.45	23.79	627.07
Fwd IAT Total	14482962	1.13E+15	43	63.75	2.18	6.11
Fwd IAT Mean	2610193	9.07E+13	26	38.55	7.07	64.71
Fwd IAT Std	3266957	9.29E+13	0	0	3.41	13.87
Fwd IAT Max	9042939	6.02E+14	43	63.75	2.91	10.06
Fwd IAT Min	1021893	7.38E+13	3	4.45	9.71	103.38
Bwd IAT Total	9893830	8.26E+14	3	4.45	2.88	9.81
Bwd IAT Mean	1805784	7.90E+13	3	4.45	8.25	81.55
Bwd IAT Std	1485973	3.94E+13	0	0	5.79	40.11
Bwd IAT Max	4684692	2.94E+14	3	4.45	4.44	22.42

Continued on next page

Table 1 – continued from previous page

Variable	Mean	Variance	Median	MAD	Skewness	Kurtosis
Bwd IAT Min	967261	6.90E+13	1	1.46	9.92	108.37
Fwd Header Length	-25997	0	64	32	243.52	61547
Bwd Header Length	-2273	2.11E+12	40	35.58	-717.03	528199
Fwd Packets/s	53865	61274630551	61.33	90.78	5.68	40.56
Bwd Packets/s	6995	1455552036	19.83	29.40	21.49	638.02
Min Packet Length	16.43	636.94	2	2.97	10.19	427.27
Max Packet Length	950.40	4113713	87	120.09	2.82	11.33
Packet Length Mean	171.94	93325.03	57.2	75.91	2.40	8.07
Packet Length Std	294.98	399171	25.98	38.52	2.88	11.78
Packet Length Variance	486155	2.71E+12	675	1001	5.39	39.71
Down/Up Ratio	0.68	0.46	1	0	12.00	1767
Average Packet Size	191.98	110131	72.25	96.44	2.49	8.84
Avg Fwd Segment Size	58.20	34630	34	39.73	9.14	97.93
Avg Bwd Segment Size	305.95	366336	72	106.75	2.52	8.47
Subflow Fwd Packets	9.36	562009	2	1.48	244.38	61757
Subflow Fwd Bytes	549.29	99601802	62	83.03	803.60	984884
Subflow Bwd Packets	10.39	994783	2	1.48	244.68	62021
Subflow Bwd Bytes	16162	5.12E+12	123	182.36	244.34	61607
Init Win Bytes Fwd	6990	205599059	251	374	2.56	9.55
Init Win Bytes Bwd	1989	71518873	-1	0	5.25	32.83
Act Data Pkt Fwd	5.42	405038	1	1.4826	284.60	83080
Min Seg Size Fwd	-2742	1.17E+12	24	5.9304	-474.54	233239
Active Mean	81551	4.2E+11	0	0	38.23	3877.14
Active Std	41134	1.54E+11	0	0	40.47	4113.31
Active Max	153183	1.05E+12	0	0	24.35	1326
Active Min	58296	3.33E+11	0	0	47.68	5847
Idle Mean	8316037	5.58E+14	0	0	3.07	11.10
Idle Std	503844	2.11E+13	0	0	10.49	117.39
Continued on next page						

Table 1 – continued from previous page

Variable	Mean	Variance	Median	MAD	Skewness	Kurtosis
Idle Max	8695752	5.93E+14	0	0	2.95	10.30
Idle Min	7920031	5.45E+14	0	0	3.18	11.76
Table 1 – concluded						

The pairwise Pearson correlations among the variables are suggestive of several groupings that could be informative in determining an acceptable number of components in the next steps of the analysis. The Flow Bytes Per Second, the Flow Packets Per Second, and the Forward Header Length variables induced calculation failure in the correlation calculations and were not included. The binary Flag variables were also ignored, since they have not been dealt with throughout this project, but they also exhibited strong correlations with several of the groups described below. I looked for natural groups among the variables by using correlation as a guide. I have called “strong” correlation anything above 70%. There were no strong negative correlations. “Statistics” in this list also refer to typical summary statistics of Maximum, Minimum, Mean, Variance, and Standard Deviation. Most variables available were groupings of these statistics describing the network traffic flows.

1. Forward and Backward Packet Length statistics were strongly correlated with each other and with Packet Size and Segment Size.
2. The Active statistics were strongly correlated with each other only.
3. Flow Duration was strongly correlated with Idle time statistics and with Inter Arrival Times (IAT). Idle time statistics were strongly correlated with each other.
4. Backward and Forward Subflow variables were highly correlated, along with the Active Data Packets Forward variable. Total Forward Packets was also highly correlated with the Active Data Packet Forward variable, and the Backward and Forward Packets and Subflow Bytes variables. Total Backward Packets and Total Length of Backward Packets exhibited the same correlations. Total Length of Forward Packets was perfectly correlated with Subflow Forward Bytes only.
5. Segment Size, Backward Header Length, Down/Up Ratio, Initial Win Bytes Forward and Backward, and the Forward and Backward Packets per Second variables were not strongly correlated with anything.

6. Total Forward and Backward Packets were strongly correlated with each other, with Subflow statistics, and with Total Length and Active Data Packets Forward.
7. The IAT statistics were strongly correlated with each other and with the Idle statistics.

It is easy to see, then, that there are several groupings at first glance. A simple dissimilarity matrix generated by calculating $1 - R$ on the pairwise correlation matrix gives a basic hierarchical clustering using average linkage as presented in Figure 4. It is easy to see that pruning at a height between 0.8 and 1.0 results in around 9 or 10 natural groupings of variables, with a few standalone variables. This seems like a good thing to keep in mind while proceeding with dimension reduction in the next stages of this analysis.

Figure 4: Hierarchical clustering of dissimilarity matrix based on Pearson correlation between variables.

5.2 Classical PCA Results: Global Dataset

Because PCA relies on a linear transformation onto the directions of greatest variability in the data and there is a lack of evidence that the distributions differ, a linear transformation of the data onto the principal component

axes might not provide an improvement in separating the classes into discernible clusters, especially not using global PCA. Many of the variables are on different scales, so each variable is first centered and scaled to zero mean and unit variance; however, due to the heavy skew and kurtosis, a robust centering and scaling method is also considered. Furthermore, the reliance of classical PCA on the variance-covariance matrix makes it sensitive to extreme values, so RPCA might also be appropriate and is considered below.

Below are included tables of the eigenvalues, the proportion of variation explained by them, the cumulative proportion, and the proportion expected under the Broken Stick model discussed in Section 4.1.2. A table is included for classical PCA after classical normalization of the dataset using the mean to center the variables and standard deviation for scaling for the 58 non-binary, non-constant, non-categorical variables in the dataset. A table is also provided showing the results if the median is used for centering and the median absolute deviation is used to scale the data. Furthermore, the same tables are included for a version of the dataset that includes the additional 32-dimensional embeddings from IP2Vec for the categorical network variables (Source IP, Destination IP, Source Port, Destination Port, Protocol). These embeddings are appended to the data prior to running PCA or RPCA, while the categorical versions are removed in both cases. Rather than include all 218 entries in the tables, one component beyond the maximum cutoff under each of the subspace selection methods discussed in Section 4.1.2 is used to determine the lengths of the longer tables.

Table 2: Results of Classical PCA with mean centering, scaled using standard deviation, on Global CICIDS2017 Dataset with 58 Variables

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
1	15.008	25.876	25.876	8.0
2	9.132	15.745	41.621	6.3
3	5.861	10.104	51.250	5.4
4	4.074	7.024	58.750	4.8
5	2.980	5.139	63.887	4.4
6	2.496	4.303	67.190	4.1
7	2.169	3.739	71.929	3.8
8	1.879	3.240	74.170	3.5
9	1.553	2.678	77.845	3.3
10	1.522	2.623	80.470	3.1
11	1.274	2.197	82.667	3.0
Continued on next page				

Table 2 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
12	1.159	1.999	84.665	2.8
13	1.123	1.936	86.602	2.7
14	1.017	1.754	88.356	2.5
15	0.975	1.682	90.037	2.4
16	0.923	1.591	91.628	2.3
17	0.848	1.462	93.090	2.2
18	0.776	1.338	94.428	2.1
19	0.749	1.292	95.720	2.0
20	0.626	1.080	96.799	1.9
21	0.512	0.883	97.682	1.8
22	0.278	0.479	98.161	1.7
23	0.260	0.448	98.609	1.6
24	0.188	0.325	98.933	1.6
25	0.116	0.200	99.133	1.5
26	0.104	0.179	99.311	1.4
27	0.081	0.140	99.452	1.3
28	0.066	0.113	99.565	1.3
29	0.062	0.106	99.671	1.2
30	0.042	0.072	99.744	1.2
31	0.028	0.049	99.792	1.1
32	0.024	0.041	99.834	1.1
33	0.019	0.033	99.867	1.0
34	0.016	0.027	99.894	1.0
35	0.014	0.024	99.918	0.9
36	0.014	0.023	99.940	0.9
37	0.011	0.019	99.960	0.8
38	0.007	0.011	99.971	0.8
39	0.004	0.006	99.978	0.7
Continued on next page				

Table 2 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
40	0.003	0.005	99.983	0.7
41	0.003	0.004	99.987	0.6
42	0.002	0.003	99.990	0.6
43	0.002	0.003	99.993	0.6
44	0.001	0.002	99.995	0.5
45	0.001	0.002	99.997	0.5
46	0.001	0.001	99.998	0.4
47	0.000	0.001	99.998	0.4
48	0.000	0.001	99.999	0.4
49	0.000	0.000	99.999	0.3
50	0.000	0.000	99.999	0.3
51	0.000	0.000	99.999	0.3
52	0.000	0.000	100.000	0.2
53	0.000	0.000	100.000	0.2
54	0.000	0.000	100.000	0.2
54	0.000	0.000	100.000	0.1
56	0.000	0.000	100.000	0.1
57	0.000	0.000	100.000	0.1
58	0.000	0.000	100.000	0.0
Table 2 – concluded				

In Table 2, the average eigenvalue of 1.00 is reached at the 14th eigenvalue. If the scaled version is used, the average eigenvalue is 0.7, which is reached by the 19th eigenvalue. In terms of cumulative proportion of variance explained, 14 PCs explain 88.36% of the variance, while 19 PCs explain 95.72% of the variance. The Broken Stick model makes the cutoff at 5 PCs, based on the eigenvalues and the neighbourhood of the Broken Stick model, as shown in Figure 5. With 5 PCs, 63.88% of the variation in the data is explained. Ideally, a large proportion of the variation might correspond to day of the week, hour of the day, and to traffic type, but it is unlikely to be that easy.

Of note is that the bootstrapping method of determining the number of dimensions for subspace selection

Figure 5: Screeplot of variation explained for PCs using the classical PCA procedure on 58 numerical variables from the CICIDS2017 dataset, mean-centered and standard deviation-scaled, against the expected variation explained under the Broken Stick model. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.

was unsuccessful on this large dataset. It was formulated originally on small samples. The authors of the paper speculated that the hypothesis test would not be sensitive enough on larger ones, speculation that proved accurate here as there was no evidence to exclude any of the PCs based on the tests conducted on the global dataset. The code took roughly one week for a full run and I did not have the opportunity to attempt to build a large sample alternative. In the interests of time and because it did not appear to help in this case, it has not been pursued any further.

Figure 6: Modified biplot (points removed, so more like a uniplot) for PCs using the Classical PCA procedure on 58 numerical variables from the CICIDS2017 dataset, mean-centered and standard deviation-scaled.

Interpretation of principal components can be difficult in the best of times. Interpretation here is con-

founded further by the presence of certain variables in the data that likely overlap with each other. For example, each of the “Forward” traffic variables has a corresponding “Backward” version. A large proportion of the variables are also summary statistics of the flows in each of these two categories, meaning they aggregate and possibly obscure some important characteristics and differences - variation in particular - that might otherwise have been found in the raw packet data. Some of this can be seen in Figure 6, where the majority of the variation along the first principal component axis is made up of a combination of most of the original variables, some of them in the negative direction but most in the positive direction. The majority of the variation along the second component axis is made up of relatively fewer variables

Before proceeding with further results, there are five directions to explore to inform next steps.

1. The embedding vectors could be included in the data to enrich it and possibly allow extraction of more useful component axes.
2. Subsetting the data to only include daily data in the application of PCA could potentially manually separate five causes of variation, or alternatively similarity, that could allow more subtle characteristics to come to light.
3. Robust scaling could be used to account for the extreme skew and kurtosis.
4. RPCA could be used to adjust for the extreme skew and kurtosis in a different way.
5. A PC subspace could be selected based on the above thresholds before proceeding.

Of primary concern in considering these options is the order in which they are carried out. The first step will be to add the embeddings vectors into the data. These are original variables that have been transformed into numeric data, so it is worthwhile to have them part of the process from the early stages onward. Subsequently, the data will be divided into single days of data. Daily benign network traffic may be moderately or highly similar across standard business hours between days, though there will always be some variation in the patterns based on the business and the typical work week, vacations, staffing, and so on. If the data were treated as time series data, the hourly and daily variables would be explicitly included in a model, but that avenue has not been pursued in this project. Splitting the data by day is an easy solution to help compensate and that should make the analysis easier in the later stages. In practice, this approach seems like it would be highly useful, either in static or in streaming approaches. Greater granularity might also be desired, such as an hourly approach. It is likely the daily data will exhibit some additional structure that could be controlled by subsetting by hour, but for now the days will be used. Combining robust scaling and RPCA together might be a consideration, but testing using 1000 projection directions and a maximum of the full complement of

components typically resulted in very few components found by the procedure. In some cases, only 1 component was found. I speculate that this overcompensates for the sharp peaks and most of the variation is sent in a few directions. Regardless, this limits the utility of combining the two.

5.3 Classical PCA with Word Vectors on Daily Data

The IP2Vec embedding procedure takes each of the five categorical variables of Source and Destination IP Address, Source and Destination Port, and Protocol and generates 32-dimensional continuous numerical embeddings for each. This may not be the optimal approach and more time could have been spent optimizing the procedure. However, the embeddings are meant to hopefully enrich the rest of the analysis rather than acting as the focus. The outcome of generating these embeddings is the addition of 160 dimensions to the 58 variables already being used. The starting point is then 218 variables, of which roughly 211 or 212 are retained in the daily data. The Forward Header Length variable was removed, as it became unstable once the individual days were separated, resulting in SVD failing. In addition, constant variables were removed in the daily data, which accounted for the remainder of the variables below 218.

The daily PCA results with classical normalization on the data are presented below. Four of the five days had 211 variables, while the data for the fourth day (Thursday, July 6, 2017) had 212, with the extra variable not being removed because it was not constant. The full tables of eigenvalues and cutoff values will not be presented here. Each table is truncated at the largest dimension suggested by the various cutoffs. In each case, this is the scaled average eigenvalue of 0.7, which falls between the 43rd and 46th component. The Broken Stick cutoff based on the eigenvalues tends to be slightly more aggressive than the theoretical model value would suggest, and is much more aggressive than the average eigenvalue-based methods. It leads to discarding a few extra dimensions based on the eigenvalues than would be discarded by using the model values for the number of dimensions being used.

