Malmenator

Network Anomaly Detection

Final Year Project

Final Report

May 4, 2020

David B. Han (3035344211)
Piyush Jha (3035342691)
Supervisor: Dr. Dirk Schnieders
University of Hong Kong

github.com/piy0999/Malmenator

Abstract

With the rise in digital adoption, having an early warning system in the event of a cyberattack is increasingly important. One of the most common ways this is implemented is through network-based intrusion detection systems (NIDSs) that identify malicious network activity and notify the appropriate stakeholders to take responsive actions. Malamenator is an NIDS project that consists of two components: an engineering component to produce a secure NIDS-integrated router, and a research component to determine the best anomaly detection methods to deploy on the system. This paper focuses on the latter.

Many approaches to anomaly detection in network flow data have sub-optimal performance due to challenges in the lack of interpretability as well as due to the absence of accurate, representative, and consistent datasets for training. Despite this, researchers are constantly proposing more and more advanced techniques to gain marginally higher performance on network datasets that have disjoint features. However, little work has been done to look at feature engineering as a method to improve model performance.

This paper highlights the shortcomings of NSL-KDD as a benchmark dataset and shows that two more recent datasets, UNSW-NB15 and CICIDS 2017, provide features that more accurately capture malicious behavior. We further show that with the appropriate feature engineering, out-of-box ensemble learners like Random Forest, XGBoost, and LightGBM are able to exhibit state of the art performance at an extremely reduced computational cost. Each of the ensemble learners were able to achieve a detection rate and f1 score above 99% for both the UNSW-NB15 and CICIDS 2017 datasets with an extremely limited training set size.

Acknowledgements

This report would not be possible without our supervisor, Dr. Dirk Schnieders, whose support and input allowed us to reach our current accomplishments. We would also like to extend deep gratitude to Ms. Mable Choi from the Centre of Applied English Studies at the University of Hong Kong for her feedback on improving the structure and content of the report.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Year after year, spending on cybersecurity increases, but the cost of cyberattacks only continues to grow[1]. Despite this phenomena, there is a shortage of investments for public research into network security, which has led to a lack of robust and interpretable network anomaly detection methods [2]. Malmenator is a project that aims to make cybersecurity more accessible to broader society at a reduced cost to enable trust in confidence in an increasingly digital world. We aim to create an effective, affordable, and portable NIDS that can be utilized in any personal environment.

Here in this report, we focus on research goal of analyzing anomaly detection techniques. These techniques will be implemented in the custom NIDS covered in [3].

1.2 Problem formulation

From an academic perspective, network anomaly detection faces two large challenges. The foremost problem is the lack of a single widely accepted network traffic dataset. Each of the benchmark network flow datasets available contain distinct features. Without a comprehensive and representative dataset, the second problem is reached - appropriately developing, testing, and deploying anomaly-based detection techniques using standardized norms and metrics. Malmenator investigates the performance of anomaly detection models across datasets and proposes a robust and scalable solution to implement as part of a hybrid anomaly detection tool for NIDS evaluation.

1.3 Contributions

Malmenator aims to push the explore new techniques for anomaly detection that are more directly relevant to the problem domain of cybersecurity. In particular, Malmenator proposes that the key to making breakthroughs lies in the creation of better features rather than better models. Lastly, Malmenator makes available an accessible NIDS implemented on top of a Raspberry Pi that functions as a secure internet access point.

This report focuses on the former contributions detailing the research into anomaly detection methodologies, while the last is covered in [3]. Specifically, this report shows the robustness of ensemble learners for anomaly detection in network flows and proposes that the features relevant to anomaly detection lie primarily in the statistical behavior and contents of packets. Thus, future benchmark datasets ought to engineer and include these relevant features from the captured pcap data.

1.4 Report organization

The remaining contents of this report are as follows. This report begins by providing a technical background to the machine learning classification techniques that are used in anomaly detection. It then gives an introduction to several key network datasets used in anomaly detection that are popular in research before proceeding to discuss some state of the art techniques that have been used to perform anomaly detection. From there, this report outlines the methodology by which we perform anomaly detection across various datasets and evaluates the performance of our models compared to other relevant research. Finally, this report provides a look into future directions of research, some challenges that we dealt with, as well as a few concluding remarks.

Please note that this report is mean to be read in conjunction with [3], and the relevance and background of certain sections rely on knowledge covered in its twin report. Relevant sections in [3] will be addressed in each chapter's overview.

Chapter 2

Technical background

2.1 Overview

This section provides an introduction to the tools employed by Malmenator for our research in network anomaly detection. This section is primarily for those who are unfamiliar with classical machine learning methods or for those who are interested in a brief overview of various anomaly detection techniques.

2.2 Machine learning methods in network anomaly detection

The anomaly detection problem boils down to the task of labeling a data point as being either a normal point or an outlier (i.e. harmless or malicious network traffic in this case). The following subsections cover different machine learning approaches to anomaly classification that this project will explore with a particular emphasis on tree based ensemble learners. Variations of these techniques are some of the highest performers on tabular data found on Kaggle today [4].

An overview for network anomaly detection techniques is depicted in figure 2.1. Note that an exhaustive and in depth discussion on the various subcategories of each methodology is not discussed in this report, but are discussed at length in several overview and survey articles including [6] [7] [8] [9] [5]. This section will center its discussion on the shallow and supervised model of anomaly detection. Specifically, we will focus on decision trees and advanced techniques that stem from them.

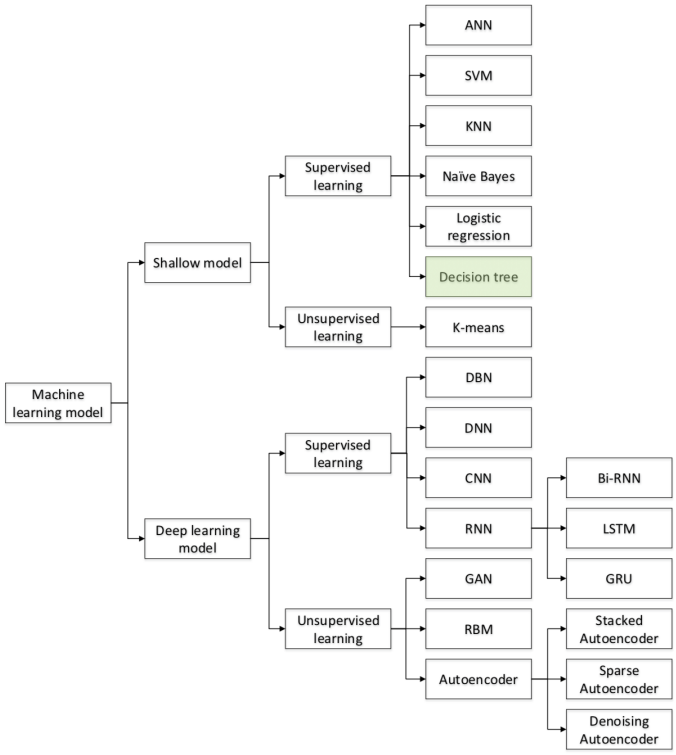


Figure 2.1: Taxonomy of network anomaly detection methods

Categorization of the main approaches to network anomaly detection as shown in [5]. In green, we have highlighted the decision tree approach which is the building blocks to the methodology that we explore. Note that this list is by no means exhaustive. Research is being done on the applications of new algorithms in anomaly detection such as genetic algorithms, artificial immune systems, and other statistical and ensemble based methods. The full-name terms of the abbreviations can be easily found and explained online.

2.3 Decision trees

Decision tree algorithms are a machine learning technique that has been widely used in both classification and regression problems. They operate by splitting the data into different partitions based on certain if-else decision rules. With each decision rule, the data is split into smaller subsets until a predefined depth of the splitting is reached or a maximum number of subsets are reached. The partitioning method can differ based on different information gain metrics, and a number of decision tree implementation have been created including ID3, C4.5, and CART [10] [11] [12].

2.3.1 Classification and regression tree (CART)

The CART (classification and regression tree) implementation of decision trees was proposed in 1986 [12], but is still one of the highest performing forms of decision trees that is currently implemented in the popular machine learning package, scikit-learn [13]. CART is similar to C4.5, but has the added benefit of supporting regression problems. Other advantages of CART found by [14] include the fact that CART is non parametric (no probabilistic assumptions) and that it is able to deal with missing values.

2.4 Ensemble learning

Ensemble learners, like their name implies, are an ensemble of different machine learning algorithms whose decisions are combined and taken into consideration holistically. This enables the ensemble to make better predictions than any of its individual constituents [15]. There are several approaches to determining the ensemble's final output including a simple majority vote of its constituent models or a weighted vote where different constituent models have a stronger input than others. The classical machine learning implemented in Malmenator below are all ensemble learners.

2.4.1 Bagging (bootstrap aggregating)

Bagging is technique for ensemble learners that is designed to reduce variance, improve stability and accuracy, and avoid overfitting. Bagging divides the original training set into multiple smaller training sets by sampling the original training set uniformly and with replacement. Each of the models inside the ensemble learner is then fit onto a corresponding smaller training set. It has been shown that bagging leads to improvements for unstable procedures, which include classification decision trees [16] [17].

2.4.2 Boosting

Boosting is another technique for ensemble learners that is used to reduce bias and variance. With boosting, multiple weak learners within an ensemble are weighted and adjusted in such a manner that they are able become a strong learner [18]. Here, weak learners refer to simpler models with non-ideal performance who are then added to a stronger classifier with a weight that corresponds to the weak learner's accuracy. With each iteration, new weak learners focus on data that was previously

misclassified. Since the rise of boosting, some variation of boosting algorithms have been used to win a majority of Kaggle competitions according to its CEO, Anthony Goldbloom [4]. Boosting was popularized by the Adaboost implementation by [19].