In the Monday to Friday daily datasets with the classical approach the cutoffs using the Broken Stick model are 11, 12, 9, 12, and 10 components respectively. The mean cutoffs are 33, 34, 33, 35, and 33 components respectively. The scaled mean cutoffs are 44, 45, 45, 46, and 43 components respectively. This suggests relatively stability in the axes of variation across the days based on the narrow range of comparable eigenvalues. For example, the first eigenvalue for each day is between 46 and 52 and similar patterns of decrease can be observed for each of the days, suggesting that the day-to-day variation follows a moderately similar pattern in terms of the component axes.

Figure 7: Screeplot of variation explained for PCs against the expected variation explained under the Broken Stick model using classical PCA on the Monday data from the CICIDS2017 dataset. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.

5.3.1 PCA: Monday

The Monday run of classical PCA results in cutoff values of 11 PCs for the Broken Stick model, selected using the *bsDimension* function from the *PCDimension*[39] package. This value can be seen in Figure 7 as the left-most orange dot on the black curve. The red curve represents the theoretical values given by the *brokenStick* function from the same package, which gives a more conservative cutoff than that based on the eigenvalues themselves. The average eigenvalue cutoff is represented by the middle orange dot and the scaled average eigenvalue is represented by the right-most orange dot. The Broken Stick value appears to be nearest to the inflection point of the eigenvalue curve, while the average and scaled average eigenvalue are closer to the point where the curve would become a straight line, as discussed in Section 4.1.2, with the average eigenvalue occurring at the 33rd component and the scaled one appearing at the 44th component.

Table 3: Results of Classical PCA on the CICIDS2017 Monday Dataset

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
1	50.60	24.00	24.00	2.80
2	21.80	10.40	34.30	2.30
3	13.50	6.40	40.70	2.10
4	9.60	4.60	45.30	1.90
Continued on next page				

Table 3 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
5	9.30	4.40	49.60	1.80
6	8.30	3.90	53.60	1.70
7	6.90	3.30	56.80	1.60
8	6.10	2.90	59.70	1.60
9	5.40	2.60	62.30	1.50
10	4.50	2.10	64.50	1.50
11	4.10	1.90	66.40	1.40
12	3.70	1.80	68.20	1.40
13	3.30	1.60	69.70	1.30
14	3.10	1.50	71.20	1.30
15	2.70	1.30	72.50	1.30
16	2.40	1.10	73.60	1.20
17	2.40	1.10	74.70	1.20
18	2.20	1.10	75.80	1.20
19	2.10	1.00	76.80	1.20
20	1.90	0.90	77.70	1.10
21	1.90	0.90	78.60	1.10
22	1.80	0.80	79.40	1.10
23	1.60	0.80	80.20	1.10
24	1.50	0.70	80.90	1.00
25	1.40	0.70	81.60	1.00
26	1.30	0.60	82.30	1.00
27	1.30	0.60	82.90	1.00
28	1.20	0.60	83.40	1.00
29	1.20	0.60	84.00	0.90
30	1.10	0.50	84.50	0.90
31	1.10	0.50	85.00	0.90
32	1.00	0.50	85.50	0.90
33	1.00	0.50	86.00	0.90
Continued on next page				

Table 3 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
34	1.00	0.50	86.50	0.90
35	1.00	0.50	86.90	0.90
36	1.00	0.50	87.40	0.80
37	0.90	0.40	87.80	0.80
38	0.90	0.40	88.30	0.80
39	0.90	0.40	88.70	0.80
40	0.90	0.40	89.10	0.80
41	0.80	0.40	89.50	0.80
42	0.80	0.40	89.80	0.80
43	0.80	0.40	90.20	0.80
44	0.70	0.30	90.50	0.70
45	0.70	0.30	90.90	0.70
Table 3 – concluded				

At this stage, the use of the chosen version of robust standardization on this data revealed itself to be questionable. Using the median and MAD for normalization resulted in 13 of the original 58 variables having between half and all values converted to NAs. This was not the case with the global data. If normalization using the robust method is done on the global dataset instead of the daily datasets and the subsetting occurs afterward, then the NA problem is ameliorated. If the subsetting is done first, the smaller datasets are more unstable. Most likely, there are rounding errors due to the either very large or very small values here. In the calculation of MAD, the median and many of the X_i s are most likely so close to each other that their absolute difference is essentially zero and the calculation “blows up” and results in an NA. A different version of robust standardization might be a meaningful approach for this type of data but further pursuit of a suitable transformation is beyond the scope of this project. As such, robust standardization was discarded as a method to be used going forward. Instead, a comparison will be made between PCA and RPCA.

5.3.2 PCA: Tuesday

The Tuesday run of classical PCA results in cutoff values of 12 PCs for the Broken Stick model. This value can be seen in Figure 8 as the left-most orange dot on the black curve. The average eigenvalue cutoff is

Figure 8: Screeplot of variation explained for PCs against the expected variation explained under the Broken Stick model using classical PCA on the Tuesday data from the CICIDS2017 dataset. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.

represented by the middle orange dot, occurring at the 36th eigenvalue, and the scaled average eigenvalue is represented by the right-most orange dot, occurring at the 45th eigenvalue.

Table 4: Results of Classical PCA on the CICIDS2017 Tuesday Dataset

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
1	48.30	22.90	22.90	2.80
2	19.00	9.00	31.90	2.30
3	14.20	6.70	38.60	2.10
4	8.90	4.20	42.80	1.90
5	8.80	4.20	47.00	1.80
6	7.40	3.50	50.50	1.70
7	7.10	3.40	53.90	1.60
8	6.70	3.20	57.10	1.60
9	5.40	2.60	59.60	1.50
10	5.10	2.40	62.00	1.50
11	4.40	2.10	64.10	1.40
12	4.30	2.00	66.20	1.40
13	3.70	1.80	67.90	1.30

Continued on next page

Table 4 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
14	3.30	1.60	69.50	1.30
15	2.90	1.40	70.90	1.30
16	2.90	1.40	72.20	1.20
17	2.60	1.20	73.50	1.20
18	2.30	1.10	74.50	1.20
19	2.20	1.00	75.60	1.20
20	2.20	1.00	76.60	1.10
21	2.00	1.00	77.50	1.10
22	1.90	0.90	78.40	1.10
23	1.80	0.90	79.30	1.10
24	1.60	0.80	80.10	1.00
25	1.60	0.80	80.80	1.00
26	1.50	0.70	81.60	1.00
27	1.50	0.70	82.30	1.00
28	1.40	0.70	83.00	1.00
29	1.40	0.60	83.60	0.90
30	1.30	0.60	84.20	0.90
31	1.20	0.60	84.80	0.90
32	1.10	0.50	85.30	0.90
33	1.10	0.50	85.80	0.90
34	1.00	0.50	86.30	0.90
35	1.00	0.50	86.80	0.90
36	1.00	0.50	87.30	0.80
37	0.90	0.50	87.70	0.80
38	0.90	0.40	88.20	0.80
39	0.90	0.40	88.60	0.80
40	0.90	0.40	89.00	0.80
41	0.90	0.40	89.40	0.80
42	0.80	0.40	89.80	0.80
Continued on next page				

Table 4 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
43	0.80	0.40	90.20	0.80
44	0.70	0.40	90.60	0.70
45	0.70	0.30	90.90	0.70
Table 4 – concluded				

5.3.3 PCA: Wednesday

Figure 9: Screeplot of variation explained for PCs against the expected variation explained under the Broken Stick model using classical PCA on the Wednesday data from the CICIDS2017 dataset. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.

The Wednesday run of classical PCA results in cutoff values of 9 PCs for the Broken Stick model. This value can be seen in Figure 8 as the left-most orange dot on the black curve. The average eigenvalue cutoff is represented by the middle orange dot, occurring at the 33rd eigenvalue, and the scaled average eigenvalue is represented by the right-most orange dot, occurring at the 45th eigenvalue.

Table 5: Results of Classical PCA on the CICIDS2017 Wednesday Dataset

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
1	52.60	24.90	24.90	2.80
2	24.10	11.40	36.30	2.30
3	13.40	6.30	42.70	2.10
4	9.80	4.60	47.30	1.90
5	9.10	4.30	51.60	1.80
6	8.50	4.00	55.60	1.70
7	7.40	3.50	59.10	1.60
8	5.80	2.70	61.90	1.60
9	4.80	2.30	64.10	1.50
10	3.90	1.90	66.00	1.50
11	3.80	1.80	67.80	1.40
12	3.60	1.70	69.50	1.40
13	3.30	1.60	71.00	1.30
14	2.90	1.40	72.40	1.30
15	2.60	1.20	73.60	1.30
16	2.30	1.10	74.70	1.20
17	2.20	1.00	75.70	1.20
18	2.10	1.00	76.70	1.20
19	2.00	0.90	77.60	1.20
20	1.80	0.80	78.40	1.10
21	1.70	0.80	79.30	1.10
22	1.60	0.80	80.00	1.10
23	1.60	0.70	80.70	1.10
24	1.40	0.70	81.40	1.00
25	1.30	0.60	82.10	1.00
26	1.30	0.60	82.70	1.00
27	1.30	0.60	83.30	1.00

Continued on next page

Table 5 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
28	1.20	0.60	83.80	1.00
29	1.10	0.50	84.40	0.90
30	1.10	0.50	84.90	0.90
31	1.10	0.50	85.40	0.90
32	1.00	0.50	85.90	0.90
33	1.00	0.50	86.40	0.90
34	1.00	0.50	86.90	0.90
35	1.00	0.50	87.40	0.90
36	0.90	0.40	87.80	0.80
37	0.90	0.40	88.20	0.80
38	0.90	0.40	88.70	0.80
39	0.90	0.40	89.10	0.80
40	0.90	0.40	89.50	0.80
41	0.80	0.40	89.90	0.80
42	0.80	0.40	90.30	0.80
43	0.80	0.40	90.70	0.80
44	0.80	0.40	91.00	0.70
45	0.70	0.40	91.40	0.70
Table 5 – concluded				

5.3.4 PCA: Thursday

The Thursday run of classical PCA results in cutoff values of 12 PCs for the Broken Stick model. This value can be seen in Figure 10 as the left-most orange dot on the black curve. The average eigenvalue cutoff is represented by the middle orange dot, occurring at the 35th eigenvalue, and the scaled average eigenvalue is represented by the right-most orange dot, occurring at the 46th eigenvalue.

Figure 10: Screeplot of variation explained for PCs using classical PCA on the Thursday data from the CICIDS2017 dataset, mean-centered and standard deviation scaled, against the expected variation explained under the Broken Stick model. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.

Table 6: Results of Classical PCA on the CICIDS2017 Thursday Dataset

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
1	46.10	21.70	21.70	2.80
2	19.00	9.00	30.70	2.30
3	15.10	7.10	37.80	2.10
4	12.60	5.90	43.80	1.90
5	8.20	3.90	47.60	1.80
6	8.00	3.80	51.40	1.70
7	7.90	3.70	55.20	1.60
8	5.80	2.70	57.90	1.60
9	5.50	2.60	60.50	1.50
10	4.60	2.20	62.60	1.50
11	4.40	2.10	64.70	1.40
12	4.10	1.90	66.60	1.40
13	3.60	1.70	68.40	1.30
14	3.10	1.50	69.80	1.30

Continued on next page

Table 6 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
15	2.80	1.30	71.10	1.30
16	2.70	1.30	72.40	1.20
17	2.40	1.10	73.50	1.20
18	2.20	1.00	74.60	1.20
19	2.10	1.00	75.60	1.20
20	2.00	0.90	76.50	1.10
21	1.90	0.90	77.40	1.10
22	1.80	0.90	78.20	1.10
23	1.80	0.80	79.10	1.10
24	1.70	0.80	79.90	1.00
25	1.60	0.80	80.70	1.00
26	1.60	0.70	81.40	1.00
27	1.40	0.70	82.10	1.00
28	1.30	0.60	82.70	1.00
29	1.30	0.60	83.30	0.90
30	1.30	0.60	83.90	0.90
31	1.10	0.50	84.40	0.90
32	1.10	0.50	84.90	0.90
33	1.10	0.50	85.40	0.90
34	1.00	0.50	85.90	0.90
35	1.00	0.50	86.40	0.90
36	1.00	0.50	86.90	0.80
37	1.00	0.50	87.30	0.80
38	1.00	0.40	87.80	0.80
39	0.90	0.40	88.20	0.80
40	0.90	0.40	88.70	0.80
41	0.90	0.40	89.10	0.80
42	0.90	0.40	89.50	0.80
43	0.90	0.40	89.90	0.80
Continued on next page				

Table 6 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
44	0.80	0.40	90.30	0.70
45	0.80	0.40	90.60	0.70
46	0.70	0.30	91.00	0.70
Table 6 – concluded				

5.3.5 PCA: Friday

Figure 11: Screeplot of variation explained for PCs using classical PCA on the Friday data from the CI-CIDS2017 dataset, mean-centered and standard deviation scaled, against the expected variation explained under the Broken Stick model. Eigenvalues are in black. Broken Stick values are in red. Cutoff values are orange circles.

The Friday run of classical PCA results in cutoff values of 10 PCs for the Broken Stick model. This value can be seen in Figure 10 as the left-most orange dot on the black curve. The average eigenvalue cutoff is represented by the middle orange dot, occurring at the 33rd eigenvalue, and the scaled average eigenvalue is represented by the right-most orange dot, occurring at the 43rd eigenvalue.