2.4.2.1 Gradient boosting

Gradient boosting is variation on the standard boosting algorithm that approaches the error adjustments of boosting from a gradient-descent based formulation [20] [21]. In gradient boosting, new weak learners are maximally correlated with the negative gradient of the loss function associated with the whole ensemble [22]. The loss function can be chosen or customized by the user, which enables gradient boosted ensamble to be highly flexible and customizable to a variety of domains.

2.4.3 Random forest (RF)

Random forest is supervised ensemble learner method that is used primarily for classification. It consists of a predefined number of decision trees who are trained one different subsets of the training set. Here, boosting can be used to divide up the training set into subsets for each decision tree. At the time of prediction, the random forest ensemble outputs the mode of the decisions of the individual decision trees [23]. RFs are highly robust because they can reduce overfitting to the training set [24], and are widely considered a top-performing machine learning algorithm [25]. In this paper, we focus on random forests that implement underlying decision trees with the CART algorithm that is implemented in scikit-learn [13].

2.4.4 XGBoost and LightGBM

In this paper, we focus on two highly popular implementations of gradient boosting libraries: XGBoost [26], which was released in 2016 , and LightGBM [27], released in 2017. The main difference between the two is that XGBoost uses a pre-sorted and histogram-based algorithm to detemine the appropriate partition of decisions. On the other hand, LightGBM uses a new technique, gradient-based one-side sampling (GOSS), to find the best split. Both libraries are extremely robust and high-performing.

2.5 Particulars for anomaly detection

In the anomaly detection, extra care must be taken when applying models to data due to the nature of the problem scope. Different challenges in machine learning for anomaly detection are listed below.

2.5.1 Class balancing

By definition, anomalies appear far less frequently than the majority class. This can cause issues when training classifiers since the classifiers may tend to overlook features that relate to anomalous activities. Thus, a number of techniques have been proposed to remedy this solution, namely:

1. *Downsampling the majority class*: Downsampling the majority class is a popular technique to acheive class balance, and is commonly used preprocess the training data in network anomaly detection.

2. *Upsampling the minority class*: Upsampling the minority class is similar to bootstrapping data in that we can duplicate the minority class. However, it is not commonly used in network anomaly detection due to the biases that can arise from duplicating a large number of flows.

3. *Creating synthetic minority class data*: SMOTE is one of the most popular techniques for generating synthetic data [28]. This technique works by finding clusters of minority data and creating synthetic data that would also lie within that cluster. However, the data generated by this technique may not reflect real-life malicious traffic data, so this technique is not consistently used in network anomaly detection.

4. *Combine the minority classes*: In multiclass problems with several different anomaly classes, the problem can be reworked into a binary classification problem to detect anomalies. This approach is widely used for network anomaly detection.

5. *Penalize misclassified minority classes extra*: By assigning more importance to classifying minority classes, we can allow models to learn from a smaller set of anomalies. Interestingly, although this technique is widely applied in other problem domains, it has not found much traction in the network anomaly detection area.

2.5.2 Model performance metrics

In network anomaly detection, great care is taken to reduce the number of false negatives and minimize the amounts of false positives as described in table 2.1. A true positive (TP) occurs when a predicted anomaly is a real anomaly, and a true negative (TN) occurs when a predicted benign flow is a real benign flow. More importantly, we focus on false positives (FP), when a benign flow is predicted as anomalous, as well as false negatives (FN), when an anomalous flow is predicted to be benign. Also note that in the network anomaly detection domain, false positives are also known as false alarms.

		Actual Values	
		True (Anomaly)	False (Benign)
Predicted Values	True (Anomaly)	True Positive	False Positive
	False (Benign)	False Negative	True Negative

Table 2.1: Classification of TP, FP, FN, and TN

Table indicating the different types of predictions for a binary classification problem. In anomaly detection, we aim to minimize the number of false positives and false negatives.

To quantify performance of a model, there are the concepts of accuracy, precision, recall (known in this space as detection rate), false alarm rate (FAR), and f1-score. The formulas of each are listed below:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy is the ratio of correct predictions to the total predictions. It is the most intuitive performance measure, but it fails with asymmetric datasets such as anomaly detection. For example, if only 1% of netflow traffic is anomalous, one could achieve .99 accuracy by blindly predicting everything as benign.

$$Precision = \frac{TP}{TP + FP}$$

Precision is the ratio of correct positive (anomalous) predictions to the total predicted positives. High precision implies a low false positive rate, so it is a useful metric in anomaly detection.

$$Recall = Detection\ Rate = \frac{TP}{TP + FN}$$

Recall is the ratio of correct positive predictions to all actual positive instances. This can be useful

in helping us indirectly asses our false negatives.

$$False\ Alarm\ Rate = \frac{FP}{FP + TN}$$

False alarm rate is the ratio of incorrectly predicted positives to all true negatives. This can be helpful in letting us know what percentage of benign network traffic we are flagging as anomalous.

$$F1 = 2 * \frac{(Recall * Precision)}{Recall + Precision}$$

Lastly, the F1 score is the weighted average of precision and recall and gives a holistic picture that captures the rate of false positives and negatives.

2.6 Summary

In this section, we have given a high level overview of some different approaches that are used for anomaly detection, and have discussed in greater detail the types of algorithms that are used in this paper. Furthermore, this section discussed unique challenges in the network anomaly detection space and their remedies.

Chapter 3

Literature Review

3.1 Overview

This chapter provides an in depth look into the context of the Malmenator project through a review of relevant literature. It first looks at research in the areas of network traffic dataset generation before proceeding with various model approaches in network anomaly detection.

Given that cybersecurity is a niche field in cybersecurity, it may prove useful to read ıCh.2 A Primer on Cybersecurity from [3] for a general introduction to the field as well as an overview of different types of cyberattacks. It is also advisable to read ıCh.3 Technical Background from the same report for a deeper understanding of the data types, such as flow-based data, that will be encountered later in this section.

3.2 Network traffic datasets

One of the major research challenges in training and evaluating an NIDS is the unavailability of a single comprehensive network traffic dataset which that reflects modern network traffic scenarios [7]. Since the effectiveness of an NIDS is evaluated based on its performance against data set that contains normal and abnormal behaviors, it is critically important to use a combination of datasets that feature different attack behaviours to serve as a metric for evaluation.

3.2.1 Methods for network traffic dataset generation

Testing NIDSs against realistic data has always been a challenge in the network security field as captured data from the real world is sensitive and is not publicly available [29]. Thus, the common practice is to generate realistic data in an environment that mimics the real world as close as possible.

There are three main methods by which network data is generated:

1. *Simulation*: all entities of the network are simulated
2. *Emulation*: attackers and targets are physically implemented, while the network structure is simulated with software
3. *Physical Representation*: all entities of the network are physically implemented

Each method has their own benefits and drawbacks that are more fully discussed in [29]. It important to note that testing NIDSs is not a straightforward task, and that it is recommended to use a number of datasets for comprehensive evaluation [30]. In the network security field, it is important to be aware of the means by which datasets are generated as network dataset generation is a huge research area itself.

3.2.2 Old reference datasets

The most famous and classical benchmark network datasets are the KDD99, DARPA 1998, and DARPA 1999 datasets, were all gathered from simulated environments [31] [32] [33]. Unfortunately, since their inception nearly two decades ago, a number of studies have shown that evaluating an NIDS on these datasets is ineffective as they do not reflect realistic network data [34] [35] [36] [37]. These datasets have not been able to accurately reflect the changing landscape of network cybersecurity.

3.2.3 Key datasets

3.2.3.1 NSL-KDD

To solve some issues revolving around the KDD99 dataset such as duplicated entries and improper class balancing, the NSL-KDD dataset was created in 2009. NSL-KDD enhances the KDD99 dataset by removing a large number of duplicated columns and eliminating rows with erroneous data [38]. It has advantages over the original KDD99 dataset including that there are no redundant records

in the train sets, there are no duplicate records in the test sets, and there are fewer records in the training and test sets that make it more reasonable to run experiments on.

Attack Class	Training Instances	Testing Instances
Normal	67343	9710
DoS	45927	7458
Probe	11656	2422
R2L	995	2887
U2R	52	67
Total	125973	22544

Table 3.1: Attack class proportions of NSL-KDD training and testing set

#	Feature
1	duration
2	protocol_type
3	service
4	flag
5	src_bytes
6	dst_bytes
7	land
8	wrong_fragment
9	urgent
10	hot
11	num_failed_logins
12	logged_in
13	num_compromised
14	root_shell
15	su_attempted
16	num_root
17	num_file_creations
18	num_shells
19	num_access_files
20	num_outbound_cmds
21	is_host_login
22	is_guest_login
23	count
24	srv_count
25	serror_rate
26	srv_serror_rate
27	rerror_rate
28	srv_rerror_rate
29	same_srv_rate
30	diff_srv_rate
31	srv_diff_host_rate
32	dst_host_count
33	dst_host_srv_count
34	dst_host_same_srv_rate
35	dst_host_diff_srv_rate
36	dst_host_same_src_port_rate
37	dst_host_srv_diff_host_rate
38	dst_host_serror_rate
39	dst_host_srv_serror_rate
40	dst_host_rerror_rate
41	dst_host_srv_rerror_rate

Table 3.2: The 41 features in the NSL-KDD dataset

However, it is important to note that the underlying network traffic of the NSL-KDD dataset dates back to more than two decades ago, which renders it a poor representation of a modern attack environment [39]. Furthermore, the reduced number of elements inside the NSL-KDD dataset do not address the concerns laid out in [36]. Despite this, the KDD99 and NSL-KDD cup continue to be widely used as benchmark datasets for validating anomaly detection models even with the

availability of newer and more relevant datasets.

The NSL-KDD dataset contains a predefined train and test set that enable researchers to quickly and easily run experiments. Their breakdown and attributes are listed in tables 3.1 and 3.2.

3.2.3.2 UNSW-NB15

The UNSW-NB15 dataset was created using an emulated environment using the IXIA Perfect Storm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) using a mix of synthetic modern attack patters and real user behaviors. It is available in both packet-based and flow-based formats, which makes it highly suitable for experimentation [39]. The tcpdump tool was used to capture pcap files of the raw traffic which contained nine different forms of attacks: Generic, Exploits, Backdoors, Fuzzers, Analysis, Backdoors, DoS, Reconnaissance, Worms, and Shellcode. UNSW-NB15 was created with the intention of solving several of the criticisms of the KDD99 and NSL-KDD datasets in the hope of being a contemporary benchmark dataset for NIDSs [39].

Attack Class	Training Instances	Testing Instances
Normal	56000	37000
Fuzzers	18184	6062
Analysis	2000	677
Backdoors	1746	583
DoS	12264	4089
Exploits	33393	11132
Generic	40000	18871
Reconnaissance	10491	3496
Shell Code	1133	378
Worms	130	44
Total	175341	82332

Table 3.3: Attack class proportions of UNSW-NB15 training and testing set

The UNSW-NB15 dataset contains 49 features and is also presplit into training and test sets for experimentation that are highlighted in tables 3.3 and 3.4. Additionally, they also make the entire super set of flow data available to experimentation on larger datasets as needed.

3.2.3.3 CICIDS 2017

The CICIDS 2017 dataset was also created using an emulated environment and contains network traffic in both packet and flow-based formats. It comprises of 5 days of network traffic generated in an emulated environment and comes in both bidirectional flow-based format and the standard

#	Feature	Description
	Flow features	
1	srcip	Source IP address
2	sport	Source port number
3	dstip	Destinations IP address
4	dsport	Destination port number
5	proto	Protocol type (e.g. TCP)
	Basic features	
6	state	State dependent protocol (e.g. CON)
7	dur	Flow duration
8	sbytes	Source to destination bytes
9	dbytes	Destination to source bytes
10	sttl	Source to destination time to live
11	dttl	Destination to source time to live
12	sloss	Source packets retransmitted/dropped
13	dloss	Destination packets retransmitted/dropped
14	service	e.g. HTTP, STP
15	sload	Source bits per second
16	dload	Destination bits per second
17	spkts	Source to destination packet count
18	dpkts	Destination to source packet count
	Content features	
19	swin	Source TCP window advertisement value
20	dwin	Destination TCP window advertisement value
21	Stcpb	Source TCP base sequence number
22	dtcpb	Destination TCP base sequence number
23	smeansz	Mean of the packet size transmitted by the srcip
24	dmeansz	Mean of the packet size transmitted by the dstip
25	trans_depth	The connection of http request/response transaction
26	res_bdy_len	The content size of the data transferred from http
	Time features	
27	sjit	Source jitter
28	djit	Destination jitter
29	stime	Row start time
30	ltime	Row last time
31	sintpkt	Source inter-packet arrival time
32	dintpkt	Destination inter-packet arrival time
33	tcprtt	Setup round-trip time, the sum of synack and ackdat
34	synack	The time between the SYN and the SYN_ACK packets
35	ackdat	The time between the SYN_ACK and the ACK packets
36	is_sm_ips_ports	If srcip (1) = dstip (3) and sport (2) = dsport (4), assign 1 else 0
	Additional features	
37	ct_state_ttl	No. of each state (6) according to values of sttl (10) and dttl (11)
38	ct_flw_http_mthd	No. of methods such as Get and Post in http service
39	is_ftp_login	If the ftp session is accessed by user and password then 1 else 0
40	ct_ftp_cmd	No of flows that has a command in ftp session
41	ct_srv_src	No. of rows of the same service (14) and srcip (1) in 100 rows
42	ct_srv_dst	No. of rows of the same service (14) and dstip (3) in 100 rows
43	ct_dst_ltm	No. of rows of the same dstip (3) in 100 rows
44	ct_src_ltm	ltm No. of rows of the srcip (1) in 100 rows
45	ct_src_dport_ltm	No of rows of the same srcip (1) and the dsport (4) in 100 rows
46	ct_dst_sport_ltm	No of rows of the same dstip (3) and the sport (2) in 100 rows
47	ct_dst_src_ltm	No of rows of the same srcip (1) and the dstip (3) in 100 records
	Labelled features	
48	Attack_cat	The name of each attack category
49	Label	0 for normal and 1 for attack records

Table 3.4: The 49 features in the UNSW-NB15 dataset

Attack Class	Instances
BENIGN	2359087
Bot	1966
DDoS	41835
DoS GoldenEye	10293
DoS Hulk	231072
DoS Slow-httptest	5499
DoS slowloris	5796
FTP-Patator	7938
Heartbleed	11
Infiltration	36
PortScan	158930
SSH-Patator	5897
Web Attack - Brute Force	1507
Web Attack - Sql Injection	21
Total	2829888

Table 3.5: Attack class proportions of CICIDS 2017 dataset

packet-based format. It consists of more than 80 features for each flow and enriches the data about IP addresses and cyberattacks by providing extra metadata for both of them. Normal user behavior was executed via scripts, and the attacks have a wide range, including SSH brute force, DDoS, heartbleed, and botnet attacks [40]. A detailed breakdown of this dataset is given in tables 3.5 and 3.6.

A major advantage of this dataset is that it comprises of a wide variety of modern cyberattack scenarios, which makes it a prime choice for its utilization in this project. The CICIDS2017 dataset is not split into any predefined training and testing sets, and a number of approaches have been taken for selecting testing and training datasets that allow for appropriate attack representation and class balancing [41] [42].

3.2.4 Dataset selection

According to the most recent survey on network datasets published earlier this year [30], the CIDDS-001, CICIDS 2017, UNSW-NB15, and UGR16 data sets the the most ideal for uses across generalized applications. UGR16 has a huge volume, CIDDS-001 contains detailed metadata for deeper analysis, and CICIDS 2017 and UNSW-NB15 have large variations in attack scenarios [30]. However, the CIDDS-001 and UGR'16 datasets have not been widely used in literature, so utilizing them as benchmark datasets is impractical. There are also a number of other anomaly detection datasets, but they are more appropriately utilized in specialized cases and for certain evaluation scenarios

#	Feature
1	Destination Port
2	Flow Duration
3	Total Fwd Packets
4	Total Backward Packets
5	Total Length of Fwd Packets
6	Total Length of Bwd Packets
7	Fwd Packet Length Max
8	Fwd Packet Length Min
9	Fwd Packet Length Mean
10	Fwd Packet Length Std
11	Bwd Packet Length Max
12	Bwd Packet Length Min
13	Bwd Packet Length Mean
14	Bwd Packet Length Std
15	Flow Bytes/s
16	Flow Packets/s
17	Flow IAT Mean
18	Flow IAT Std
19	Flow IAT Max
20	Flow IAT Min
21	Fwd IAT Total
22	Fwd IAT Mean
23	Fwd IAT Std
24	Fwd IAT Max
25	Fwd IAT Min
26	Bwd IAT Total
27	Bwd IAT Mean
28	Bwd IAT Std
29	Bwd IAT Max
30	Bwd IAT Min
31	Fwd PSH Flags
32	Bwd PSH Flags
33	Fwd URG Flags
34	Bwd URG Flags
35	Fwd Header Length
36	Bwd Header Length
37	Fwd Packets/s
38	Bwd Packets/s
39	Min Packet Length
40	Max Packet Length
41	Packet Length Mean
42	Packet Length Std
43	Packet Length Variance
44	FIN Flag Count
45	SYN Flag Count
46	RST Flag Count
47	PSH Flag Count
48	ACK Flag Count
49	URG Flag Count
50	CWE Flag Count
51	ECE Flag Count
52	Down/Up Ratio
53	Average Packet Size
54	Avg Fwd Segment Size
55	Avg Bwd Segment Size
56	Fwd Header Length
57	Fwd Avg Bytes/Bulk
58	Fwd Avg Packets/Bulk
59	Fwd Avg Bulk Rate
60	Bwd Avg Bytes/Bulk
61	Bwd Avg Packets/Bulk
62	Bwd Avg Bulk Rate
63	Subflow Fwd Packets
64	Subflow Fwd Bytes
65	Subflow Bwd Packets
66	Subflow Bwd Bytes
67	Init_Win_bytes_forward
68	Init_Win_bytes_backward
69	act_data_pkt_fwd
70	min_seg_size_forward
71	Active Mean
72	Active Std
73	Active Max
74	Active Min
75	Idle Mean
76	Idle Std
77	Idle Max
78	Idle Min
79	Label

Table 3.6: The 79 features in the CICIDS 2017 dataset

where they may excel such as in botnet attacks or android malware attacks. Discussions on other specialized or older datasets can be found in [30] [43] [39] and [29]. It is important to note that many existing network datasets suffer from inaccurate labelling and poor attack diversity [6]. Thus, results in this field must be taken with a grain of salt, with close attention being paid to the details of model implementation and validation.