Table 7: Results of Classical PCA on the CICIDS2017 Friday Dataset

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
1	46.20	21.90	21.90	2.80
2	19.90	9.40	31.30	2.30
3	15.10	7.20	38.50	2.10
4	13.40	6.40	44.90	1.90
5	9.90	4.70	49.60	1.80
6	9.40	4.40	54.00	1.70
7	9.00	4.20	58.20	1.60
8	6.60	3.10	61.40	1.60
9	5.80	2.80	64.10	1.50
10	4.70	2.20	66.40	1.50
11	4.00	1.90	68.30	1.40
12	3.90	1.90	70.10	1.40
13	3.50	1.70	71.80	1.30
14	3.00	1.40	73.20	1.30
15	2.60	1.20	74.40	1.30
16	2.40	1.10	75.50	1.20
17	2.20	1.00	76.60	1.20
18	2.10	1.00	77.60	1.20
19	2.00	1.00	78.50	1.20
20	1.90	0.90	79.40	1.10
21	1.90	0.90	80.30	1.10
22	1.70	0.80	81.10	1.10
23	1.60	0.80	81.90	1.10
24	1.50	0.70	82.60	1.00
25	1.40	0.70	83.30	1.00
26	1.30	0.60	83.90	1.00
27	1.20	0.60	84.50	1.00
28	1.20	0.60	85.10	1.00
Continued on next page				

Table 7 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
29	1.20	0.50	85.60	0.90
30	1.10	0.50	86.10	0.90
31	1.10	0.50	86.60	0.90
32	1.00	0.50	87.10	0.90
33	1.00	0.50	87.60	0.90
34	1.00	0.50	88.10	0.90
35	1.00	0.50	88.50	0.90
36	0.90	0.40	89.00	0.80
37	0.90	0.40	89.40	0.80
38	0.90	0.40	89.80	0.80
39	0.80	0.40	90.20	0.80
40	0.80	0.40	90.60	0.80
41	0.70	0.30	90.90	0.80
42	0.70	0.30	91.30	0.80
43	0.70	0.30	91.60	0.80
Table 7 – concluded				

5.4 RPCA: Daily Results

Each run of RPCA based on Hubert and colleagues[28] using the modification to deal with skew resulted in roughly 12 to 14 components. When attempted with robust normalization, the result was typically 1 to 3 vastly inflated components. As stated above, that method was dropped from future analysis. The results in Table 8 included below list the eigenvalues obtained for each daily dataset using 1000 directions for projection pursuit, with the maximum number of eigenvalues set to the maximum of the dataset dimensions. Computational power was a limiting factor in choosing the number of pursuit directions, but increasing from as low as 250 directions to as high as 1000 directions did not appear to make a substantial improvement to the success of the method in finding additional directions, but using fewer pursuit directions tended to result in RPCA locating slightly fewer components.

In general, using the same seed for each run of RPCA on different days meant the eigenvalues across

days remained fairly stable. This may not be the case with random initializations. Coincidentally, the number of robust components with the seed I used compared favourably to the Broken Stick and elbow methods of selecting the number of components to retain. An easy explanation is that beyond the elbow the slope of the eigenvalues, denoting additional variation explained by adding another component, levels off, making it more difficult for a projection pursuit algorithm to locate new component directions beyond that point without many more attempts. Guided projection pursuit informed by the classical components, but perhaps accounting for the detected outliers in the RPCA method, might be a good way to boost the algorithm, but is yet another thing that is beyond the scope of this project.

The mean, variance, and cutoff values described in Section 4.2 for orthogonal distances and robust score distances are provided in Table 9. The two distance scores correspond to the subsets associated with the observations not considered outliers in the initial subspace and the observations not considered outliers in the final subspace. The final subspace tends to include the vast majority of observations, meaning that few malicious traffic observations are flagged as final outliers by RPCA. The initial subset captures some, though certainly not all, of the malicious traffic, but a large proportion of the outliers are also benign. Still, the initial h -subset, called the H_0 subset, could be a useful addition to the analysis and identification of the malicious traffic. I speculate that the H_0 subset might be roughly equivalent to, and could be retained for further use in the same way as, an autoencoder’s hidden layer.

Table 8: Results of Robust PCA on the CICIDS2017 Daily Datasets

Component	Monday	Tuesday	Wednesday	Thursday	Friday
1	53.36	50.75	53.43	46.91	44.25
2	19.75	17.94	20.46	16.00	19.47
3	13.10	13.83	10.57	13.19	10.09
4	7.98	8.15	8.34	10.12	8.78
5	3.79	4.46	3.24	7.23	6.73
6	3.46	4.32	3.18	3.91	4.75
7	2.86	3.81	2.06	3.66	3.60
8	2.75	2.93	1.77	3.52	2.59
9	1.55	1.76	1.34	3.05	1.51
10	0.63	0.82	0.80	2.46	1.04
11	0.35	0.76	0.73	1.20	0.47
12	0.21	0.47	0.56	0.57	0.36
13		0.42	0.44	0.25	
14			0.27		

5.5 Autoencoder Results

Since autoencoders can be considered a version of PCA (see Section 4.3), PCA dimension reduction or dimension selection methods will be used for the choice of the number of nodes in the hidden layer of the

Table 9: Robust PCA Score Distance (SD) and Orthogonal Distance (OD) Measures on the CICIDS2017 Daily Datasets.

Day	SD Mean	SD Var	SD Cutoff	OD Mean	OD Var	OD Cutoff
Monday	3.96	11.99	17.28	6.72	29.36	15.60
Tuesday	3.81	8.63	14.51	6.69	29.23	13.56
Wednesday	4.13	15.96	11.99	6.33	18.06	26.29
Thursday	4.28	8.58	12.43	6.51	15.23	33.32
Friday	4.38	10.40	11.33	6.33	29.16	27.64

Table 10: RPCA H_0 Subset Correspondence to Traffic Types on CICIDS2017 Daily Datasets

Traffic Type	H_0 Outlier	% of Traffic	H_0 Inlier	% of Traffic
Monday				
BENIGN	132435	24.99	397483	75.01
Tuesday				
BENIGN	97598	22.59%	334476	77.41%
FTP-Patator	7937	99.99%	1	0.01%
SSH-Patator	5897	100.00%	0	0.00%
Wednesday				
BENIGN	159717	36.30%	280314	63.70%
DoS GoldenEye	788	7.66%	9505	92.34%
DoS Hulk	6190	2.68%	224883	97.32%
DoS Slowhttptest	3508	63.79%	1991	36.21%
DoS slowloris	2928	50.52%	2868	49.48%
Heartbleed	0	0.00%	11	100.00%
Thursday				
BENIGN	112810	24.70%	343942	75.30%
Infiltration	30	83.33%	6	16.67%
Web Attack \x96 Brute	1236	82.02%	271	17.98%
Web Attack \x96 Sql Injection	0	0.00%	21	100.00%
Web Attack \x96 XSS	620	95.09%	32	4.91%
Friday				
BENIGN	150177	36.25%	264145	63.75%
Bot	1965	99.95%	1	0.05%
DDoS	21170	16.54%	106857	83.46%
PortScan	2454	1.54%	156476	98.46%

autoencoder. While the Broken Stick model accounted for only 64% to 66% of the variation in the data, variation may not be the primary method of discriminating between malicious and benign traffic.

There are a few options. The first is to use something near the Broken Stick cutoff. For example, the Broken Stick model and the RPCA cutoff tend to fall around the same number of dimensions. This suggests that despite not accounting for a large proportion of variation, it is difficult to separate the remaining components and multiple heuristics suggest these extra dimensions may be less meaningful in some way. The Broken Stick and RPCA cutoffs also happen to coincide closely with the hierarchical clustering done on the pairwise correlations between the original variables in Section 5.1, adding strength to this decision. A second option is to use the average eigenvalue cutoff, which retains the majority of the variation without keeping much more

than half of the original variables, or roughly 20% of the dimensions after adding the embeddings. The other options, most important of which is a grid search, have been discussed in Section 4.3. Some exploratory work using a grid search with a large but not exhaustive list of possibilities for as many as five hidden layers suggested a single hidden layer with around 40 to 50 nodes as the best option of those tested. This corresponds roughly to the most conservative of the PCA cutoffs, the scaled average eigenvalue, adding a vote to the more conservative end. In the interests of simplifying this project, I have decided to go with the more aggressive cutoffs. A cutoff around 0.7 on the dendrogram for the correlation on the global data resulted in 14 variable groupings. The Broken Stick model gives daily cutoffs of 11, 12, 9, 12, and 10, while RPCA gives daily cutoffs of 12, 13, 14, 13, 12. The maximum of these cutoffs, the RPCA one, will be used for the remainder of this project.

It is still difficult with unsupervised learning to identify which clusters are meaningful for threat detection. The malicious traffic, in most cases, will not be outliers, and may not even be separate clusters. The autoencoder performs this function arguably better than PCA and RPCA, at least in some cases. For example, the first three dimensions of the hidden layer of an earlier autoencoder run on the Friday data is shown in Figure 27, it is easy to see the classes in separate clusters. A way of identifying which clusters are malicious still needs to be developed, but this makes it clearer that an autoencoder with a lower-dimensional hidden layer may be a good first step toward locating clusters.

It is worth mentioning that there is deeper apparent structure in this visual representation of the data that probably reflects hourly data. For example, in Figure 12, displays three dimensions of the 10-dimensional hidden layer of an autoencoder run on the July 7, 2017 data subset from 4:00pm to 5:00pm. The malicious traffic looks like a reflection of the benign traffic across an unknown axis. While further research would be needed to do so, characterization of such a relationship could provide a simple pattern-based way to exclude a whole class of malicious behaviour. However, for this analysis the daily data will be used.

The hidden layers were extracted from an unsupervised autoencoder based on a hyperbolic tangent activation function run over 200 epochs. The anomaly scores are based on the Reconstruction Mean Square Error for each observation, or by feature. For simplicity, the scores by observation are used here. These scores are ordered and plotted by index, as shown in Figure 13, which shows the reconstruction error for the Friday data. The x-axis in these plots is not particularly meaningful, and given that this dataset is synthetic the patterns visible in their indices should not be taken as suggestive of anything important here. The rest of the daily plots are excluded, but closely resemble the ones shown here. Red points represent benign traffic, while any other colour represents malicious traffic. In most cases, the indices of benign and malicious traffic are interspersed with each other and there is no clear pattern to the quantity of reconstruction error for malicious traffic as being noticeably different from benign traffic. The ordered reconstruction error is shaped roughly

Figure 12: First three dimensions of the 10-node hidden layer of an autoencoder run on the Friday CI-CIDS2017 dataset for a single hour from 4:00pm to 5:00pm. Malicious traffic is green and benign traffic in red.

the same on each day, suggesting no major differences day by day based on the proportion of the data that was outlying.

Typically, the area under a receiver operating characteristic (ROC) curve would be used to evaluate the reconstruction error, but this is not a classification problem. In clustering the lack of labels means we cannot quantify the classification error in the same way. The unsupervised method that works best, tested on labelled data, might reasonably be chosen to use for clustering, but comparison of the relative success will be difficult when there are no labels. Another reasonable approach might have been to train the autoencoder on the Monday dataset as a baseline model and test it on the other days to see whether it detects greater error in the days with malicious activity. Some attempts to do such a comparison are discussed in Section 2.

The first two dimensions of the hidden layers are shown below for the Monday, Wednesday, and Friday datasets. The Monday visualization in Figure 14 shows that much of the data is clustered into relatively tight elliptical shapes in the absence of malicious traffic, although there are some linear bar-shaped elements. As

Figure 13: Anomaly scores based on Reconstruction MSE for each observation in the Friday CICIDS2017 dataset.

mentioned previously, some of the structure could be explained by the hourly time element. For example, each relatively large cloud might be associated with hourly traffic or with a group of similar machines within the network. Further investigation would be needed to determine whether one of these is the case. The malicious traffic does not appear to retain the elliptical shape as much, at least in the dimensions on display in Figures 15 and 16. While the Monday data shows groups of observations of roughly similar size and shape, the Wednesday and Friday plots show elongated masses and smaller, oddly-shaped point clouds of malicious observations. The benign traffic may have been influenced by the presence of the malicious observations, as some of the clouds of benign observations are also shaped differently in the latter two plots.

For the planned density-based clustering, so far the autoencoder results appear visually to be the most likely to yield separate clusters of malicious and benign traffic than the PCA and RPCA results. In some choices of three dimensions the separation is less apparent, but in others there is total separation. However, there is one final thing to check. The principal axes are uncorrelated but not independent.

5.6 Results: ICA

An Icacasso implementation by Erkkka Heinila¹⁵ based on Himberg[27] was used in Python. While it allows for bootstrapping as well as random initialization, only the random initializations were used. Each daily dataset

¹⁵<https://github.com/Teekuningas/icasso>

Figure 14: First two dimensions of the autoencoder hidden layer for the Monday CICIDS2017 dataset.

was first run through PCA, RPCA, and the autoencoder and its dimension reduced. The resulting dataset was passed to Icasso, which ran 30 iterations of *FastICA* with random starting states. The deflation rather than parallel version of *FastICA* was used, with the exponential function. A cluster quality index based on distance between components, as discussed in Section 4.4.4, is meant to suggest a number of clusters in a similar way to a scree plot.

However, this implementation of Icasso does not provide output of a cluster index, just the individual scores, so it is good for providing the iterated source components and the visualizations shown here. Based on these visualizations, there is some room to remove a few components that did not appear to generate the tight clusters, or equivalently a high index, that would suggest they were stable components. The original authors, Himberg and Hyvärinen, explored the trustworthiness of spatial proximity in this case and found multidimensional scaling (MDS) to be slightly less trustworthy than curvilinear component analysis, but it was not terrible and it is what we have. Referring to Figure 17, which shows an example of MDS on the resulting component clusters from Icasso when using the Tuesday data using the autoencoder, it would be easy to suppose that there are 8 or 9 relatively stable components. Tight clusters after that are less easily noticed, suggesting that components that could be grouped together beyond the first 9 were discovered less often or that some of these components were not stable across the 30 iterations of *FastICA*.

A different way of examining the relationships between components can be seen in Figure 18 from the

Figure 15: First two dimensions of the autoencoder hidden layer for the Wednesday CICIDS2017 dataset.

Python output of the run of the autoencoder version of the Tuesday data, which shows the dendrogram for the hierarchical clustering of the components. A higher quality index from Icasso corresponds to more stable components, which should roughly correspond to a higher cutoff value and fewer component clusters from the dendrogram. The height of the splits on the branches can act as a way of determining the quality of a cluster of components. Branches that split near the bottom of the tree may not represent stable components, may represent weak signals that are difficult to detect over only a few runs of ICA but are nevertheless present, or may not represent components at all.

Since actual component clustering labels were also not available in the Python implementation of Icasso, I have clustered the source matrix that was generated. I used agglomerative hierarchical clustering with average linkage in R using *agnes* from the *cluster* package, extracted cluster indices by column using a cutoff of 0.88, and averaged the resulting components as described in the literature. That results in around the same number of high quality clusters of components as would be expected if the number of sources were roughly equal to the dimension reduction carried out. For example, on the Tuesday autoencoder data, there were 8 clusters. Setting a height of around 0.8 or 0.9 for the dendrogram cutoff generally resulted in between 8 and 10 relatively high quality components in a given dataset.

To simplify this project further, the three Tuesday datasets has been selected to carry out clustering moving forward. In part, this is because there are too many subsets and iterations to fully examine here. The Tuesday

Figure 16: First two dimensions of the autoencoder hidden layer for the Friday CICIDS2017 dataset.

datasets exhibited a few moderately well-defined malicious clusters in the autoencoder hidden layer; however, these were embedded in the primary cloud of the benign traffic observations, so they will not be trivial for density-based clustering to extract. For this data, the first component of the RPCA output, shown in Figure 23, showed a very similar shape, with the two types of malicious traffic centered around one side of the main benign traffic point cloud. The resulting averaged independent components based on the Icasso method can be seen in Figure 19, where it is clear that the majority of the malicious traffic has been separated from the main observation cloud. While there are other small clusters present around the malicious traffic, they are each small, allowing much simpler investigation. Similar results occur in the PCA and RPCA versions of the Tuesday data. At the same cutoff of 0.88 on the dendrogram, there were 8 components for PCA and 9 for RPCA. Using the averaged components, clusters of the majority of the malicious traffic become separated from the main point clouds in the same way as seen in Figure 19. It is now a reasonable time to cluster the data, a task that will be carried out on the three ICA versions of the data.