3.3 Network anomaly detection

3.3.1 High performing models

There has been a significant amount of research done on anomaly detection that range the gamut of deep neural networking to fuzzy computing to information theory. Here, we highlight some of the techniques that have the highest performance or that are most similar to our models for comparison.

The creators of the CICIDS 2017 dataset ran some classical out-of-box machine learning techniques on the dataset as a set of benchmarks [40]. Their findings are summarized in table 3.7. It is interesting to note that KNN, RF, and ID3 far outperform AdaBoost on the dataset.

Algorithm	Precision	Recall	F1
KNN	0.96	0.96	0.96
RF	0.98	0.97	0.97
ID3	0.98	0.98	0.98
AdaBoost	0.77	0.84	0.77
MLP	0.77	0.83	0.76
Naive-Bayes	0.88	0.04	0.04
QDA	0.97	0.88	0.92

Table 3.7: Benchmark model performance metrics from the creators of CICIDS 2017

Table summarizing the findings in [40] that shows the performance results for the K-Nearest Neighbors (KNN), Random Forest (RF), ID3 Decision Tree, Adaboost, Multilayer perceptron (MLP), Naive-Bayes (NB), Quadratic Discriminant Analysis (QDA) on the CICIDS 2017 dataset.

In late 2018, Ahmim et al. built upon his earlier research on a binary tree of classifiers [44] and implemented a novel hierarchical model using three distinct classifiers: one for binary anomaly detection, one for multiclass classification, and a third one that uses the output of the former two in making a final predictor [45]. They divided the dataset into two sets of 40,000 flows for training and testing. Each set had 20,000 benign and 20,000 attack flows, with each of the attacks represented in both train and test data. They were able to obtain an overall accuracy of .967 with a detection rate (1 - false negative rate) of .944.

In April 2019, Vinayakumar and his team implemented a number of deep neural networks (DNNs) across a number of datasets including KDD99, NSL-KDD, UNSW-NB15, and CICIDS 2017 datasets [42]. They determined the optimal hyperparamters and network topology of the DNN purely by looking at the KDD99 dataset. Ultimately, they provided various model results for DNNs of varying fully connected layers using a ReLU activation function for both binary classification (combining the attack classes) and multiclass classification problems. Futhermore, they ran test results of clasical machine learning classifiers as a comparison benchmark. Their highest performing proposed DNN results and highest performing classical machine learning classifiers are summarized and extracted into Table 3.8. For the binary classification task, the classical machine learning techniques performed better in the NSL-KDD and UNSW-NB15 in nearly every regard - especially tree-based ensemble classifiers. However, the researchers found that the DNNs exhibited better performance in the CICIDS 2017 dataset and for multiclass classification problem where there was a prevalence of imbalanced classes.

Dataset (architecture)	Accuracy	Precision	Recall	F1
Novel DNN Architectures				
NSL-KDD (DNN 1-layer)	0.801	0.692	0.969	0.807
UNSW-NB15 (DNN 1-layer)	0.784	0.944	0.725	0.820
CICIDS 2017 (DNN 1-layer)	0.963	0.908	0.973	0.939
Classical Machine Learning				
NSL-KDD (Decision Tree)	0.930	0.928	0.943	0.935
UNSW-NB15 (Random Forest)	0.903	0.988	0.867	0.924
CICIDS 2017 (Random Forest)	0.940	0.849	0.969	0.905

Table 3.8: Selected performances in binary classification from Vinayakumar et al. on various datasets

Table summarizing the findings in [42] regarding their proposed DNN techniques as well as the best classical machine learning algorithms run on some datasets. Interestingly, the classical machine learning algorithms outperformed the deep learning techniques by a significant margin in the binary classification task.

Other research done in March 2019 on improving the performance of the AdaBoost algorithm on CICIDS 2017 used a combination of Synthetic Minority Oversampling Technique (SMOTE), Principal Component Analysis (PCA), and Ensemble Feature Selection (EFS) [46]. The utilized different approaches for features selection including identifying 16 features that accounted for the most variance according to PCA as well as another set of 25 features that were giving the highest weight importance in ensemble learning. Building on top of these features, the researchers were able to achieve a maximum accuracy, precision, recall, and F1 Score of 81.83%, 81.83%, 100%, and 90.01% respectively.

In June 2019, Chiba et al. proposed a combination of machine learning algorithms for anomaly detection [47]. They proposed a deep neural network whose hyperparameters were encoded into strings and used as chromosomes in an improved genetic algorithm to optimize the network parameters. After 200 generations, the network architecture with the highest fitness was determined using the AUC as a fitness function. On the They were able to achieve impressive results on the NSL-KDD, CICIDS 2017, and CIDDS-001 datasets, achieving an F1 score of .99 on all three of them with both false positive and false negative rate below 0.01 across all datasets. They created train and test sets for CICIDS 2017 in the same manner as [45] with 40,000 flows for train and test sets each.

In December 2019, further work was done to improve the performance of XGBoost on the UNSW-NB15 data [48]. Since XGBoost only support numeric values, all nominal values such as protocol type (e.g. TCP and UDP), the flow state (e.g ACC and CLO) and its service (e.g. HTTP and SSH) were not included in the analysis. Furthermore, the source and destination ports were not included in the analyses. The researchers ran a number of classical machine learning models on the data in comparison with XGBoost and found that Random Forest had a similar performance. XGBoost and Random Forest were able to obtain accuracies of 92% and 93% respectively.

3.3.2 Feature selection

Significant work has also been done to identify the key features within each dataset. The relevant papers found all focus on finding a minimal subset of relevant features that either preserve of enhance the performance of models. However, little work has been done to engineer new features or to determine the most ideal features across datasets.

In 2015, Moustafa and Slay used an association rule mining (ARM) algorithm to determine the strongest features of the KDD99 and UNSW-NB15 datasets [49]. They recreated the KDD99 dataset using the features in the UNSW-NB15 dataset and determined that the features encapsulated in the UNSW-NB15 feature set were stronger predictors than the original features in KDD99 for binary classifiers. Most notably, they highlighted that the features of the flow state, packet life duration, packet loss, the window advertisement size, the packet rate, and the mean packet size were all critical features.

In October 2019, Reis et al. reduced the feature set of CICIDS 2017 to 10 key features using permutation importance [50]. Reis et al. used three feature selection approaches, namely gini importance, permutation importance, and drop column importance to find the most important

features, and then utilized a mixture of RF, XGBoost, and LightGBM to evaluate the dataset. They found that the most important features included the the minimum flow interarrival time (IAT), initial window of bytes, the destination port, and the mean and minimum size of packets. Using the reduced feature set, the authors were able to achieve a high of 93% precision, 91% recall and 91.9% F1-score, on the multiclass classification formulation of the problem.

Binbusayyis and Vaiyapuri [51] also propose a feature a combination of ensemble methods for feature selection process consistency based feature selection (CBF), correlation based feature selection (CFS), information gain based feature selection (IG), and ReliefF Filter. Using these feature selection and weighting methods in conjunction with a RF classifier, they were able to achieve a maximum of .9993 accuracy, a .999 detection rate, and a .001 false alarm rate on the CICIDS 2017 dataset. They identified key features to include the destination port, flow duration, the flow packet and bit rate, and the mean flow IAT and its standard deviation.

3.4 Constraints of machine learning

Despite the huge variety of network anomaly detection techniques being researched, all machine learning methodologies must keep several key points in mind in order for to be functional in a real world implementation. According to one of the world's leading cybersecurity firms, Kaspersky Labs, network anomaly detection models must: (1) be interpretable, (2) have relatively few false positives, (3) adapt to counteractions, and (4) be trained on a large and comprehensive dataset [52]. These key points are touted not only in industry, but also in academic research environments [53].

3.5 Summary

This section first explored methods for formatting and creating network data before proceeding to highlight several recent and comprehensive datasets. Lastly, this section discussed several recent papers for applying machine learning to the network anomaly detection problem. With ample background knowledge of the topic and relevant research, the next section will detail how Malmenator approaches the anomaly detection problem.

Chapter 4

Methodology

4.1 Overview

This chapter covers in detail data preprocessing and analysis of the three selected datasets. From there, we discuss our methodology for model training and evaluation with regards to different metrics. For the engineering methodology of Malmenator, please refer to [3]

4.2 General methodology

Given that each of the benchmark netflow datasets contain many different engineered features, with only a small subset of them overlapping, we opted to push for research that would enable us to look into the types of features that are most relevant to discovering anomalies in the domain space rather than proposing novel architectures that have marginally better performance. To do this, we closely scrutinize the predictive accuracy of the models as well as the relative importance of their features.

4.3 Data processing

Unlike some of the prior work done to perform rigorous feature selection on each individual dataset, we opted to do as little feature selection as possible in order to better visualize the important features. instead, we only eliminated features with significant correlation (absolute value ¿ 0.95) to other features after normalization on a standardized normal distribution. The selection of ensemble decision tree models also made our research less reliant to heavy feature selection. For the NSL-

KDD dataset, little work was done except to use that as a benchmark comparison due to lack of access to the underlying pcap files. For the UNSW-NB15 dataset, we used both the suggested train/test datasets as well as engineered our own from the raw data for comparison. For the CICIDS 2017 dataset, since there is no recommended train/test split, we created our own. The following procedures were taken in the process of creating our own custom train/test datasets from UNSW-NB15 and CICIDS 2017:

1. *Dataset Cleaning*: Remove rows with missing elements. Correct cells with impossible values. Eliminate columns with the same value for all features

2. *Feature Engineering*: One-hot encode categorical variables with a dummy variable dropped to prevent multicollinearity. Pseudo-encode information properly (e.g. not treating ports as a continuous value)

3. *Feature Scaling & Correlation Detection*: Remove columns with correlation of R ¿ 0.95 to better isolate key features

4. *Train/Test Split*: Vary train test split to test hypothesis of number of flow instances requires to develop and accurate baseline. Can also vary between stratified and not stratified (i.e. whether or not to expose the training set to same distribution of attacks as test set)

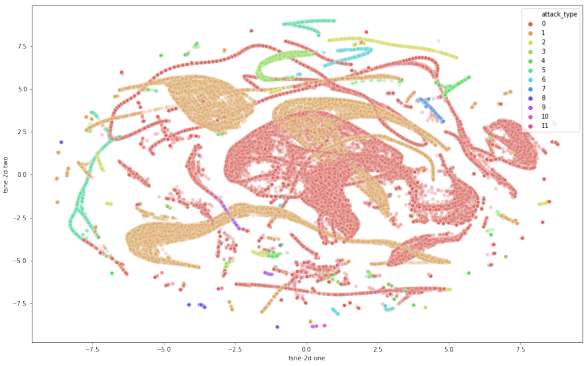
5. *Class Balancing*: Downsample the majority class, benign traffic, to enable the classifier to better learn from the features. Combine the minority attack classes for pure binary anomaly classification.

6. *Dataset Visualization*: Visualize the training data to see how well the features explain the variance within the data.

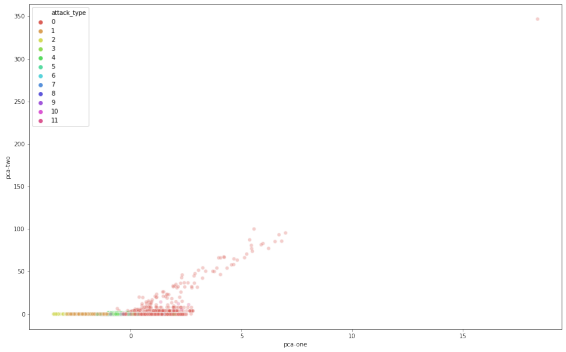
4.3.1 NSL-KDD

For the NSL-KDD dataset, minimal data preprocessing work was needed since it is a processed and cleaned subset of the underlying KDD99 dataset. There were no missing attributes, erroneous columns, or anything else that needed cleaning. The main steps that were performed for feature engineering was the one hot encoding of the 'service', 'protocol_type', and 'flag' features. Furthermore, the feature 'num_outbound_cmds' was removed due to being uniformly filled with 0. In addition, the following features were removed since they had a correlation value R ¿ 0.95 with other features in the dataset: 'num_root', 'srv_serror_rate', 'srv_rerror_rate', 'dst_host_serror_rate',

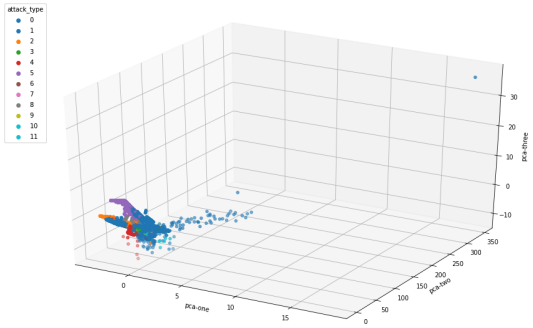
'dst_host_srv_serror_rate', 'dst_host_srv_rerror_rate', and 'flag_S0'. The modified training dataset in visualized in figure 4.1. It is evident that the decision boundaries between the classes are non-linear, but also that the features themselves do not adequately explain variance between the types of network flows.



(a) t-SNE



(b) 2-dimensional PCA



(c) 3-dimensional PCA

Figure 4.1: Visualization of NSL-KDD training set

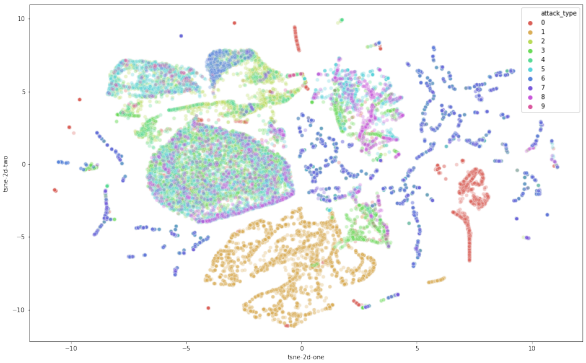
The three figures show different representations of of the high dimensional data space. The t-SNE shows that there is significant overlap between the benign flows (colored in red) with the attack classes. The same is true for the underlying PCA analysis where the feature space does not adequately represent the latent variance in the data.

4.3.2 UNSW-NB15

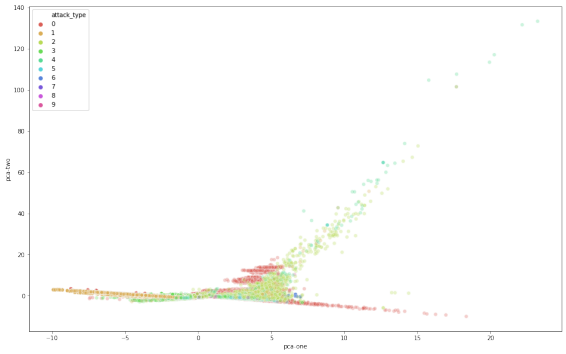
4.3.2.1 Presplit

UNSW-NB15 contains a proposed presplit train/test dataset. Similar to with NSL-KDD, we performed one hot encoding of the 'service', 'proto', and 'state' features and removed features with

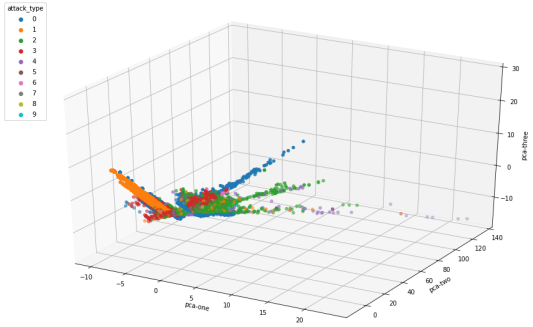
high correlation after scaling, namely: 'sbytes', 'dbytes', 'sloss', 'dloss', 'dwin', 'ct_src_dport_ltm', 'ct_dst_src_ltm', 'ct_ftp_cmd', 'ct_srv_dst', 'protocol_tcp', 'protocol_arp', 'state_FIN'. The t-SNE dimensional reduction technique as well as PCA were both run on the data to visualize the variability in the data that is accounted for by the features in figure 4.2, and it is clear there is a nice spread.



(a) t-SNE



(b) 2-dimensional PCA



(c) 3-dimensional PCA

Figure 4.2: Visualization of the UNSW-NB15 downsampled presplit training set

The figures depict different representations of of the high dimensional data space.for UNSW-NB15. This downsampled dataset has been feature engineered, and both the PCA and t-SNE show that although the data is not linearly separable, its latent vectors can account for a large amount of its variance

4.3.2.2 Custom Split

In addition to the presplit train/test data, the authors of UNSW-NB15 also made the raw netflow data available. This raw netflow data contains 2540047 flows, which is more than 10x more compared to 175341 flows in the presplit test dataset. Furthermore, this raw data contains

additional information such as the source and destination port addresses. Using this information, we engineered additional features for if a port was a "well known" port (i.e. range 0-1023) or if it was a "registered" port registered with the Internet Assigned Numbers Authority (IANA) or by Internet Corporation for Assigned Names and Numbers (ICANN). Registered ports range from 1024 to 49151. Like in the presplit data, columns that were filled entirely with the same value or which had exceedingly high correlation values to another column were removed. Columns which had more than 50% of their attributes empty were also removed, which includes 'ct_flw_http_mthd', 'is_ftp_login', and 'ct_ftp_cmd'. Note that all of these columns were filled in the presplit train/test set.

From here, we create our own enlarged stratified dataset. Stratified in this case means that both the test and train datasets contain the same distribution of normal and attack class types. The breakdown of classes can be found in 4.1. Here, we experimented with lowering the training set to ludicrously low levels to see if the models were robust enough to learn generalisable features using a combination of the original features as well as the engineered features.

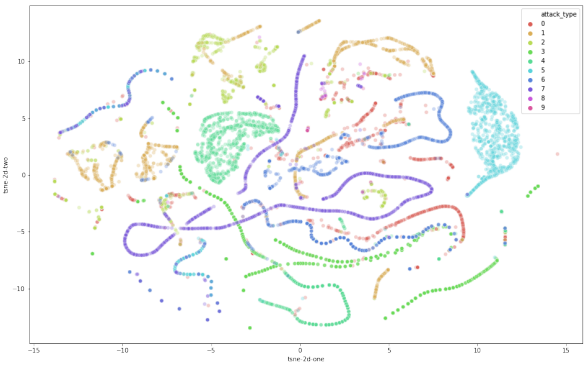
Split	1	2	3	4	5	6	7	8	9	**Attacks**	**Total**
UNSW-NB15 Stratified											
Train (70%)	150837	31167	16972	11447	9791	1874	1630	1058	122	**224898**	**574898**
Test (30%)	64644	13358	7274	4906	4196	803	699	453	52	**96385**	**246385**
UNSW-NB15 Stratified Small Train Proportion											
Train (10%)	21548	4452	2425	1635	1399	268	233	151	17	**32128**	**82128**
Test (90%)	193933	40073	40073	14718	12588	2409	2096	1360	157	**289155**	**739155**
UNSW-NB15 Stratified Tiny Train Proportion											
Train (5%)	10774	2226	1212	818	699	134	116	76	9	**16064**	**41064**
Test (95%)	204707	42299	23034	15535	13288	2543	2213	1435	165	**305219**	**780219**
UNSW-NB15 Stratified Miniscule Train Proportion											
Train (2%)	4309	890	485	327	280	54	47	30	3	**6425**	**16425**
Test (98%)	211172	43635	23761	16026	13707	2623	2282	1481	171	**314858**	**804858**

Table 4.1: Feature breakdowns for the different UNSW-NB15 test/train splits

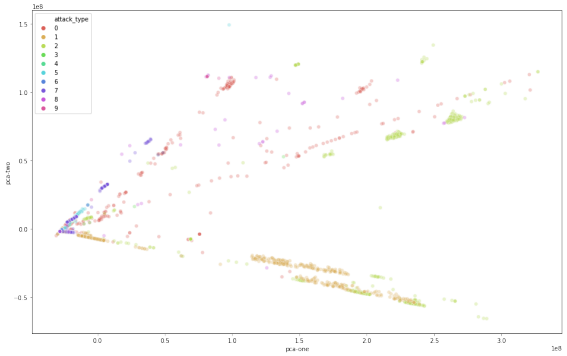
Table summarizing the feature breakdowns for the different custom splits. The attack type integer mappings are as follows: 1: Generic, 2:Exploit, 3:Fuzzer, 4:DoS, 5:Recon, 6:Analysis, 7:Backdoor, 8:Shellcode, and 9:Worm.