5.7 Clustering Results

Density-based clustering using HDBSCAN* was carried out each of the Tuesday datasets derived from averaging the independent components.

Figure 17: Multidimensional Scaling plot of independent components for ICA run on the Tuesday Autoencoder subset of 12 dimensions.

5.7.1 PCA Clustering

The minimum cluster size was set to 1000 to allow for some of the noise to be excluded. For the PCA version, the minimum number of clusters was set to 15 based on visual inspection of most two-dimensional and three-dimensional scatterplots of the averaged independent components, similar to that shown in Figure 19. In order to allow for very large clusters, based on the main cloud visible in the same figure, the parameter to allow for a single cluster was enabled. Single large clusters can utilize the old DBSCAN epsilon neighbourhoods without greatly slowing down the algorithm, so a cluster selection epsilon neighbourhood was set at 0.0002 based on a fraction of roughly 10% the smallest univariate IQR of the averaged independent components across the entire dataset.

With these settings, there were three clusters. The majority of the malicious traffic ended up in the smallest cluster, which only missed 7 observations. The 1 malicious observation found in the largest cluster tended to be found there in any visualization, visually lying far away from the rest of the malicious observations in most dimensions. It is easy to see why 19.6% of Cluster 1 is actually benign traffic, as there were several ellipsoid-shaped groups of benign traffic lying nearby to the two malicious traffic ellipsoids along most axes, as shown in Figure 20. These non-malicious clusters were near enough that in terms of density that all of these smaller groups were merged into a single cluster. A more conservative epsilon would result in more like 30 or 40 clusters based on other trials, so these might be separated.

Another way to look at the clusters is the outlier scores generated by HDBSCAN. These scores fall between 0 and 1, with higher scores corresponding to greater outlyingness. These scores approximate the density-based cluster validation index described in Section 4.6.4. For a conservative outlier score of 0.7,

Figure 18: Icacso dendrogram of independent components for ICA run on the Tuesday PCA subset of 12 dimensions.

Table 11: PCA-ICA Cluster Assignment

Label	Outliers	Cluster 1	Cluster 2	Cluster 3
Benign	3295	3373	17971	407435
FTP-Patator	0	7937	0	1
SSH-Patator	6	5891	0	0

almost all of the observations can be seen to be inliers in Table 12. Such low numbers of outlying points suggests that the density-based approach is worthwhile because any observation, malicious or not, is more likely than not to be found in a dense cloud. This is confirmed by the strength of the inclusion probabilities, shown in Table 13, as the majority of the data is above 95% probability of inclusion, defined in this table as “strong” inclusion.

Table 12: PCA-ICA Outlier Scores Above and Below 0.7

	Benign	FTP-Patator	SSH-Patator
Outlier	3102	17	51
Inlier	428972	7921	5846

Table 13: PCA-ICA Probability Scores above and below 0.95

Inclusion Strength	Benign	FTP-Patator	SSH-Patator
Strong	422785	7928	5860
Weak	9289	10	37

Figure 19: Three independent components for averaged ICs based on Icasso on the Tuesday autoencoder dataset.

5.7.2 RPCA-ICA Clustering

For the RPCA data with independent components, the minimum cluster size was again set to 1000 to allow for some of the noise to be excluded from the clusters. The minimum number of clusters was set to 15 based on visual inspection of most two-dimensional and three-dimensional scatterplots of the averaged independent components. The same general parameters were used as in the PCA clustering. The cluster selection epsilon neighbourhood was set at 0.00001 based on a fraction of roughly 2% of the smallest IQR of the averaged independent components again.

Figure 21 displays one clustering, where the clusters were allowed to be smaller by using a narrower epsilon neighbourhood. A larger epsilon neighbourhood in the HDBSCAN* algorithm can allow clusters to be connected without compromising the computational efficiency substantially and accounts for the possibility of a single large cluster. A total of 49 clusters were located, with four of them including 99.9% malicious traffic (with 1 benign observation), and 0.35% of malicious traffic being classified as outliers. The majority of these appear to be located near the two malicious cluster groupings, but too far away in some dimension to have been reached by the MST.

The outlier scores for this version of the data were similar to those of the PCA version. Most observations were classified as members of a cluster at scores of 0.7 or below. Table 15 shows that slightly less than 2% of all observations have outlier scores above this level. Even at a cutoff score of 0.2 less than one third of all

Figure 20: Three IC dimensions coloured based on cluster assignment in Figure 11. Red observations are Cluster 1. Green observations correspond to Cluster 2. Blue observations correspond to Cluster 3. Black observations correspond to outliers.

observations are considered outliers. The inclusion probabilities, shown in Table 16 are somewhat weaker, but this is to be expected with the larger number of clusters. At a 95% probability threshold, between 60% and 80% of the malicious traffic and roughly 70% of the benign traffic is predicted to be included in the clusters generated here. A tighter neighbourhood would increase these values and decrease the number of clusters, with the tradeoff of generating more outliers.

Table 14: RPCA-ICA Cluster Assignment

Label	Outliers	Cluster 3	Cluster 4	Cluster 6	Cluster 7	Other Clusters
Benign	55933	1	0	0	0	376140
FTP-Patator	29	0	0	3949	3960	0
SSH-Patator	19	2909	2969	0	0	0

Table 15: RPCA-ICA Outlier Scores above and below 0.7

	Benign	FTP-Patator	SSH-Patator
Outlier	6832	24	1
Inlier	425242	7914	5896

Table 16: RPCA-ICA Probability Scores above and below 0.95

Inclusion Strength	Benign	FTP-Patator	SSH-Patator
Strong	165875	2632	2497
Weak	266199	5306	3400

Figure 21: Clustering for RPCA-ICA data. Red and green represent the four malicious clusters with only one benign point included, while 48 of the black points represent malicious observations that were misclassified as outliers.

5.7.3 Autoencoder-ICA Clustering

For the autoencoder data with independent components, the minimum cluster size was again set to 1000 to allow for some of the noise to be excluded from the clusters. The minimum number of clusters was set to 15 for consistency, though as many as 40 clusters could be visually identified. The same general parameters were used as in the other versions. The cluster selection epsilon neighbourhood was set at 0.0000001 based on a fraction of roughly 2% of the smallest IQR of the averaged independent components. The clusters were similar to the previous versions. There were more outliers overall in this version if one compares Table 15 to Table 18, but the difference would be difficult to quantify. The strength of the inclusion probabilities is also fairly comparable.

Table 17: Autoencoder-ICA Cluster Assignment

Label	Outliers	Clusters 4-5	Clusters 6-7	Other Clusters
Benign	46814	0	1	385260
FTP-Patator	10	0	7927	0
SSH-Patator	12	5885	0	0

Table 18: RPCA-ICA Outlier Scores above and below 0.7

	Benign	FTP-Patator	SSH-Patator
Outlier	13922	180	74
Inlier	418152	7758	5823

Table 19: Autoencoder-ICA Probability Scores above and below 0.95

Inclusion Strength	Benign	FTP-Patator	SSH-Patator
Strong	198265	2665	2460
Weak	233809	5273	3437

5.7.4 Pairwise Cluster Distances

In Table 20 can be seen the distance between the malicious clusters and the single large benign cluster as well as the likely-meaningless centroid of the outliers. The centroid of the outliers is simply the center of the overall mass of points that were not part of any cluster, which is here represented with as the cluster numbered -1. Table 21 shows some the largest and smallest pairwise distances between clusters in the case where a larger number of clusters is allowed. Some of the largest distances can be seen to correspond to the four malicious clusters (clusters 3, 4, 6, and 7), but not all of them. Most of the largest distances do correspond to the group of clusters that had separated from the main cloud of observations in the visualizations, as would be expected.

Table 20: Pairwise Distances Between Clusters: Tuesday Data and PCA

Comparison No.	Cluster 1	Cluster 2	Distance
1	-1	2	0.000732
2	1	2	0.001772
3	-1	1	0.001813
4	-1	0	0.002466
5	0	2	0.002625
6	0	1	0.003129

Table 21: Pairwise Distances Between Clusters: Tuesday Data and
RPCA

Comparison No.	Cluster 1	Cluster 2	Distance
1	0	4	0.0029768
2	0	3	0.0029648
3	0	7	0.0028955
4	0	2	0.0028920
5	0	6	0.0028898
6	0	14	0.0028787
7	1	4	0.0028723
8	0	11	0.0028663
9	1	3	0.0028563
10	0	16	0.0028340
11	0	13	0.0028276
12	0	44	0.0028164
13	0	18	0.0028153
14	0	9	0.0028103
15	0	30	0.0028060
16	0	31	0.0027880
17	0	47	0.0027863
18	0	40	0.0027862
19	0	48	0.0027848
20	1	2	0.0027839
21	1	7	0.0027837
22	0	38	0.0027831
23	0	43	0.0027785
24	0	28	0.0027774
25	1	6	0.0027749
26	0	42	0.0027711
27	0	27	0.0027649
Continued on next page			

Table 21 – continued from previous page

Comparison No.	Cluster 1	Cluster 2	Distance
28	0	36	0.0027647
29	0	39	0.0027635
30	0	33	0.0027628
1195	44	45	0.0001930
1196	34	46	0.0001901
1197	20	31	0.0001894
1198	28	48	0.0001861
1199	32	43	0.0001853
1200	33	39	0.0001818
1201	41	47	0.0001812
1202	41	44	0.0001792
1203	20	24	0.0001783
1204	38	44	0.0001782
1205	18	31	0.0001782
1206	32	42	0.0001778
1207	19	24	0.0001766
1208	42	45	0.0001750
1209	35	40	0.0001715
1210	41	46	0.0001713
1211	28	47	0.0001682
1212	41	48	0.0001665
1213	9	14	0.0001621
1214	24	25	0.0001577
1215	43	45	0.0001441
1216	44	47	0.0001394
1217	40	42	0.0001366
1218	42	43	0.0001352
1219	18	30	0.0001304
1220	44	48	0.0001150
Continued on next page			

Table 21 – continued from previous page

Comparison No.	Cluster 1	Cluster 2	Distance
1221	36	37	0.0001078
1222	3	4	0.0001032
1223	6	7	0.0000954
1224	30	31	0.0000643
1225	47	48	0.0000394
Table 21 – concluded			

6 Discussion

The results covered the outcomes of each method in terms of concrete details. Now, I discuss the implications of using these approaches and the limitations of each.

6.1 PCA, RPCA, and Autoencoder Discussion

The strong correlation groupings exhibited by the original feature set for the CICIDS2017 dataset suggest that PCA for dimension reduction may be a useful approach. The classical approach on the global dataset yields acceptable dimension reduction results for the first five or ten dimensions, after which the improvement in amount of variation explained drops off considerably. The first 15 dimensions explain over 90% of the variation, and the first 20 dimensions explain nearly 97% of the variation. Each of the heuristic methods of thresholding the number of dimensions that were considered sets the cutoff within this range of dimensions. Depending upon one's level of comfort, these results imply that a substantial reduction in dimension can be made using classical PCA. The difficulty is that there is known structure that could account for at least five dimensions based on day, making such a reduction less meaningful. The choices discussed in Section 5.2 led to the addition of 160 embedding dimensions on the categorical variables of Protocol and Source and Destination IP Addresses and Ports first. Subsetting into five daily datasets was also done because exploratory work revealed much of the variation occurring in five easily-separated regions that were unlikely to be coincidental.

With the classical PCA approach, these changes resulted in roughly 9 to 12 principal component axes being retained based on the Broken Stick model. The aggressive cutoff was used because the goal of this project is to find an approach to dimension reduction that allows a large reduction in the dimensionality while also allowing potentially useful clusters of the malicious traffic. Retaining too many dimensions makes that task considerably more difficult, requiring too much exploratory work and computational power for the hypothetical security analyst who needs timely ways of detecting threats. One downside of this choice is that around 30% of the variation is lost by excluding the extra dimensions. If it is believed that malicious traffic may be tied to variance in the data, then excluding this much of the variation is ill-advised. It appears the differences between the malicious and benign traffic may not rely on variability, so this choice is at least empirically justified. On the other hand, the majority of the measures used, such as number of packets, minimum number of packets, and flow durations, could be expected to be small and around the same size. Typical packet transfers make such a heavy-tailed and right-skewed distribution of network traffic common[23]. A few large flows are not unusual, but even in a very high volume network we should expect the majority of the data to accumulate near the lower end of the distribution.

We might also expect malicious traffic to be designed specifically to blend in with benign traffic, since a major goal of malicious actors is penetration without detection. For more skilled attackers, such as those who carry out APTs and espionage campaigns, long-term persistent avoidance of detection is a real possibility. If these types of traffic are present, spread may not be a meaningful discriminator between the malicious and benign traffic types.

Several attack types stand out in visual inspection of the first PC in Figure 22. The Infiltration traffic is low frequency and extends across the range of this component, displaying little overlap with the benign traffic along this axis. The low frequency may be a characteristic of this type of attack, keeping the level of activity low enough that threat detection technologies are not triggered by volume. Heartbleed attacks, on the other hand, rely on making packet payload requests that are larger than allowed. The server promises to send the requested amount, but cannot do so with the requested payload, so the difference is made up by sending other information. For example, the server could send the header information of another user, which might include the username and password or other sensitive information¹⁶. It makes sense that this traffic would stand out from other data derived from PCAP files since it is an exploit that leans heavily on captured packet information. The other attacks that stand out are Denial of Service (DoS) and Distributed DoS (DDoS) attacks, which rely on overwhelming a server with more traffic than it can handle. As such, we would also expect the distribution of such attacks to be offset from standard benign traffic.

The remainder of the attacks are not as easy to distinguish from typical traffic in the first PC, although we can see that several of them have apparently multimodal distributions, whereas the benign traffic does not. These patterns are also common among the first several principal components, albeit on different scales, making it difficult to separate the attacks from the benign traffic visually, even when viewing in two or three dimensions. Since PCA relies heavily on the mean and variance, both of which are sensitive to skew and kurtosis, RPCA was a good consideration here. PCA relies on maximizing variance in each principal direction, but measures of centrality and spread in the presence of non-symmetric data are less informative, such as in this case of extreme skew. The hope was that a transformation that is robust to these issues might allow better separation of these groups.

The automated dimension-reduction cutoff for RPCA added only a few extra dimensions compared to the eigenvalue-based Broken Stick model, roughly corresponding to the elbow of the screeplot. The screeplot is a good illustration of why this elbow is a reasonable value for k for this data. Visually, only a small amount of additional variance is explained by adding any more principal components than RPCA retained. The average and scaled average eigenvalue cutoffs are much farther to the right on the curve, occurring after it has flattened. As mentioned in the results of this method, the choice of the same seed for each day may have

¹⁶See for example: <https://cloudshark.io/articles/packet-capture-of-heartbleed-in-action/>

Figure 22: Ridge Plot of the first principal component by traffic type.

Figure 23: Ridge Plot of the first RPCA component of the Tuesday dataset by traffic type.

caused comparable rotations for each day, so re-running with randomization might result in some differences, although this was not observed during this project. It is also possible that the day-to-day data is relatively similar, so comparable rotations are not unexpected. Other runs of RPCA with random seeds resulted in substantially similar outcomes in terms of values for k and for the eigenvalues, so the major principal axes appear to be located. However, some smaller directions of variation may be missed due to the limited number of projection directions used, the repeated use of the same seed, and the difficulty of dealing with small amounts of local variation.

Figure 24: Ridge Plot of the first RPCA component of the Wednesday dataset by traffic type.

In comparison to the global data, the eigenvalues appeared fairly similar from day to day, both with PCA and RPCA, especially those above the elbow of the screeplot. The smaller eigenvalues with RPCA may differ more from each other and from the PCA eigenvalues, or may not be included at all, based on the directions that are checked by the algorithm. In comparison to PCA, RPCA has more potential to generate interesting and useful insights in the presence of such skewed data. The initial set of outliers can be used as a good subspace for further analysis. For example, with the Tuesday dataset only 1 malicious observation was not flagged as an outlier, while less than 23% of the benign traffic was flagged as being outlying. This suggests that Patator-type traffic could be more easily detected with this type of algorithm than some other types of traffic may be. In contrast, in the Wednesday dataset only 5.3% of the malicious traffic was flagged overall, with only two of the four types of DoS attack being described as outliers. In particular, the DoS GoldenEye

(7.66%) and DoS Hulk (2.68%) attacks went virtually undetected as outliers, while the DoS Slowhttptest (63.79%) and DoS slowloris (50.52%) attacks were a more likely to be flagged. The extremely low number of observations (11) of the Heartbleed attacks make it unsurprising that none of them were flagged. The same applies to the SQL-Injection Web Attacks (21 observations) in the Thursday data, but not to the relatively few XSS Web Attacks and Infiltration attacks, of which 95% were flagged. Conversely, despite the large number of PortScan and DDoS observations in the Friday data, nearly 159000 and 128000 observations respectively, very few of them were flagged as outliers.

Several of these attack types were not easily separated from the bulk of the benign traffic during exploratory work and visual inspection, so not being flagged as outliers is not entirely surprising. The approach is promising because it results in only a few PCs, matching well with the number of PCs given by some of the non-robust cutoffs, and it suggests that the Hubert distance measure used to establish the initial h -subset early in the algorithm may be a worthwhile one to consider for malicious traffic detection in network data for some species of attacks. The robust measure of outlyingness also gives an extra evaluation measure that PCA does not provide. As can be seen in Figure 24, the traffic types that were not outlying also happen to lie close to the main group of benign observations in the Wednesday version of RPCA. While most of the data is multimodal, the two modes of the benign traffic are fairly spread out compared to the malicious traffic, so only one mode of the benign traffic overlaps with the malicious traffic. This is likely an artifact of the data generation process that would be dealt with by increasing granularity to hourly instead of daily data. An interesting additional direction that could enrich analysis of the h -subset would be to examine the types of outliers - good and bad leverage points and orthogonal outliers - to see whether there is a correspondence between malicious traffic types, outlier types, and inclusion in the h -subset. For example, the malicious Wednesday traffic might differ from the malicious Tuesday traffic in this sense.

Autoencoders add an interesting element to this discussion. The hidden layers of exploratory autoencoders run on the global dataset easily revealed structure that corresponded to the daily data, as shown in Figure 25, while the hidden layers of autoencoders run on the daily datasets revealed additional structure that could correspond to hourly data or to hosts. Some of the additional structure could correspond to hourly information, especially the stretched-out groups of observations and the parallel structures in the visualization shown in Figure 26. Some of the visible structure also appears to highlight traffic clusters, some of which is easily distinguishable as malicious in labelled data.

A single hidden layer was used with the autoencoder, but a deeper network might provide strong results. For example, in Figure 27, three hidden layers were used, revealing separation of most malicious observations from the main body of traffic. The stacks of thin parallel groups of data along the left side of the plot, along with streaks of malicious data throughout the middle and mixed with these parallel groupings, almost

Figure 25: First three dimensions of the central hidden layer of an autoencoder run on global dataset. Red observations correspond to benign traffic while other colours correspond to malicious traffic.

Figure 26: First two dimensions of the hidden layer of an autoencoder run on Wednesday dataset. Red observations correspond to benign traffic while other colours correspond to malicious traffic.

certainly reveals the hourly structure. There are ten non-continuous hours of data in the Wednesday data. The discontinuity in data generation likely explains the split between two groups of benign traffic and similarly between the two groups of malicious traffic. Referring back to Figure 12 for a visualization of the hidden layer of a single hour of the Friday dataset, additional structure becomes revealed, possibly associated with single users or host groups within the network. These results, while empirical, suggest that deeper granularity,

Figure 27: First two dimensions of the central hidden layer of a deep autoencoder with 3 hidden layers run on Wednesday dataset. Red observations correspond to benign traffic while other colours correspond to malicious traffic.

alongside a deep autoencoder, or possibly even a shallow autoencoder, could be enough to highlight some types of malicious traffic using clustering without further transformation. Most likely, the ideal approach would be a streaming algorithm updating at regular intervals throughout the day.

6.2 IP2Vec Embeddings

Up to this point little has been made of the category embeddings included in this data. In fact, the sample versions of the deep autoencoders shown directly above did not include the embeddings at all, but they continued to reveal useful patterns. Each of the main PCA, RPCA, and autoencoder results presented in Section 5 did include the embeddings, which may possibly have proven useful in revealing helpful subspaces. However, it is not clear that the embeddings were needed. Reasonable results were achieved earlier in the exploratory phases without the embeddings, and the dimension reduction with and without the embeddings ended up being similar. Given the additional computational time and effort of generating embeddings and attaching them to the data at each step, not to mention the significant investment of expertise needed to

validate them, it may not be worth it for timely security analysis of incoming network traffic in a streaming environment.

To make embeddings worthwhile, pre-generated embeddings for a network that have been validated by the network experts and that are automatically updated at the same intervals could still provide some benefit. Embeddings could be used to help identify new addresses or ports, or to highlight ports that should not be associated with a particular host. However, the usefulness of this technique seems limited, especially when security administrators already set up blacklists and flag unusual outgoing and incoming traffic patterns in many other ways.

It should also be acknowledged that the use of IP2Vec here is not the original intended use. Rather than for dimension reduction, the embeddings were used directly for clustering, with few or no additional features included. Such an approach may still be worthwhile separately or in conjunction with the remainder of this analysis, as one might inform the results of the other; however, further analysis of this possibility has not been pursued on the CICIDS2017 dataset.

Finally, analysis of which specific IP addresses are connected to malicious traffic has not been done here since it is a known problem with this dataset that certain IP addresses can be used to perfectly classify malicious traffic. However, the possibility exists in real-world data that embeddings could be used to help identify compromised groups of machines, especially in the context of a directed graphical model, so the approach is not entirely without merit.

6.3 ICA and Clustering Discussion

Using Icaasso for multiple random starting points for the ICA algorithm appeared to be a successful approach. Low quality components were easy to spot in the dendrogram and in the MDS plots. The low number of components due to the previous dimension reduction allowed easy selection of a final number of components based on visual counting, but larger data would not allow this approach. The use of the quality index in that case would be essential.

One question that might arise at this stage is whether ICA needs to be used at all. A second round of transformation may seem excessive, but we must recall again that PCA is strongly tied to the spread of the data. Independence, rather than decorrelation, might allow better separation between the malicious and benign traffic. After all, it is expected that malicious traffic comes from a different source at a fundamental level. Regardless of how well the malicious activities are obfuscated, it must display some difference in its characteristics than traffic driven by the unobtrusive activities of people who are simply going about their daily tasks. The features included in this dataset may not be the ideal ones to reveal these characteristics to

us, but it is worthwhile to check.

Figure 28: First three PCs from PCA on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Grey observations reflect other clusters. Black observations reflect outliers not contained within a cluster.

Figure 29: First three PCs from RPCA on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Grey observations reflect other clusters. Black observations reflect outliers not contained within a cluster.

The output of the three versions of dimension reduction allows density-based clustering to be carried out. However, the clusters would not as easily be separated from the rest of the traffic because each of these methods has not separated the malicious traffic in terms of distance. In particular, in Figures 28 and 29, this can be seen in the PCA and RPCA output data. The malicious traffic is all grouped together, but is not greatly separated from the benign traffic. Rotating around the first three principal component axes allows one to find directions that might introduce some distance between them, such that a clustering routine might count them as separate cluster, but it would be difficult to identify these clusters in particular as being problematic or of

Figure 30: First three dimensions of the hidden layer of the autoencoder on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Grey observations reflect other clusters. Black observations reflect outliers not contained within a cluster.

interest compared to the many other such groupings. The autoencoder version, shown in Figure 30, we can more easily see some separation from the main cloud of observations, however small. That said, ICA was hoped to add additional distance.

Proceeding, then, with ICA, on the three versions of the Tuesday data, it is immediately clear that PCA methods were successful in reducing the dimension while retaining useful ones. This does not mean that additional useful sources were not excluded, but that the sources that were retained are stable enough to use. In clustering the sources based on the hierarchical cutoff, some error is introduced that means the axes are not strictly independent. However, they should stay close enough to independence that they will not cause problems later. What would likely be a better alternative, and what was probably intended by the original authors, is to use a lower cutoff that separates the lowest quality components into separate clusters, then to eliminate those clusters entirely and retain only the most stable components.

After separating the averaged sources, it remains only to cluster them. The density-based approach is sensible based on many of the properties of this data observed so far. The benefit of using HDBSCAN* is that its implementation allows a much more dynamic approach than simply using DBSCAN or HDBSCAN. The connectivity and speed improvements of HDBSCAN can be combined with the ϵ -neighbourhood of DBSCAN, but in a way that allows it to be used locally and a single large cluster to form if the connectivity can be found within the neighbourhoods. A good heuristic for the size of ϵ was to use $1 - 2\%$ of the smallest IQR among the dimensions retained in the previous stage. Using a proportionally large ϵ on the *PCA* data

resulted in three clusters, one of which was a single large cluster of 94% of the benign observations. Both of the malicious groups of data were contained in a single cluster of which 80% was malicious observations. This cluster can be seen in Figure 31 on the right side. The malicious red and green coloured observations ended up clustered with the nearby black shaded observations due to the large epsilon. This is not necessarily objectionable. In a situation with unknown class labels, if malicious traffic can be expected to be split off by using ICA then a cluster of observations that is relatively far away from the main cluster, as can be seen in Figure 31, might be a starting point for investigation.

Figure 31: First three PCA-ICA components on the Tuesday dataset. Red and green observations reflect malicious traffic clusters. Black observations reflect all other clusters and outliers.

Another option, which was explored in the RPCA and autoencoder versions of the clustering, is to use a smaller ϵ as described above. This resulted in 48 and 50 clusters respectively. In each case, 4 total clusters represented 99% of all malicious traffic alone. In all attempts, 1 benign observation always ends up merged with the malicious clusters. In addition, approximately 20 to 40 malicious observations are marked as outliers. These are easy to locate visually because they are within range of the malicious clusters. The other outliers, however, are less easy. One of the two largest groups of observations, in terms of total number of observations, in each of these two clusterings is the outliers. That may not be a problem here, but in general it increases the difficulty of knowing which observations to investigate. Couple that with the reality that the visualizations presented so far are cherry-picked from among many combinations, shown below, and these visualizations are not necessarily a particularly strong way of identifying malicious traffic.

Without ICA, the groups of observations that appear to be separated from the main cloud in some dimensions do not include any of the malicious traffic. Interestingly, the difference between the three dimension-reduction methods may not be large enough to matter. Furthermore, choosing which dimensions and which rotations within those dimensions to use to visually identify separation between potential groups is almost entirely subjective. Principal Component Analysis and the autoencoder alone both result in similar shapes,

Figure 32: Ordered distance between cluster centroids from the PCA-ICA clustering in which three clusters were found. Red points represent distance from malicious cluster centroids to other cluster centroids. Grey points represent distance between benign cluster centroids. Black points represent distance to the outliers centroid.

though the autoencoder outcome seems to be more tightly bunched and the groupings are less noisy than the PCA outcome overall. The RPCA outcome appears to induce more tightly bunched groupings with more noise than the autoencoder but possibly less than PCA. In any case, since PCA-type transformations lean into the natural variation in the data, lack of visible separation in the first three dimensions, alongside less tangible distance between clusters, suggests that ICA could be an essential step here.

We would like a way to measure whether the group of clusters that contains the malicious observations is farther away from the main cloud of observations, alongside nearby local outliers based on connectivity and a handful of other dense areas that may be counted as clusters depending upon the parameters used. By definition, the outliers are far away from any of the clusters locally. Globally, at least on this data and with the transformations carried out, the transformations seem to move the malicious traffic farthest away from the majority of the rest of the data. This is demonstrated in Figure 32, which shows the case when a single large cluster dominates the space. The malicious clusters are farthest from every non-malicious cluster, while the benign clusters are nearest to each other. The distance to the centroid of the outliers is tied directly to the other distances since each cluster is almost certain to have some outliers, making the centroid of the outliers representative of all clusters at once and so fairly meaningless in this context.

Visually, a distance plot is appealing, but the actual separation can only be noticed in selected 3-dimensional rotations in chosen dimensions, not necessarily in the first three component axes and not in a well-established pattern. In part, this is because ICA does not result in ordered components, and in part it is because the mean vectors are at least 8-dimensional, so the 3-dimensional visualization of the distance is more of an illustration than a ground truth. In the RPCA version, 19 of the 56 combinations of 3-dimensions out of the 8 dimensions

Figure 33: RPCA-ICA cluster distances. Red points represent distances from malicious cluster centroids to other cluster centroids. Green points represent distances from clusters with centroids nearest to malicious centroids to other clusters. Grey points represent distances between benign cluster centroids.

illustrated this clear pattern of separation, while most of the other combinations showed different groups of points farther away from the main cloud, or only had a single massive point cloud. Often, the malicious clusters were not visible, much less distinguishable.

The more extensive version of the pairwise distances shown in Figure 33 illustrates what the distances looks like when the ϵ -neighbourhood is shrunk. The index is not important. The outlier centroid has been removed from the plot, and pairwise distances between malicious centroids have also been removed. Hence, this plot illustrates only distances between malicious centroids and benign ones and between mutually benign centroids. The green points identify distances between benign centroids and the two centroids closest to the malicious centroids.