4.3.3 CICIDS 2017

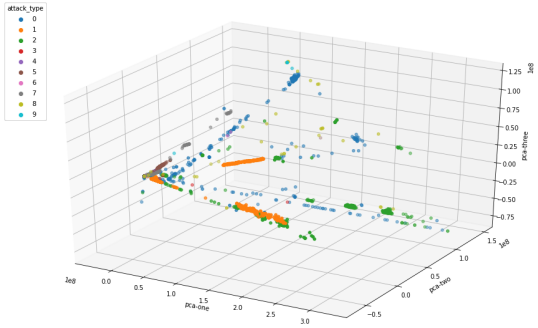
CICIDS 2017 did not have any presplit train/test sets, so we first created a split of the data similar to that found in [45] and [42] that consisted of equal sized training and test sets, each of which contained 40000 flows. Both testing and training sets consisted of 20000 attacks and 20000 benign flows.



(a) t-SNE



(b) 2-dimensional PCA



(c) 3-dimensional PCA

Figure 4.3: Visualization of a CICIDS 2017 downsampled training set

In the figures above, the PCA and t-SNE dimensionality reduction techniques show that the variance in the data is greatly explained through its features. Of the 3 datasets analyzed, its principal component account for the greatest amount of variance.

For creating our own custom feature set, we uncovered a number of issues that to our knowledge was not mentioned by other researchers. First, flow data from one of the component csv files had some values of 'Init_win_bytes_forward' set to -1. However, it is impossible to initiate a byte window of size -1, and it took a large amount of digging to discover to uncover that it was cause by a software issue with CICFlowmeter and should be set to 0 instead. Second, some rows contained negative segment lengths which is also impossible. These entries were removed from usage. Other rows containing NaNs and inf values were removed as they were in other papers. Lastly, empty columns and highly correlated columns after standardization were removed. Visualization of the dataset is shown in 4.3 and the class breakdowns of the train/test splits are in 4.2

Split	1	2	3	4	5	6	7	8	9	**Attacks**	**Total**
CICIDS 2017 Stratified											
Train (70%)	35000	35000	35000	5552	4126	1369	1526	25	5	**117603**	**397602**
Test (30%)	15000	15000	15000	2379	1769	587	654	10	2	**50401**	**170402**
CICIDS 2017 Stratified Small Train Proportion											
Train (10%)	5000	5000	5000	793	589	196	218	3	1	**16800**	**56800**
Test (90%)	45000	45000	45000	7138	5306	1760	1962	32	32	**151204**	**511204**
CICIDS 2017 Stratified Tiny Train Proportion											
Train (5%)	2500	2500	2500	396	295	98	109	2	0	**8400**	**28400**
Test (95%)	47500	47500	47500	7535	1858	2071	2213	33	7	**159604**	**539604**
CICIDS 2017 Stratified Miniscule Train Proportion											
Train (2%)	1000	1000	1000	159	118	39	43	1	0	**3360**	**11360**
Test (98%)	49000	49000	49000	7772	5777	1917	2137	34	7	**164644**	**556644**

Table 4.2: Feature breakdowns for the different CICIDS 2017 test/train splits

Table summarizing the feature breakdowns for the different custom splits. The attack type integer mappings are as follows: 1: DDoS, 2:DoS, 3:PortScan, 4:FTP-Patator, 5:SSH-Patator, 6:Bot, 7:Web Attack, 8:Infiltration, and 9:Heartbleed.

4.4 Model performance and feature importance

We fit the RF, XGBoost, and LightGBM ensemble learners across various splits of the datasets. Table 5.1 contains the model performance on the presplit NSL-KDD, presplit UNSW-NB15, and several custom and feature engineered splits of UNSW-NB15. The performance metrics for the CICIDS 2017 dataset are listed in table 5.2 and contains a variety of scenarios of train/test splits that include giving the ensemble a limited number of training samples and changing attack distributions between the train and test sets.

In addition, we are interested to see which features in each of the underlying datasets are the most pertinent to anomaly detection. To this end, for each of the models trained on each of the data partitions, we track them in table 5.4 for NSL-KDD and UNSW-NB15 and table 5.5 for CICIDS 2017.

4.5 Summary

In this chapter, we cover the process of feature engineering and data preprocessing that enables us to train computationally efficient, yet robust models that have comparable performance to state of the art models. Understanding the procedures done here are key to understanding the implications of the results

Chapter 5

Results and discussion

5.1 Overview

This chapter focuses on analyzing the results from the experiment. Here, we focus on the the performance and robustness of the models across the different train/test splits discussed in the methodology, the feature importances, and the important implications this has toward the future of feature engineering in anomaly detection.

5.2 Model performance metrics

The most critical aspect of creating an anomaly detection model is its evaluation - this is doubly true for the cybersecurity space for the reasons listed in the introduction. On the feature engineered UNSW-NB15 and CICIDS 2017 datasets, the model consistently is able to achieve over 99% on accuracy, precision, recall, f1 score, and detection rate (See tables 5.1 and 5.2). At the same time, the model regularly has less than a 0.5% false alarm rate. Both ensemble learners that used gradient boosting, XGBosst and LightGMB, exhibited similar performance metrics. The RF classifier was also similarly robust. On the UNSW-NB15 dataset, the RF model had noticeable fewer false negatives when exposed to a small training set, although the reverse was true on the CICIDS 2017 dataset.

The model's stable performance regardless independent of the training test size seem to show that the model is able to generalize well overfitting to the profiles of the different attack classes - given that there are sufficient features to learn from. It is interesting to note that the ensemble learners trained on the presplit training data set were outperformed by models exposed to an order

Model architecture	TP	TN	FP	FN	Accuracy	Precision	Recall	F1
NSL-KDD								
train=125973 (85%), test=22544 (15%)								
Random Forest	9448	7931	263	4902	0.771	0.973	0.658	0.785
XGBoost	9441	8331	270	4502	0.788	0.972	0.677	0.798
LightGBM	9436	8144	275	4689	0.780	0.972	0.668	0.792
UNSW-NB15 Presplit Train/Test								
train=175341 (68%), test=82332 (32%)								
Random Forest	27123	44692	9877	640	0.872	0.733	0.977	0.838
XGBoost	27520	44600	9480	732	0.876	0.744	0.974	0.844
LightGBM	27326	44687	9674	645	0.875	0.739	0.977	0.841
UNSW-NB15 Stratified								
train=574898 (70%), test=246385 (30%)								
Random Forest	148461	96065	1539	320	0.992	0.998	0.990	0.994
XGBoost	148465	96027	1535	358	0.992	0.998	0.990	0.994
LightGBM	148324	96131	1676	254	0.992	0.998	0.989	0.993
UNSW-NB15 Stratified								
Small Train Proportion: train=82128 (10%), test=739155 (90%)								
Random Forest	444397	288458	5603	697	0.991	0.998	0.988	0.993
XGBoost	444862	287708	5138	1447	0.991	0.997	0.989	0.993
LightGBM	444701	288217	5299	938	0.992	0.998	0.988	0.993
UNSW-NB15 Stratified								
Tiny Train Proportion: train=41064 (5%), test=780219 (95%)								
Random Forest	468768	304679	6232	540	0.991	0.999	0.987	0.993
XGBoost	469359	303699	5641	1520	0.991	0.997	0.988	0.992
LightGBM	469271	303914	5729	1305	0.991	0.997	0.988	0.992
UNSW-NB15 Stratified								
Miniscule Train Proportion: train=16426 (2%), test=804857 (98%)								
Random Forest	483382	314462	6601	413	0.991	0.999	0.987	0.993
XGBoost	484086	313105	5897	1770	0.990	0.996	0.988	0.992
LightGBM	484064	313385	5919	1490	0.991	0.997	0.988	0.992

Table 5.1: Model performance on NSL-KDD and UNSW-NB15 datasets

Table summarizing the prediction results for the models trained on different partitions of the NSL-KDD and UNSW-NB15 datasets. UNSW-NB15 Original refers to training on the provided test/train split with