The largest pairwise distances include the malicious clusters and the distance from any cluster on one side of the space to a cluster on the other. These distances can be seen as the separate group at the top left corner of Figure 33. The distances from the malicious clusters to each benign cluster can be seen in the red groups, with the two separate bunches representing the four clusters of malicious traffic, which fall into two larger groups by attack type. The green group represents the distance from the two centroids closest to the malicious clusters to each of the benign clusters. The distance from the first of the two nearby clusters to the malicious clusters can be seen in the red points at the bottom left of the plot, while the distance from the

second of the nearby clusters can be seen slightly below the groups of red points.

As should be clear, the distances may be meaningful, but there is no clear way to say which distances are associated with a group of malicious observations. We might consider testing the difference, but, as Gao, Bien, and Witten highlighted recently[22], this should be considered double-dipping. The clusters are generated expressly to identify a difference. Testing this difference is, in essence, testing whether we located differences when locating differences - circular and ultimately meaningless.

6.3.1 A Limitation

We are left, then, with an understanding that by removing correlation and dependence among the variables and reducing the dimension by eliminating combinations that generate little variation, larger distances between dense areas of the data may, in some way, be identifiable as of interest to a security analyst. Whether or not it may be associated with malicious traffic requires considerably more work, and likely additional data to help highlight concrete differences between the activities taking place.

Importantly, binary variables were excluded in this thesis. Early exploratory work included them, however, the majority of the data needed to be normalized before proceeding with PCA. Normalizing a dichotomous {0, 1} variable might have been fine or it might not have been. I did not have the opportunity to explore the issue, and rather than blindly include un-transformed binary variables or include blindly-transformed binary variables, I blindly removed them instead. As can be seen throughout the thesis, the results still provide interesting insights, but the exclusion of a large number of variables without any analysis may damage those insights to a certain extent. It certainly leads to complications in the next stage of this project as we consider the real-world dataset from Rank.

6.4 The Rank Data: Putting it all Together

All of the exploratory work to this point provides a foundation upon which to try this method on real-world data. The Rank data is unlabelled data, so it will be difficult to determine whether the clusters are meaningful, but that is an expectation that has been discussed all along. Rank provided 19708 individual data files representing what most likely coincides with some standard *JSON* file length. These were merged into 68 data files that correspond to a few days of data. They may be further merged into a single large dataset of roughly 13.4GB, but the synthetic data until now has suggested trying a narrower timeframe, so a single one of these 68 files is used. The first ranges from August 31, 2019 at 8:00pm to September 3, 2019 at 5:11pm and has 791581 observations. A matching set of Alerts data was located based on timeframe. However, there was no identifying feature that could be used to link the Alerts to the Events. The Alerts data had nearly 1500

potential variables upon which to match, most of them empty, as discussed in Section 3.1.2. Matching on date-time was tested and it successfully captured 24 events that had matching timestamps between the two datasets. Since that represented only 0.003% of all of the data in this dataset, the Alerts data was deemed to be too uninformative for this analysis and was thereafter ignored, despite the extensive work that went into validating it. One possible solution might be a fuzzy matching scheme using date-time range or a sliding window.

Among the 33 variables in the Events dataset after two that contained duplicate information were removed, there were few that resembled the PCAP statistics used in the previous sections. Several categorical variables associated with packet headers were available, which included the Source and Destination IP addresses and Ports and the Protocol. Numerical variables included the number of Requested and Response Bytes, IP Bytes, and Packets, the Connection Duration, and the Source and Destination Autonomous System Numbers (ASN). However, other available variables were either strictly categorical or were binary {0, 1}. The binary variables included flags on Internal Sources and Destinations and on Status. The remainder were categorical, but each was dominated by the empty category and was otherwise sparsely populated. The summary statistics for the raw data shows that again the data has large skew and kurtosis values except in cases of binary variables such as “Source Internal,” which has negative skew, positive kurtosis, and is most values of 1 because much of the traffic is internal traffic. None of this is out of line with the synthetic data used so far, with the main difference being the inclusion of the binary variables.

After initial processing, the next step was to generate embeddings on the major categorical variables, such as the IP addresses. An alternative approach not pursued in this thesis might be to run a mixed version of PCA and multiple correspondence analysis (MCA) on the entire thing, although this would likely not be possible on the IP addresses and ports, with tens of thousands of categories at minimum. Instead of using MCA, the categorical variables were used to generate numerical embeddings, while the binary variables were retained in the numeric dataset. Hence, over half of the data was used in the embeddings, with 17 variables total. The window for IP2Vec was the entire “sentence” for each event, or all 17 “words.” Due to the small number of categories in several of the categorical variables, this may not have been the best approach. A larger number of variables would generate more useful continuous numerical representations during the embedding process. It might have been better to exclude the variables with only a few categories entirely than to use them, but it is hoped that their inclusion enriched the embeddings in some way.

Table 22: Summary Statistics: Rank Data

Variable	Mean	Variance	Median	MAD	Skewness	Kurtosis
Resp Bytes	36185	633e+11	0	0	768	640162
Req Bytes	10184	9.86e+10	156	188	643	475302
Duration	2.73e+05	3.98e+15	417849	619491	317	159480
Source ASN	3455.02	2.04e+06	0	0	6	57
Destination ASN	13729	2.99e+07	0	0	6	45
Bytes	46370	7.34e+11	224	332	647	485032
Req Packets	37	14673902	6	5	392	177845
Resp Packets	48	73919528	0	0	669	518125
Req IP Bytes	15984	1.77e+11	366	428	495	289193
Resp IP Bytes	51834	1.51e+12	0	0	738.93	599370
Observed Protocol Has Known Port	0.73	0.20	1.00	0	-1	2
Source Internal	0.91	0.08	1	0	-2	8
Destination In- ternal	0.61	0.24	1	0	-0.46	1
Status	0.48	0.25	0	0	0.07	1
Table 22 – concluded						

Classical PCA with classical normalization was used to determine the dimension-reduction to be used in ICA, since it was concluded earlier that each of the dimension-reduction methods explored on the synthetic data resulted in similar eventual clustering outcomes. Given the typical skewness and kurtosis that is displayed in Table 6.4, Robust PCA might be better as a general choice for data of this type. However, classical PCA was used here due to computing power limitations and the limitations described in Section 7.

Including all 17 variable embeddings of 32 dimensions each caused the Carleton virtual machine to fail during SVD. A parallel version of SVD or an on-disk rather than in-memory implementation of PCA such as *big.PCA* from the *bigpca* package - although something faster would be better - is highly recommended

for future work in this area. However, part of the problem may have been stability issues introduced by using the categorical variables with only a handful of categories to generate embeddings, as these could have failed to generate meaningful embeddings and introduced empty results instead. Reducing the embeddings to the 7 that corresponded to ports, protocols, and IP addresses failed for that reason. The embeddings that came from ports induced extensive NAs during transformation, corresponding to more than 60% of the data being empty in the corresponding embedding vectors. Eventually, 64 embedding dimensions, corresponding to Source and Destination IP addresses, were the only ones included. The dataset was reduced to only 6 component dimensions using the Broken Stick model on the eigenvalues. The average eigenvalue set the cutoff at 19 components, while the scaled average set the cutoff at 22 components.

Table 23: Results of Classical PCA on the first Rank Dataset

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
1	12.11	15.53	15.53	6.33
2	8.49	10.89	26.42	5.05
3	6.55	8.40	34.82	4.41
4	5.12	6.57	41.39	3.98
5	4.41	5.65	47.04	3.66
6	4.29	5.49	52.53	3.41
7	2.86	3.66	56.20	3.19
8	2.67	3.43	59.63	3.01
9	2.60	3.33	62.95	2.85
10	2.47	3.16	66.12	2.71
11	1.97	2.53	68.65	2.58
12	1.93	2.47	71.12	2.46
13	1.53	1.96	73.07	2.35
14	1.42	1.81	74.89	2.26
15	1.34	1.72	76.61	2.16
16	1.20	1.54	78.15	2.08
17	1.16	1.49	79.64	2.00
18	1.09	1.40	81.05	1.92
Continued on next page				

Table 23 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
19	1.00	1.28	82.33	1.85
20	0.97	1.25	83.58	1.78
21	0.93	1.19	84.77	1.72
22	0.72	0.92	85.70	1.66
23	0.70	0.90	86.59	1.60
24	0.68	0.87	87.46	1.55
25	0.66	0.85	88.31	1.49
26	0.63	0.81	89.12	1.44
27	0.58	0.75	89.86	1.39
28	0.56	0.71	90.58	1.34
29	0.50	0.64	91.22	1.30
30	0.45	0.57	91.80	1.25
31	0.42	0.54	92.34	1.21
32	0.39	0.50	92.84	1.17
33	0.38	0.49	93.33	1.13
34	0.35	0.45	93.78	1.09
35	0.32	0.41	94.19	1.05
36	0.30	0.38	94.58	1.02
37	0.28	0.36	94.94	0.98
38	0.27	0.35	95.29	0.95
39	0.26	0.33	95.62	0.91
40	0.25	0.32	95.93	0.88
41	0.23	0.29	96.22	0.85
42	0.20	0.26	96.49	0.82
43	0.19	0.24	96.72	0.79
44	0.18	0.23	96.95	0.76
45	0.17	0.22	97.18	0.73
46	0.17	0.22	97.39	0.70
Continued on next page				

Table 23 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
47	0.15	0.20	97.59	0.67
48	0.14	0.18	97.78	0.64
49	0.14	0.18	97.95	0.62
50	0.13	0.17	98.12	0.59
51	0.12	0.16	98.28	0.57
52	0.12	0.15	98.44	0.54
53	0.12	0.15	98.58	0.52
54	0.11	0.14	98.72	0.49
55	0.10	0.13	98.85	0.47
56	0.09	0.12	98.97	0.44
57	0.09	0.12	99.09	0.42
58	0.08	0.10	99.19	0.40
59	0.07	0.09	99.28	0.38
60	0.06	0.08	99.37	0.35
61	0.06	0.08	99.45	0.33
62	0.06	0.07	99.52	0.31
63	0.05	0.06	99.59	0.29
64	0.05	0.06	99.65	0.27
65	0.04	0.05	99.70	0.25
66	0.04	0.05	99.75	0.23
67	0.03	0.04	99.79	0.21
68	0.03	0.04	99.83	0.19
69	0.03	0.04	99.86	0.17
70	0.03	0.03	99.90	0.16
71	0.02	0.03	99.93	0.14
72	0.02	0.03	99.96	0.12
73	0.02	0.02	99.98	0.10
74	0.01	0.01	99.99	0.08
Continued on next page				

Table 23 – continued from previous page

Component	Eigenvalue	Explained Variation %	Cumulative Variation	Broken Stick %
75	0.01	0.01	100.00	0.07
76	0.00	0.00	100.00	0.05
77	0.00	0.00	100.00	0.03
78	0.00	0.00	100.00	0.02

Table 23 – concluded

The transformed data was then passed through Icaasso as before. There were 5 independent components at a cutoff of 0.8 that appeared to be fairly stable on the dendrogram in Figure 34, with one additional unstable component that splits into two unstable branches slightly lower. A cutoff of 0.6 could increase the number of components to roughly nine to eleven, but the extra components may be less stable and possibly less meaningful. The MDS plot was similar, with six to eight clusters of components. However, several of them are only loosely clustered, indicating they may not be as stable as we might prefer. A dendrogram cutoff of 0.88, as used in the previous work, would generate between two and three components instead. That might be preferable, as described below, based on the resulting shape after the independent components are generated, but keeping five or six components is a safer bet given what structure is apparent up to this point.

Figure 34: Dendrogram of the clustered independent components of the example Rank dataset.

Selecting the 5 components and performing density-based clustering using HDBSCAN with an ϵ -neighbourhood

Figure 35: MDS plot of the clustered independent components of the example Rank dataset.

based on 1% of the minimum univariate IQR of the averaged independent components, a minimum number of samples of 5, and a minimum cluster size of 100, with the possibility of a single cluster being generated, yielded roughly 1200 clusters. Using an ϵ -neighbourhood of 2% of the maximum IQR and a minimum cluster size of 1000 resulted in around 900 clusters. Increasing the ϵ -neighbourhood to 10% of the maximum IQR decreased the number of clusters found to 162, but increasing the ϵ -neighbourhood to be roughly equivalent to the maximum univariate IQR only reduced this number to 152. Setting the minimum number of samples to 15 for the same ϵ -neighbourhood reduced this to 62 clusters. A larger minimum number of samples, such as 25 or 50, or a larger ϵ -neighbourhood, or both, yielded a single large cluster. A single large point cloud dominates most of the visual depictions of the averaged independent component space, so a single large cluster seems like a possibility, although any depiction of three dimensions that includes the first IC that was generated yields a nearly perfectly flat two-dimensional surface with a few noise points in the third dimension. It is possible that 3 or 4 components might have adequately described this data. Four of the components can be visualized together in three dimensions to yield spread in all three dimensions, but the density of this cloud may not be high enough for it to be spanned easily. There may be space between groups of observations in some dimensions that do not adequately connect the densities together with those in other dimensions.

Table 24: Pairwise Distances Between Clusters

Comparison No.	Cluster 1	Cluster 2	Distance
1	75	122	0.00158
Continued on next page			

Table 24 – continued from previous page

Comparison No.	Cluster 1	Cluster 2	Distance
2	74	122	0.00158
3	75	94	0.00158
4	75	131	0.00158
5	75	95	0.00158
6	74	94	0.00157
7	74	95	0.00157
8	74	131	0.00157
9	75	132	0.00157
10	75	154	0.00157
11	74	132	0.00156
12	75	96	0.00156
13	75	98	0.00156
14	74	96	0.00156
15	55	154	0.00156
16	75	97	0.00156
17	74	98	0.00156
18	74	97	0.00155
19	75	101	0.00155
20	75	100	0.00155
21	75	140	0.00155
22	74	101	0.00155
23	74	100	0.00155
24	75	124	0.00155
25	74	140	0.00155
26	75	99	0.00155
27	74	154	0.00154
28	74	99	0.00154
29	74	124	0.00154
30	75	139	0.00154
Continued on next page			

Table 24 – continued from previous page

Comparison No.	Cluster 1	Cluster 2	Distance
12531	97	98	0.00001
12532	96	100	0.00001
12533	21	23	0.00001
12534	21	24	0.00001
12535	22	23	0.00001
12536	27	28	0.00001
12537	22	24	0.00001
12538	1	2	0.00001
12539	99	101	0.00001
12540	2	3	0.00001
12541	99	100	0.00001
12542	29	30	0.00001
12543	29	31	0.00001
12544	100	101	0.00000490
12545	30	31	0.00000327
12546	25	26	0.00000047
12547	21	22	0.00000044
12548	53	54	0.00000038
12549	36	37	0.00000031
12550	49	50	0.00000029
12551	19	20	0.00000026
12552	23	24	0.00000026
12553	43	44	0.00000007
12554	6	8	0.00000004
12555	16	17	0.00000003
12556	6	9	0.00000003
12557	6	7	0.00000003
12558	7	9	0.00000002
12559	77	78	0.00000002
Continued on next page			

Table 24 – continued from previous page

Comparison No.	Cluster 1	Cluster 2	Distance
12560	7	8	0.00000002
12561	8	9	0.00000002
Table 24 – concluded			

That separation can be observed at a high enough magnification on Figure 36 or any similar plot, which displays three of the clustered independent components that have subsequently been clustered using HDB-SCAN. The density of points in the main cloud of observations is not enough to generate connectedness between the clusters there unless a fairly large neighbourhood is used. The large number of outliers that make up a portion of the cloud emphasizes the gaps. There are also small linear structures within the cloud that may correspond to traffic related to a particular host or group of hosts, to time-related structure, or possibly to something related to the binary or categorical variables included.

As could be seen with the synthetic data, a large point cloud dominates the space and may be used as a basis for further exploration. The largest distances between clusters are between the clusters in the two clumps that make up the main body of observations and a few small clusters of points at the far end of the space. There is no implicit meaning in this comparison, as it is a simple distance between centroids of clusters, but in the presence of potential malicious traffic such measurements might help indicates unusual activities that could correspond to something more malicious, or to a vulnerability in the network. Alternatively, when a model of the network traffic is being used, this more distant traffic might be used to help define the true range of observations that might occur if the distant traffic is known to be benign. Defining such range might strengthen the model's ability to detect true deviations that had not previously been accounted for.

Figure 36: Plot of three of the clustered independent components of the example Rank dataset after HDBSCAN is run. The red observations are the three clusters farthest from any other cluster. The green clusters are the nine clusters that were farthest from the three red clusters. Black observations are outliers not included in any cluster.

7 Conclusions, Limitations, and Future Directions

Most of this thesis was exploratory in nature and relied heavily on visual inspection. Work on the CIDS2017 dataset allowed a more guided approach to exploration that was intended to reveal useful methods for exploratory analysis of real world network traffic data. The combination of PCA methods with randomized iterations of ICA and then density-based clustering using HDBSCAN typically separated some portion of the data. Whether the induced distance is meaningful has not been proven, but is a potential future direction to take this work.

This thesis could be extended in numerous ways, many of which have been mentioned throughout the text. What is probably the most important one is that the data was analyzed as though it were static. Treating this type of data as time series, or using streaming or online methods, would be more appropriate given that

network traffic patterns naturally change over time. Online or time series methods may therefore be better suited to generating meaningful insights. There has been work on streaming versions of PCA - see, for example, [55] - that might allow a good approximation of SVD at each timepoint. An online adaptation of ICA was also proposed two decades ago[77]. While streaming algorithms for some density-based clustering exist (e.g. [84]), in 2019 Campello et al. listed this as an area for further efforts[11]. One might speculate that if time were treated as a local dimension then density-connectedness could be used to generate clusters in time. Connecting these clusters to other dimensions in a similar way to subspace clustering, could possibly be used to connect periodic behaviour. It is easy to guess that this would be computationally expensive, but a hierarchical and adaptive approach such as that used by HDBSCAN* seems like a reasonable first step.

In the use of PCA, one major deficiency was the use of primarily heuristic approaches to choose an appropriate subspace. It would be useful to add one or more of the validation methods described by Forkman and colleagues[20] and discussed in Section 4.1.1. In particular, a large-sample version of the bootstrapped hypothesis testing described by the same authors would have been useful in determining a more theoretically justified cutoff. Additionally, validation of a standardization method for this data would have been appropriate. Although the robust standardization used introduced problems, that does not necessarily invalidate the application of robust standardization. There are alternatives that could have been tried in order to stabilize the data. Given the apparent success of RPCA, despite it not being a clear “winner” over the other methods, it would be useful to try to improve the centering or scaling to see whether it leads to a clearer outcome during source separation or clustering. RPCA could also have been improved similarly to the use of Icasso by carrying out multiple runs with randomized starting points. An alternative would be to introduce guided pursuit that would possibly allow the number of pursuit directions to be reduced while also perhaps increasing its success in locating the directions of greatest variation.

With ICA, one aspect that was ignored in the use of Icasso was the addition of the bootstrap to further validate the components. The number of iterations of *FastICA* was also fairly low at only thirty runs. Additional runs would help clarify the stability or instability of each component, allowing easier clustering and selection of the stable components, though it would be at the cost of additional computational power that might damage the usefulness. The linearity of the data was also not explored, and there is no guarantee that linear PCA and ICA were appropriate. In addition, ICA was not explored on its own, separate from the use of PCA. This is primarily because dimension reduction was considered desirable, and ICA without PCA assumes the same number of sources as there are dimensions. However, a preliminary step might be to run many rounds of Icasso on a standard set of traffic data from a particular network to determine roughly how many components make sense in the context rather than relying on exploratory PCA first. That number could be updated periodically to capture changes in the network environment, such as the addition or subtraction of

users or other host machines, which could inform a static or a streaming model being used to detect threats.

The IP2Vec embeddings could also be improved. For one thing, there is redundancy introduced in appending 32 dimensions for each variable that was embedded. The embeddings must be correlated since the window includes all of the variables being embedded in all cases. Thus, the embedding for a Source IP Address is generated based on the Destination IP Address and associated Ports and Protocols, while the embedding for a Destination IP Address is generated based on the Source IP Address and the associated Ports and Protocols. Such correlations are not necessarily problematic. However, the window around a vocabulary term remained static, and the word order remained static, so the embeddings rely on column order. If the Source IP address is the first column, the Destination Port is the second last column, and the Protocol is the last column prior to tokenization, then the Protocol is always the farthest word from the Source IP Address and the Destination Port is always the closest word to the Protocol. This likely decreases the effectiveness of any interpretation of the embeddings that relies on context, as is the case with the IP2Vec Skip-gram model.

The exclusion of the binary variables during the CICIDS2017 portion of the thesis, and the problems associated with the categorical variables during the Rank portion of the thesis, require additional work as well. Binary variables may be used in PCA and in an autoencoder since they are numerical, but they are not continuous. They were excluded at first because of that. Standardization changes them to proportions and it was not clear this would be a meaningful transformation. Categorical variables could be addressed using MCA instead of PCA. As mentioned above, mixed versions of PCA and MCA exist and could have been utilized here. It is possible that MCA could be used on the binary variables as well.

There are, of course, other deficiencies that were not addressed, such as optimization of the hyperparameters in Word2Vec, the autoencoder, and HDBSCAN. Even simple feature selection at the beginning could have been used as a method of dimension reduction or to improve the quality of the work thereafter. It is likely that the features included in these datasets could have been improved considerably with the particular approach in mind. Other approaches may therefore yield better results on these particular datasets. For example, a directed Bayesian probabilistic graphical model was one possible direction that was considered during the earlier stages of the thesis, and was pursued elsewhere and discussed briefly in [10].

Another deficiency that I was able to ignore because in the end I worked with the Tuesday data alone has to do with DoS and DDoS attacks. The Wednesday data, which was less cooperative when trying to isolate the attack data in the early exploratory work, has only DoS and DDoS attacks of several types. Using complete network flows after the fact with such attacks would be meaningless because they are meant to flood a network and prevent it from working. Prevention must therefore be timely. The approach used in this thesis would not be well-suited to detecting and preventing these attacks. Extraction of some specific characteristic of DoS and DDoS attacks in the early stages of network flows would perhaps be better suited to detection and

prevention in this case, and an updating streaming approach rather than a static one is required.

Overall, some combination of the steps used in this thesis may prove beneficial in exploratory analysis of network traffic to help identify observations that should be flagged for further analysis. As mentioned throughout the thesis, without attack labels it is difficult to use clustering approaches as more of a guide. However, if similar features were used alongside a large group of alerts, it might prove helpful to use this methodology to separate traffic that is more likely to be benign from traffic that stands out in some way. If this approach were to prove successful it could greatly reduce the overwhelming number of alerts generated by a modern IDS, and thus help moderate alert fatigue.

Appendix A: Data Schemata

This appendix outlines the general schemata found in the Rank Software data.

Alerts Schema

```
1 {"index",
2   "type",
3   "id",
4   "score",
5   "source":{
6     "type",
7     "data":{
8       "rank_alert":{
9         "created",
10        "modified",
11        "triggered",
12        "partition",
13        "publishPartition",
14        "score",
15        "name",
16        "category",
17        "description",
18        "message",
19        "structuredMessage": "list of 37082 entries",
20        "killchain": "list of 37082 entries",
21        "additionalDetails":{
22          "Anomaly Date",
23          "AnomalyEventId",
24          "AnomalyReason":{
25            "anomaly_method",
26            "anomaly_source",
27            "anomaly_score",
28            "id",
```



```

29         "anomaly_assets",
30         "anomaly_killchain",
31         "anomaly_impact"
32     }
33 },
34     "justification": "list of 37082 entries",
35     "interesting": "data.frame, up to 300 variables",
36     "triggerName",
37     "closed",
38     "alertOpenDuration"
39 }
40 }
41 "date",
42 "origin",
43 "correlationIds": "vector of IDs",
44 "alertState":{
45     "status",
46     "suppressionInterval:{
47         "from",
48         "to",
49         "window",
50         "mode"
51     },
52     "assignee"
53 },
54 "id",
55 "feedbackData":{
56     "history":{
57         "default",
58         "open",
59         "tag",
60         "closed",

```

```

61         "comment"
62     },
63     "tags"
64 },
65     "source":{
66         "ip",
67         "ipHighBits",
68         "ipLowBits"
69     }
70 }
71 }

```

Events Schema

```

1  {"datetime",
2   "source ip",
3   "source port",
4   "source assets id",
5   "source name",
6   "source internal",
7   "source routingMode",
8   "source reputation",
9   "destination ip",
10  "destination port",
11  "destination assets id",
12  "destination name",
13  "destination internal",
14  "destination routingMode",
15  "destination reputation",
16  "protocol service",
17  "protocol udpOrTCP",
18  "data conn respBytes",

```

```

19 "data conn reqBytes",
20 "data conn duration",
21 "data conn state",
22 "data conn status",
23 "data conn history",
24 "source asn",
25 "source country",
26 "destination asn",
27 "destination country",
28 "data conn bytes",
29 "data conn reqPackets",
30 "data conn respPackets",
31 "data conn reqIPBytes",
32 "data conn respIPBytes",
33 "data conn info observedProtocol name",
34 "data conn info observedProtocol hasKnownPort",
35 "data conn info observedProtocol portProtocol"
36 }

```

CICIDS2017 Schema

```

1 {"Flow ID",
2  "source IP",
3  "source port",
4  "destination IP",
5  "destination port",
6  "Protocol",
7  "Timestamp",
8  "Flow Duration",
9  "Total Fwd Packets",
10 "Total Backward Packets",
11 "Total Length of Fwd Packets",

```

12 "Total Length of Bwd Packets",
13 "Fwd Packet Length Max",
14 "Fwd Packet Length Min",
15 "Fwd Packet Length Mean",
16 "Fwd Packet Length Std",
17 "Bwd Packet Length Max",
18 "Bwd Packet Length Min",
19 "Bwd Packet Length Mean",
20 "Bwd Packet Length Std",
21 "Flow Bytes/s",
22 "Flow Packets/s",
23 "Flow IAT Mean",
24 "Flow IAT Std",
25 "Flow IAT Max",
26 "Flow IAT Min",
27 "Fwd IAT Total",
28 "Fwd IAT Mean",
29 "Fwd IAT Std",
30 "Fwd IAT Max",
31 "Fwd IAT Min",
32 "Bwd IAT Total",
33 "Bwd IAT Mean",
34 "Bwd IAT Std",
35 "Bwd IAT Max",
36 "Bwd IAT Min",
37 "Fwd PSH Flags",
38 "Bwd PSH Flags",
39 "Fwd URG Flags",
40 "Bwd URG Flags",
41 "Bwd Header Length",
42 "Fwd Header Length",
43 "Fwd Packets/s",

44 "Bwd Packets/s",
45 "Min Packet Length",
46 "Max Packet Length",
47 "Packet Length Mean",
48 "Packet Length Std",
49 "Packet Length Variance",
50 "FIN Flag Count",
51 "SYN Flag Count",
52 "RST Flag Count",
53 "PSH Flag Count",
54 "ACK Flag Count",
55 "URG Flag Count",
56 "CWE Flag Count",
57 "ECE Flag Count",
58 "Down/Up Ratio",
59 "Average Packet Size",
60 "Avg Fwd Segment Size",
61 "Avg Bwd Segment Size",
62 "Fwd Avg Bytes/Bulk",
63 "Fwd Avg Packets/Bulk",
64 "fwd Avg Bulk Rate",
65 "Bwd Avg Bytes/Bulk",
66 "Bwd Avg Packets/Bulk",
67 "Bwd Avg Bulk Rate",
68 "Subflow Fwd Packets",
69 "Subflow Fwd Bytes",
70 "Subflow Bwd Packets",
71 Subflow Bwd Bytes",
72 "Init Win bytes forward",
73 "Init Win bytes backward",
74 "act data pkt fwd",
75 "min seg size foward",

```
76 "Active Mean",  
77 "Active Std",  
78 "Active Max",  
79 "Active Min",  
80 "Idle Mean",  
81 "Idle Std",  
82 "Idle Max",  
83 "Idle Min",  
84 "Label"  
85 }
```

References

- [1] R. Abdulhammed, H. Musaffer, A. Alessa, M. Faezipour, and A. Abuzneid. Features dimensionality reduction approaches for machine learning based network intrusion detection. *Electronics*, 8, 2019. doi: 10.3390/electronics8030322.
- [2] M. H. Abdulraheem and N. B. Ibraheem. A detailed analysis of new intrusion detection dataset. *Journal of Theoretical and Applied Information Technology*, 97(17), 2019.
- [3] C. Aggarwal and C. Reddy. *Data Clustering Algorithms and Applications*. 2013.
- [4] A. Ahmim, L. Maglaras, M. A. Ferrag, M. Derdour, and H. Janicke. A novel hierarchical intrusion detection system based on decision tree and rules-based models. 2019. doi: 10.1109/DCOSS.2019.00059.
- [5] D. Aksu, S. Ustebay, M. Aydin, and T. Atmaca. *Intrusion Detection with Comparative Analysis of Supervised Learning Techniques and Fisher Score Feature Selection Algorithm*, pages 141–149. 2018. doi: 10.1007/978-3-030-00840-6_16.
- [6] M. L. at Berkeley. Deep learning decall fall 2017 day 6: Autoencoders and representation learning, 2017. URL <https://www.youtube.com/watch?v=R3DNKE3zKFk>.
- [7] B. Boehmke and B. Greenwell. *Hands-On Machine Learning with R*. 2019. URL <https://bradleyboehmke.github.io/HOML/>.
- [8] A. Bohara, M. Noureddine, A. Fawaz, and W. Sanders. An unsupervised multi-detector approach for identifying malicious lateral movement. pages 224–233, 2017. doi: 10.1109/SRDS.2017.31.
- [9] G. Brogi. *Real-time detection of Advanced Persistent Threats using Information Flow Tracking and Hidden Markov Models*. PhD dissertation, Ecole Doctorale Informatique, Telecommunications et Electronique (Paris), 2018.
- [10] B. Burr, S. Wang, G. Salmon, and H. Soliman. On the detection of persistent attacks using alert graphs and event feature embeddings. pages 1–4, 04 2020. doi: 10.1109/NOMS47738.2020.9110439.
- [11] R. J. Campello, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. 2020. doi: 10.1002/widm.1343.
- [12] R. J. G. B. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 160–72, 2013. doi: 10.1007/978-3-642-37456-2_14.