Model architecture	TP	TN	FP	FN	Accuracy	Precision	Recall	F1
CICIDS 2017 Standard Split (w/ Feature Engineering) Stratified: train=40000 (%50), test=40000 (50%)								
Random Forest	19939	19918	61	82	0.996	0.997	0.996	0.996
XGBoost	19954	19963	46	37	0.998	0.998	0.998	0.998
LightGBM	19945	19966	55	34	0.998	0.997	0.998	0.998
CICIDS 2017 Stratified train=397602 (70%), test=170402 (30%)								
Random Forest	119862	50255	139	146	0.998	0.999	0.999	0.999
XGBoost	119866	50342	135	59	0.999	0.999	1.000	0.999
LightGBM	119829	50315	172	86	0.998	0.999	0.999	0.999
CICIDS 2017 Stratified Small Train Proportion: train=56800 (10%), test=511204 (90%)								
Random Forest	359361	150368	639	836	0.997	0.998	0.998	0.998
XGBoost	359463	359463	537	467	0.999	0.999	0.999	0.999
LightGBM	359377	150829	623	375	0.998	0.998	0.999	0.999
CICIDS 2017 Stratified Tiny Train Proportion: train=28400 (5%), test=539604 (95%)								
Random Forest	379249	158172	751	1432	0.996	0.998	0.996	0.997
XGBoost	379391	159087	609	517	0.998	0.998	0.999	0.999
LightGBM	379308	159114	692	692	0.997	0.998	0.998	0.998
CICIDS 2017 Stratified Miniscule Train Proportion: train=11360 (2%), test=556644 (98%)								
Random Forest	390976	162631	1024	2013	0.995	0.997	0.995	0.996
XGBoost	390939	163938	1061	706	0.997	0.997	0.998	0.998
LightGBM	390979	163880	1021	764	0.997	0.997	0.998	0.998

Table 5.2: Model performance on CICIDS 2017 datasets

Table summarizing the prediction results for the models trained on different partitions of the CICIDS 2017 datasets. CICIDS 2017 standard refers to the partitioning found in [45] and [42], while the CICIDS 2017 Custom is our own split.

of magnitude less data, but with engineered features.

The performance of our models are comparable to any state of the art model, including the advanced evolutionary neural network in [42] as well as other advanced models including those proposed in An Efficient Hybrid Self-Learning Intrusion Detection System Based on Neural Networks [58], Network Intrusion Detection System based PSO-SVM for Cloud Computing [57], An Enhanced Approach to Fuzzy C-means Clustering for Anomaly Detection [55], among others. Table 5.3 clearly shows the competitive performance of ensemble learners with appropriate feature engineering.

Model architecture	Year	Accuracy	Precision	Recall	F1
ANN [54]	2016	0.994	-	0.941	-
KNN [29]	2017	-	0.96	0.96	0.96
ID3 [29]	2017	-	0.98	0.98	0.98
Random Forest [29]	2017	-	0.98	0.97	0.97
AdaBoost [46]	2018	0.818	0.818	1.00	.901
Fuzzy C-means Clustering [55]	2018	0.953	0.995	0.958	-
Behavior Based ID [56]	2018	0.989	-	0.992	0.987
PSO-SVM [57]	2019	0.991	0.995	0.991	0.991
DNN [42]	2019	0.784	0.944	0.725	0.820
SOM + RBF + ANN [58]	2019	-	0.976	0.982	-
IGA-SA + DNN [47]	2019	0.999	0.999	0.999	0.999
Random Forest (Malmenator)	2020	0.998	0.999	0.999	0.999
XGBoost (Malmenator)	2020	0.999	0.999	1.000	0.999
LightGBM (Malmenator)	2020	0.998	0.999	0.999	0.999

Table 5.3: Model performance comparisons

Table comparing our model's performance on the feature engineered train (70%) / test (30%) CICIDS 2017 splits with other high performance models in the literature.

5.3 Feature importances and its implications

Interestingly, different ensemble models appear to favor different features. For example, on CICIDS 2017, LightGBM consistently looks at 'Init_Win_bytes_forward' and 'Init_Win_bytes_backward' regardless of whether it is exposed to 2% or 70% of the dataset. On the other hand, XGBoost is able to achieve similar performance by looking at attributes relating to the length and size of packets. There is quite a large amount of variation with regards to the important features on CICIDS 2017. However, it is important to note that they all relate to derived statistical features about the packets sent.

In UNSW-NB15, we see a large amount of converge regarding the important features. Both RF and XGBoost look heavily at 'sttl' and 'ct_state_ttl', both of which directly relate to the number of stops a packet makes as it travels from one computer to another. On the other hand, LightGBM looks heavily on the mean size of the packets as well as the total number of bytes that are sent. Although the features these ensembles select are quite different, they are still able to achieve similar performance metrics.

As for NSL-KDD, all of the ensembles ultimately rely significantly on the number of bytes sent from the source to the destination. Nearly all of the revelant features that enabled the ensemble learners to have a robust performance in UNSW-NB15 and CICIDS 2017 are not present in NSL-

KDD except for the number of bytes sent from the source to destination. Many of the features in NSL-KDD pertain to the number of attempts at making connections as well as the types of connection, which do not appear to be as highly relevant to anomaly detection as one may initially suppose. This is reflected by an overall much lower performance of models on the NSL-KDD dataset when compared to models that utilize the features of UNSW-NB15 and CICIDS 2017.

Model architecture	Top features
NSL-KDD train=125973 (85%), test=22544 (15%)	
Random Forest	src_bytes, dst_bytes
XGBoost	service_ecr_i, service_http, src_bytes,
LightGBM	src_bytes, dst_bytes, duration
UNSW-NB15 Presplit Train/Test train=175341 (68%), test=82332 (32%)	
Random Forest	ct_state_ttl, sttl
XGBoost	sttl
LightGBM	smean, sbytes
UNSW-NB15 Stratified train=574898 (70%), test=246385 (30%)	
Random Forest	ct_state_ttl, sttl
XGBoost	sttl, ct_state_ttl,
LightGBM	smeansz, sbytes, dbytes
UNSW-NB15 Stratified Small Train Proportion: train=82128 (10%), test=739155 (90%)	
Random Forest	ct_state_ttl, sttl
XGBoost	sttl, ct_state_ttl,
LightGBM	smeansz, sbytes, dtcpb
UNSW-NB15 Stratified Tiny Train Proportion: train=41064 (5%), test=780219 (95%)	
Random Forest	ct_state_ttl, sttl
XGBoost	sttl, ct_state_ttl,
LightGBM	smeansz, sbytes, dtcpb
UNSW-NB15 Stratified Miniscule Train Proportion: train=16426 (2%), test=804857 (98%)	
Random Forest	ct_state_ttl, sttl
XGBoost	sttl, ct_state_ttl
LightGBM	dmeansz, smeansz, Sload, sbytes

Table 5.4: Feature importance for models trained on the NSL-KDD and UNSW-NB15

Table summarizing the two most important features for each of the ensemble learners in the dataset. Looking at the important features enables us to continue to engineer more relevant feature columns in the future.

It is also worthwhile to note that the most important features in the presplit UNSW-NB15 dataset are the same as those in the feature engineered UNSW-NB15 dataset. However, the engineered

Model architecture	Top features
CICIDS 2017 Standard Split (w/ Feature Engineering)	
Stratified: train=40000 (%50), test=40000 (50%)	
Random Forest	Init_Win_bytes_forward, Packet Length Variance, Fwd Packet Length Min
XGBoost	Bwd Packet Length Min, Max Packet Length
LightGBM	Init_Win_bytes_forward, Init_Win_bytes_backward, Flow IAT Min
CICIDS 2017 Stratified	
train=397602 (70%), test=170402 (30%)	
Random Forest	Packet Length Variance, Packet Length Std, Avg Bwd Segment Size
XGBoost	Average Packet Size, Bwd Packet Length Std
LightGBM	Init_Win_bytes_forward, Init_Win_bytes_backward
CICIDS 2017 Stratified	
Small Train Proportion: train=56800 (10%), test=511204 (90%)	
Random Forest	Packet Length Variance, Packet Length Std, Avg Bwd Segment Size
XGBoost	Average Packet Size, Bwd Packet Length Std, Bwd Header Length
LightGBM	Init_Win_bytes_forward, Init_Win_bytes_backward
CICIDS 2017 Stratified	
Tiny Train Proportion: train=28400 (5%), test=539604 (95%)	
Random Forest	Packet Length Variance, Bwd Packet Length Std, Average Packet Size
XGBoost	Average Packet Size, Bwd Packet Length Std, Max Packet Length
LightGBM	Init_Win_bytes_forward, Init_Win_bytes_backward, Flow IAT Min
CICIDS 2017 Stratified	
Miniscule Train Proportion: train=11360 (2%), test=556644 (98%)	
Random Forest	Init_Win_bytes_forward, Packet Length Variance, Fwd Packet Length Min
XGBoost	Bwd Packet Length Min, Max Packet Length
LightGBM	Init_Win_bytes_forward, Init_Win_bytes_backward, Flow IAT Min

Table 5.5: Feature importance for models trained on the CICIDS 2017 dataset

Table summarizing the two most important features for each of the ensemble learners in CICIDS 2017 depending on partitions. Looking at the important features enables us to continue to engineer more relevant feature columns in the future.

features enabled the classifiers to make more fine tuned and accurate decisions. Access to an increased number of relevant features provides a similar performance boost to state of the art models and is far more computationally efficient and practical in real-world scenarios.

5.4 Future Works

The results of this paper open up a new direction in the field of network anomaly detection research. That is, what features can be captured from network traffic that are best able to highlight the behavior of malicious traffic? Until now, the collection and conversion of benchmark dataset pcap files into netflow data has been a process largely determined by the team proposing a new dataset.