- [13] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Trans. Knowl. Discov. Data*, 10(1), 2015. doi: 10.1145/2733381. URL <https://doi.org/10.1145/2733381>.
- [14] J. J. Davis and A. J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Comput. Secur.*, 30(6–7):353–375, 2011. doi: 10.1016/j.cose.2011.05.008.
- [15] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.
- [16] H. Ekenel and B. Sankur. Feature selection in the independent component subspace for face recognition. *Pattern Recognition Letters*, 25(12):1377 – 1388, 2004. doi: <https://doi.org/10.1016/j.patrec.2004.05.013>.
- [17] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD’96, page 226–231. AAAI Press, 1996.
- [18] B. Everitt and T. Hothorn. *An introduction to applied multivariate analysis with R*. Springer, 2011. URL <http://dx.doi.org/10.1007/978-1-4419-9650-3>.
- [19] G. Fernandez and S. Xu. A case study on using deep learning for network intrusion detection. 2019.
- [20] J. Forkman, J. Josse, and H.-P. Piepho. Hypothesis tests for principal component analysis when variables are standardized. *Journal of Agricultural, Biological and Environmental Statistics*, 2019. doi: 10.1007/s13253-019-00355-5.
- [21] J. Forshaw. *Attacking Network Protocols: A Hacker’s Guide to Capture, Analysis, and Exploitation*. No Starch Press, USA, 1st edition, 2017. ISBN 1593277504.
- [22] L. L. Gao, J. Bien, and D. Witten. Are clusterings of multiple data views independent?, 2019.
- [23] M. Gidlund and N. Debernardi. Scheduling performance of heavy-tailed data traffic in wireless high-speed shared channels. In *Proceedings of the 2009 IEEE Conference on Wireless Communications ‘&’ Networking Conference, WCNC’09*, page 1818–1823. IEEE Press, 2009.
- [24] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. <http://www.deeplearningbook.org>.
- [25] J. Heaton. *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, Inc., 2nd edition, 2008. ISBN 1604390085.

- [26] A. Hendrikse and L. Spreeuwens. Component ordering in independent component analysis based on data power. In R. Veldhuis, R. Veldhuis, and H. Cronie, editors, *Proceedings of the 28th Symposium on Information Theory in the Benelux*, number LNCS4549, pages 211–218, Netherlands, 6 2007. Werkge-meenschap voor Informatie- en Communicatietheorie (WIC).
- [27] J. Himberg and A. Hyvärinen. Icasso: software for investigating the reliability of ica estimates by clustering and visualization. *2003 IEEE XIII Workshop on Neural Networks for Signal Processing (IEEE Cat. No.03TH8718)*, pages 259–268, 2003.
- [28] M. Hubert and E. Vandervieren. An adjusted boxplot for skewed distributions. *Computational Statistics ‘&’ Data Analysis*, 52:5186–5201, 2007. doi: 10.1016/j.csda.2007.11.008.
- [29] M. Hubert, P. Rousseeuw, and K. Branden. Robpca: A new approach to robust principal component analysis. *Technometrics*, 47:64–79, 2005. doi: 10.1198/004017004000000563.
- [30] M. Hubert, P. Rousseeuw, and T. Verdonck. Robust pca for skewed data and its outlier map. *Computational Statistics ‘&’ Data Analysis*, 53:2264–2274, 2009. doi: 10.1016/j.csda.2008.05.027.
- [31] A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Netw.*, 13(4–5):411–430, 2000. doi: 10.1016/S0893-6080(00)00026-5.
- [32] F. T. Intelligence. Apt30 and the mechanics of a long-running cyber espionage operation. Technical report, FireEye Labs, Milpitas, CA, 2015.
- [33] D. Jackson. Stopping rules in principal component analysis: a comparison of heuristical and statistical approaches. *Ecology*, 74:2204–2214, 1993.
- [34] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 5. ed edition, 2002.
- [35] I. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2010.
- [36] M. Joshi and T. Aldhayni. A review of network traffic analysis and prediction techniques. 2015.
- [37] U. Kairov, L. Cantini, A. Greco, A. Molkenov, U. Czerwinska, E. Barillot, and A. Zinovyev. Determining the optimal number of independent components for reproducible transcriptomic data analysis. *BMC Genomics*, 18, 2017. doi: 10.1186/s12864-017-4112-9.
- [38] M. Kakavand, N. Mustapha, A. Mustapha, and M. T. Abdullah. Effective dimensionality reduction of payload-based anomaly detection in tmad model for http payload. *KSII Transactions on Internet and Information Systems*, 10(8), 2016.

- [39] M. W. Kevin R. Coombes. *PCDimension: Finding the Number of Significant Principal Components*, 2019. R package version 1.1.11.
- [40] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman. Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, 2, 2019. doi: 10.1186/s42400-019-0038-7.
- [41] N. Kumar and U. Kumar. Anomaly-based network intrusion detection: An outlier detection techniques. 2019. ISBN 978-3-319-60617-0. doi: 10.1007/978-3-319-60618-7_26.
- [42] K. Kurniabudi, B. Purnama, D. Sharipuddin, D. Stiawan, S. Sahmin, A. Heryanto, and R. Budiarto. Network anomaly detection research: A survey. *Indonesian Journal of Electrical Engineering and Informatics*, 7:36–49, 2019. doi: 10.11591/ijeei.v7i1.773.
- [43] M. Labonne, A. Olivereau, B. Polve, and D. Zeghlache. Unsupervised protocol-based intrusion detection for real-world networks. pages 299–303, 2020.
- [44] B. Li, S. Zhang, and K. Li. Towards a multi-layers anomaly detection framework for analyzing network traffic. *Concurrency and Computation: Practice and Experience*, 29, 2016. doi: 10.1002/cpe.3955.
- [45] R. Li and X. Wang. Dimension reduction of process dynamic trends using independent component analysis. *Computers ‘&’ Chemical Engineering*, 26(3):467 – 473, 2002. doi: [https://doi.org/10.1016/S0098-1354\(01\)00773-6](https://doi.org/10.1016/S0098-1354(01)00773-6).
- [46] H. Liu, R. Paffenroth, J. Zou, and C. Zhou. Anomaly detection via graphical lasso. 2018.
- [47] K.-H. Liu, B. Li, J. Zhang, and J.-X. Du. Ensemble component selection for improving ica based microarray data prediction models. *Pattern Recognition*, 42:1274–1283, 2009. doi: 10.1016/j.patcog.2009.01.021.
- [48] Y. Liu, X. Gao, Q. Gao, L. Shao, and J. Han. Adaptive robust principal component analysis. *Neural Networks*, 119:85 – 92, 2019. ISSN 0893-6080. doi: <https://doi.org/10.1016/j.neunet.2019.07.015>. URL <http://www.sciencedirect.com/science/article/pii/S0893608019302035>.
- [49] P. Manandhar and Z. Aung. Intrusion detection based on outlier detection method. In *International Conference on Intelligent Systems, Data Mining and Information Technology - ICIDIT’2014*, 2014.
- [50] T. Masters. *Practical Neural Network Recipes in C++*. Academic Press Professional, Inc., USA, 1993. ISBN 0124790402.
- [51] L. Mathur, M. Raheja, and P. Chaudhary. Botnet detection via mining of network traffic flow. *Procedia Computer Science*, 132:1668–1677, 2018. doi: 10.1016/j.procs.2018.05.137.

- [52] L. McInnes and J. Healy. Accelerated hierarchical density based clustering. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pages 33–42. IEEE, 2017.
- [53] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.
- [54] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 2013.
- [55] I. Mitliagkas, C. Caramanis, and P. Jain. Memory limited, streaming pca. *Advances in Neural Information Processing Systems*, 2013.
- [56] D. Moulavi, P. A. Jaskowiak, R. Campello, A. Zimek, and J. Sander. Density-based clustering validation. 2014. doi: 10.1137/1.9781611973440.96.
- [57] K. P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013.
- [58] R. C. Nethery. *Special Topics in Latent Variable Models with Spatially and Temporally Correlated Latent Variables*. PhD thesis, The University of North Carolina at Chapel Hill, 2017.
- [59] P. Odiowei and Y. Cao. State-space independent component analysis for nonlinear dynamic process monitoring. *Chemometrics and Intelligent Laboratory Systems*, 103(1):59 – 65, 2010. doi: <https://doi.org/10.1016/j.chemolab.2010.05.014>.
- [60] J. Oh and N. Kwak. Generalized mean for robust principal component analysis. *Pattern Recognition*, 54, 2016. doi: 10.1016/j.patcog.2016.01.002.
- [61] P. Olver and C. Shakiban. *Applied Linear Algebra*. Springer International Publishing, 2018.
- [62] F. Palmieri, U. Fiore, and A. Castiglione. A distributed approach to network anomaly detection based on independent component analysis. *Concurrency and Computation: Practice and Experience*, 26, 2014. doi: 10.1002/cpe.3061.
- [63] R. Panigrahi and S. Borah. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. 7:479–482, 2018.
- [64] C. Pascoal, M. R. Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco. Robust feature selection and robust pca for internet traffic anomaly detection. *Proceedings - IEEE INFOCOM*, pages 1755–1763, 2012. doi: 10.1109/INFOCOM.2012.6195548.

- [65] A.-S. K. Pathan. *The State of the Art in Intrusion Prevention and Detection*. Auerbach Publications, USA, 2014. ISBN 1482203510.
- [66] G. Patil, V. Pachghare, and D. Kshirsagar. Feature reduction in flow based intrusion detection system. pages 1356–1362, 2018. doi: 10.1109/RTEICT42901.2018.9012554.
- [67] V. Platonov and P. Semenov. Dimension reduction in network attacks detection systems. *Nonlinear Phenomena in Complex Systems*, 17(3):284–289, 2014.
- [68] M. Prasad, S. Tripathi, and K. Dahal. An efficient feature selection based bayesian and rough set approach for intrusion detection. *Applied Soft Computing*, 87:105980, 2020. doi: 10.1016/j.asoc.2019.105980.
- [69] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL <http://www.R-project.org/>.
- [70] R. D. Reed and R. J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262181908.
- [71] A. C. Rencher and W. F. Christensen. *Methods of multivariate analysis*. Wiley, 2012.
- [72] T. Reynkens, V. Todorov, M. Hubert, E. Schmitt, and T. Verdonck. *Robust Sparse PCA using the ROSPCA Algorithm*, 2018. R package version 1.0.4.
- [73] M. Ring, A. Dallmann, D. Landes, and A. Hotho. Ip2vec: Learning similarities between ip addresses. pages 657–666, 11 2017. doi: 10.1109/ICDMW.2017.93.
- [74] H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of pca for traffic anomaly detection. volume 35, pages 109–120, 2007. doi: 10.1145/1254882.1254895.
- [75] B. B. Risk, D. S. Matteson, and D. Ruppert. Linear non-gaussian component analysis via maximum likelihood. *Journal of the American Statistical Association*, 114(525):332–343, 2019.
- [76] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims. Evaluation methods for unsupervised word embeddings. pages 298–307, 2015. doi: 10.18653/v1/D15-1036.
- [77] N. N. Schraudolph and X. Giannakopoulos. Online independent component analysis with local learning rate adaptation. In *Neural Information Processing Systems*, pages 789–795. MIT Press, 2000.
- [78] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3), 2017. doi: 10.1145/3068335.

- [79] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. pages 108–116, 2018. doi: 10.5220/0006639801080116.
- [80] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang. A novel anomaly detection scheme based on principal component classifier. 2003.
- [81] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman ‘&’ Hall/CRC, 1998.
- [82] B. Tang and H. He. A local density-based approach for outlier detection. *Neurocomputing*, 241:171 – 180, 2017. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2017.02.039>.
- [83] C. ter Braak. *CANOCO - a FORTRAN program for canonical community ordination by [partial] [detrended] [canonical] correspondence analysis, principal components analysis and redundancy analysis (version 2.1)*, volume 11. 1987.
- [84] L. Tu. Density-based clustering for real-time stream data. pages 133–142, 2007. doi: 10.1145/1281192.1281210.
- [85] S. Ustebay, Z. Turgut, and M. Aydin. *Cyber Attack Detection by Using Neural Network Approaches: Shallow Neural Network, Deep Neural Network and AutoEncoder*, pages 144–155. 2019. ISBN 978-3-030-21951-2. doi: 10.1007/978-3-030-21952-9_11.
- [86] G. Wang, J. Yang, and R. Li. An anomaly detection framework based on ica and bayesian classification for iaas platforms. *KSII Transactions on Internet and Information Systems*, 10(8), 2016.
- [87] J. Wang, Y. Zhang, H. Cao, and W. Zhu. Dimension reduction method of independent component analysis for process monitoring based on minimum mean square error. *Journal of Process Control*, 22(2):477 – 487, 2012. doi: <https://doi.org/10.1016/j.jprocont.2011.11.005>.
- [88] L. Wang and R. Jones. Big data analytics for network intrusion detection: A survey. *International Journal of Networks and Communications*, 7(1):24–31, 2017. doi: 10.5923/j.ijnc.20170701.03.
- [89] J. West, T. Dean, and J. Andrews. *Network+ Guide to Networks*. Course Technology Press, Boston, MA, USA, 8th edition, 2018. ISBN 133756933X.
- [90] R. Woods, L. Hansen, and S. Strother. How many separable sources? model selection in independent components analysis. *PLOS ONE*, 10, 2015. doi: 10.1371/journal.pone.0118877.
- [91] A. Yaseen, I. Shakhathreh, and K. A. Bakar. A review of clustering techniques based on machine learning approach in intrusion detection systems. *IJCSI International Journal of Computer Science Issues*, 8(3), 2011.

- [92] B. Zhang, Z. Liu, Y. Jia, J. Ren, and X. Zhao. Network intrusion detection method based on pca and bayes algorithm. *Security and Communication Networks*, 2018:1–11, 2018. doi: 10.1155/2018/1914980.
- [93] R. Zhang, Y. Huo, J.-y. Liu, and F. Weng. Constructing apt attack scenarios based on intrusion kill chain and fuzzy clustering. *Security and Communication Networks*, 2017:1–9, 2017. doi: 10.1155/2017/7536381.
- [94] C. Zhou and R. Paffenroth. Anomaly detection with robust deep autoencoders. pages 665–674, 2017. doi: 10.1145/3097983.3098052.