These teams ultimately determine the relevant features to compute and include in the flow data, which leads to wide discrepancies in the features of different datasets. It would be a monumental, yet worthwhile, work to reconvert the source pcap files for various benchamrk anomaly detection datasets into network flow data with a superset of features from the different datasets available today. This would allow researchers to have a clearer insight in to the most relevant features that exist in reality, and not just the most relevant features in a given dataset.

5.5 Challenges

5.5.1 Understanding the domain-specific data

As can be seen with the arduous task of appropriate feature engineering, a strong base of understanding the computer network and security is required to carry out this task. However, all member of the team lacked of prior knowledge in the cybersecurity field. This lead to two major issues. First, we initially set unrealistic expectations for our project scope which required us to reevaluate our decision and narrow our scope several times. Second, our approach to solving this problem was initially viewed from a more traditional lens without the domain knowledge required for us to make the breakthrough we did. This results in a number of delays to our project.

5.5.2 Scope identification

Cybersecurity is a broad domain, and this is our teams first foray into the field. This made it difficult to limit the scope of the project initially due to a lack of foresight. This was also true in the area of research particularly, as our initial approach was to design an extremely complex model to achieve high performance. Our methodology on changed once we began deeply analyzing the features and asking ourselves why they were so limited and different between different datasets.

Conclusion

This paper closely examines the performance of ensemble learners on the NSL-KDD, UNSW-NB15, and CICIDS 2017 datasets. By using appropriate feature engineering techniques to encode nominal features such as the protocol type and the destination port, we were able to utilize out of the box ensemble learners to achieve a breakthrough state of the art performance with minimal computational cost. Furthermore, we show that ensemble learners such as RF, XGBoost, and LightGBM are robust enough have extremely high performance (99% accuracy, precision, detection rate, and f1 score) even when trained on minimal training datasets. The results of this research have deep implications for future work around building and implementing anomaly detection systems. We propose that the key to an effective, scalable, and generalizable model is discovering what underlying network features are most useful for gaining insights into malicious network behaviour. The next meaningful breakthroughs in network anomaly detection models will not come from building more complex models that may have marginal improvements in accuracy on flawed data, but rather from engineering relevant features on top of trustworthy data.

Bibliography

[1] The White House, "The cost of malicious cyber activity to the u.s. economy," tech. rep., The Council of Economic Advisors, Feb 2018.

[2] Forgerock, "U.s. consumer data breach report 2019: Personally identifiable information targeted in breaches that impact billions of records," tech. rep., Forgerock, 2019.

[3] Jha, "Malmenator: Intrusion detection architecture," tech. rep., The University of Hong Kong, May 2020.

[4] A. Fogg, "Anthony goldbloom gives you the secret to winning kaggle competitions," Jan 2016.

[5] L. Liu, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences: Machine Learning for Cybersecurity Threats, Challenges, and Opportunities*, vol. 9, Oct 2019.

[6] N. Moustafa, J. Hu, and J. Slay, "A holistic review of network anomaly detection systems: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 128, p. 3355, Feb 2019.

[7] G. Fernandes, J. J. P. C. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proena, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, p. 447489, Jul 2018.

[8] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, p. 1931, Jan 2016.

[9] J. K. K. M. H. Bhuyan, D. K. Bhattacharyya, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 303–336, Jan 2014.

[10] J. Quinlan, "Simplifying decision trees," *International Journal of Man-Machine Studies*, vol. 27, no. 3, p. 221234, 1987.

[11] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[12] K. A. Grajski, L. Breiman, G. V. D. Prisco, and W. J. Freeman, "Classification of eeg spatial patterns with a tree-structured methodology: Cart," *IEEE Transactions on Biomedical Engineering*, vol. BME-33, no. 12, p. 10761086, 1986.

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

[14] H. Sharma and S. Kumar, "A survey on decision tree algorithms of classification in data mining," *International Journal of Science and Research (IJSR)*, vol. 5, p. 20942097, Apr 2016.

[15] T. G. Dietterich, "Ensemble methods in machine learning," *Proceedings of the First International Workshop on Multiple Classifier Systems*, Jun 2000.

[16] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, p. 123140, 1996.

[17] L. Breiman, "Heuristics of instability and stabilization in model selection," *The Annals of Statistics*, vol. 24, p. 23502383, Jun 1994.

[18] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, p. 197227, 1990.

[19] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, Sep 1999.

[20] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, p. 11891232, 2001.

[21] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *Ann. Statist.*, vol. 28, pp. 337–407, 04 2000.

[22] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics*, vol. 7, Dec 2013.

[23] T. K. Ho, "Random decision forests," *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Aug 1995.

[24] G. Biau, "Analysis of a random forests model," *The Journal of Machine Learning Research*, vol. 13, Apr 2012.

[25] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *Journal of Machine Learning Research*, vol. 15, no. 90, pp. 3133–3181, 2014.

[26] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Mar 2016.

[27] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *31st Conference on Neural Information Processing Systems*, 2017.

[28] R. Blagus and L. Lusa, "Smote for high-dimensional class-imbalanced data," *BMC Bioinformatics*, vol. 14, Mar 2013.

[29] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, vol. 2017, p. 177200, Jul 2017.

[30] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, p. 147167, 2019.

[31] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[32] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. Mcclung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and et al., "Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation," *Proceedings DARPA Information Survivability Conference and Exposition*, Jan 2000.

[33] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, p. 579595, 2000.

[34] P. Gogoi, M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Packet and flow based network intrusion dataset," *Communications in Computer and Information Science Contemporary Computing*, p. 322334, 2012.

[35] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," *Recent Advances in Intrusion Detection*, p. 220237, 2003.

[36] J. Mchugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security*, vol. 3, p. 262294, Nov 2000.

[37] A. Vasudevan, E. Harshini, and S. Selvakumar, "Ssenet-2011: A network intrusion detection system dataset and its comparison with kdd cup 99 dataset," *2011 Second Asian Himalayas International Conference on Internet (AH-ICI)*, 2011.

[38] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009.

[39] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," *Military Communications and Information Systems Conference (MilCIS)*, Nov 2015.

[40] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, Jan 2018.

[41] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An evaluation framework for intrusion detection dataset," in *2016 International Conference on Information Science and Security (ICISS)*, pp. 1–6, 2016.

[42] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, no. 41, pp. 525–550, 2019.

[43] G. Maci-Fernndez, J. Camacho, R. Magn-Carrin, P. Garca-Teodoro, and R. Thern, "Ugr16: A new dataset for the evaluation of cyclostationarity-based network idss," *Computers & Security*, vol. 73, p. 411424, 2018.

[44] A. Ahmim and N. G. Zine, "A new hierarchical intrusion detection system based on a binary tree of classifiers," *Information and Computer Security*, vol. 23, p. 3157, Mar 2015.

[45] 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 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Dec 2018.

[46] A. Yulianto, P. Sukarno, and N. A. Suwastika, "Improving AdaBoost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset," *Journal of Physics: Conference Series*, vol. 1192, pp. 012–018, mar 2019.

[47] Z. Chiba, N. Abghour, K. Moussaid, A. E. Omri, and M. Rida, "Intelligent approach to build a deep neural network based ids for cloud environment using combination of machine learning algorithms," *Computers & Security*, vol. 86, pp. 291–317, jun 2019.

[48] A. Husain, A. Salem, C. Jim, and G. Dimitoglou, "Development of an efficient network intrusion detection model using extreme gradient boosting (xgboost) on the unsw-nb15 dataset," in *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pp. 1–7, 2019.

[49] N. Moustafa and J. Slay, "The significant features of the unsw-nb15 and the kdd99 data sets for network intrusion detection systems," in *4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pp. 25–31, 11 2015.

[50] B. Reis, E. Maia, and I. Praa, "Selection and performance analysis of cicids2017 features importance," *Foundations and Practice of Security Lecture Notes in Computer Science*, p. 5671, Oct 2019.

[51] A. Binbusayyis and T. Vaiyapuri, "Identifying and benchmarking key features for cyber intrusion detection: An ensemble approach," *IEEE Access*, vol. 7, no. 10, pp. 6495–6513, 2019.

[52] Kaspersky Lab, "Machine learning in malware detection," tech. rep., Kaspersky, 2019.

[53] S. Saad, W. Briguglio, and H. Elmiligi, "The curious case of machine learning in malware detection," *Proceedings of the 5th International Conference on Information Systems Security and Privacy*, Feb 2019.

[54] N. Lokeswari and B. Rao, *Artificial Neural Network Classifier for Intrusion Detection System in Computer Network*, pp. 581–591. 01 2016.

[55] R. Sharma and S. Chaurasia, "An enhanced approach to fuzzy c-means clustering for anomaly detection," in *Proceedings of First International Conference on Smart System, Innovations and Computing* (A. K. Somani, S. Srivastava, A. Mundra, and S. Rawat, eds.), (Singapore), pp. 623–636, Springer Singapore, 2018.

[56] K. K. Ghanshala, P. Mishra, R. C. Joshi, and S. Sharma, "Bnid: A behavior-based network intrusion detection at network-layer in cloud environment," in *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pp. 100–105, 2018.

[57] M. M. Sakr, M. A. Tawfeeq, and A. B. El-Sisi, "Network intrusion detection system based pso-svm for cloud computing," *International Journal of Computer Network and Information Security*, vol. 11, p. 2229, Mar 2019.

[58] S. Mohammadi and F. Amiri, "An efficient hybrid self-learning intrusion detection system based on neural networks," *International Journal of Computational Intelligence and Applications*, vol. 18, no. 01, 2019.