Advancing IoT Security with Tsetlin Machines: A Resource-Efficient Anomaly Detection Approach

HENNING BLOMFELDT THORSEN, OLE GUNVALDSEN

SUPERVISOR
Per-Arne Andersen

University of Agder, 2021
Faculty of Engineering and Science
Department of Engineering and Sciences

Master

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	**Ja** / Nei
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	**Ja** / Nei
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	**Ja** / Nei
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	**Ja** / Nei
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	**Ja** / Nei
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	**Ja** / Nei
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Ja / **Nei**

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	**Ja** / Nei
Er oppgaven båndlagt (konfidensiell)?	Ja / **Nei**
Er oppgaven unntatt offentlighet?	Ja / **Nei**

Acknowledgements

We extend our deepest gratitude to Associate Professor Per-Arne Andersen for his assistance during the course of the project, for giving leads on studies worth taking a closer look at, for helping with the publication of the results, for feedback on the report, and for helping us form the framework for the thesis project.

Abstract

The number of IoT devices are rapidly increasing, and the nature of the devices leave them vulnerable to attacks. As of today there are no general security solutions that meet the requirements of running with limited resources on devices with a large variety of use cases. Traditional AI models are able to classify and distinguish between benign and malignant network traffic. However, they require more resources than IoT devices can provide, and cannot train on-chip once deployed. This thesis introduces the Tsetlin Machine as a potential solution to this problem. As a binary, propositional logic model, the Tsetlin Machine is compatible with hardware and can perform predictions in near real-time on limited resources, making it a suitable candidate for intrusion detection in IoT devices.

To assess the viability of the Tsetlin Machine as an IDS, we developed custom data loaders for the benchmark datasets: CIC-IDS2017, KDD99, NSL-KDD, UNSW-NB15, and UNSW-Bot-IoT. We ran hyperparameter searches and numerous experiments to determine the performance of the Tsetlin machine on each dataset. We discovered that preprocessing data by converting each data value to a 32-bit binary number and imposing an upper bound on class sizes proved to be an effective strategy. Furthermore, we compared the performance of the Tsetlin Machine against various classifiers from the scikit-learn library and lazy predict. The results show that the Tsetlin Machine's performance was on par with, if not superior to, other machine learning models, indicating its potential as a reliable method for anomaly detection in IoT devices. However, future work is required to determine its viability in a real-life setting, running on limited resources and classifying real-time data.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Introduction

Internet of Things (IoT) is the integration of millions of devices into networks in order to deliver and maintain intricate and comprehensive solutions for users[1]. The range and variation of use cases make the devices diverse and complex, and a single security solution has not yet been discovered that can cover the wast attack area. As a result the IoT devices lack security measures, and pose risks for end users and distributors. As an example a self driving car, if compromised, could destroy property and even cause the loss of human life. A smart stove could be made to burn down a house. If the device handles personal data, the privacy of the owner could be breached, leading to identity theft, blackmail and extortion.

1.2 Motivation

The Tsetlin Machine is a relatively new artificial intelligence algorithm that has the potential to outperform conventional algorithms like neural networks. This is especially relevant in environments with limited hardware resources, as the Tsetlin Machine operates with binary values, making it less resource-intensive. In addition the Tsetlin Machine is in development, meaning that all testing and research on new subjects is furthering the understanding of how the algorithm works, and how different hyperparameters and datasets affects the performance of the model.

The motivation behind choosing the Tsetlin Machine and anomaly detection in IoT devices is to be a part of the research and development of the Tsetlin Machine and push the understanding of the algorithm. By applying this research to anomaly detection, we might be one step closer to a more secure and stable internet of things.

1.3 Thesis Definition

This thesis seeks to find the answer the following question:

How can Tsetlin Machines be effectively trained to detect anomalies in IoT device data streams, and if so, how does the performance compare to that of other models?

For this project, there are a few starting hypotheses that will be tested:

- *The Tsetlin Machine can provide a high accuracy on real IoT network traffic.*
- *The Tsetlin Machine can perform as well or better than other machine learning methods when classifying IoT traffic.*

1.4 Goals

The goal of this thesis is to evaluate the anomaly detection performed by a Tsetlin Machine and compare these results to other state-of-the-art algorithms. This goal will be achieved through thorough testing of the Tsetlin Machine and the various hyperparameters available for tuning. The testing will be performed on the latest datasets concerning network traffic with anomalies.

Goal 1: Achieve a functioning Tsetlin Machine and test it on one dataset.

Goal 2: Implement remaining data sets for further testing and evaluation.

Goal 3: Optimize hyperparameters to maximize the performance of the model for each dataset.

Goal 4: Compare the results from the Tsetlin Machine implementation with other AI/ML models.

1.5 Field of Research

This thesis concerns the fields of Artificial Intelligence and Machine Learning, Cybersecurity, and IoT and Embedded systems.

1.6 Contributions

This thesis contributes to the Tsetlin Machine research community by expanding upon the work of Darshana et al[2] by performing an empirical study of the Tsetlin Machine's capabilities on a total of five different datasets. Furthermore, we compare the Tsetlin Machine's performance on these datasets to a wide array of other models from LazyPredict and Scikit Learn Classifier Comparison, providing a deep understanding of the Tsetlin Machine's capabilities within IoT Intrusion Detection. Another important contribution is the evaluation of the importance of the various hyperparameters and their effects on the performance of the Tsetlin Machine on each dataset. The thesis also provides a reliable method for balancing datasets by applying an upper bound to class sizes. Finally, the thesis contributes custom data loaders for the five datasets to the official Tsetlin Machine library - TMU (https://github.com/cair/tmu), enabling further research on the datasets.

1.7 Thesis structure

We structure the thesis as follows:

- Chapter 2 provides the theoretical background of the report, and introduces the Tsetlin Machine and its workings, the concept of Internet of Things (IoT) and why it is an important subject for security, and the different datasets that were used, and the details surrounding them.
- Chapter 3 contains rescent studies in the field of AI and internet of things, as well as some of the current solutions for intrusion detection and prevention.
- Chapter 4 details the experimental results achieved during this project, and compares them to the results of other AI models. The results are divided into sections for each dataset.

- Chapter 5 takes a deeper dive into the results and discusses their implications as well as the troubles faced during development and testing, and proposes further research into AI in IoT Security.
- Chapter 6 summarizes the project and the major findings.

Chapter 2

Theoretical Background

This chapter aims to introduce key elements in this thesis. First we will introduce the relatively new Tsetlin Machine, and how this novel machine learning algorithm uses propositional logic to classify data. We then turn our attention to the internet of things and how the growing ecosystem of devices, and lack of security within, poses a risk to end users and providers. Lastly we will cover the features of the datasets used for training and testing in this thesis.

2.1 Tsetlin Machines

To fully understand how the Tsetlin Machine algorithm works, insight into the Tsetlin Automata is required. The Tsetlin Automata(TA) is a finite state machine (FSM) where states are placed in linear branches to represent an action taken by the TA, typically saying *"True"* or *"False"*[3]. The transitions between states are based on a hidden, underlying reward probability distribution. When the TA finds itself on a given branch of states, it will only take the action those states represent, with the intent being that over the course of a large number of steps, the TA will find itself on the branch representing the action with the highest reward probability[3]. A visual representation of a Tsetlin automata can be seen in figure 2.1.

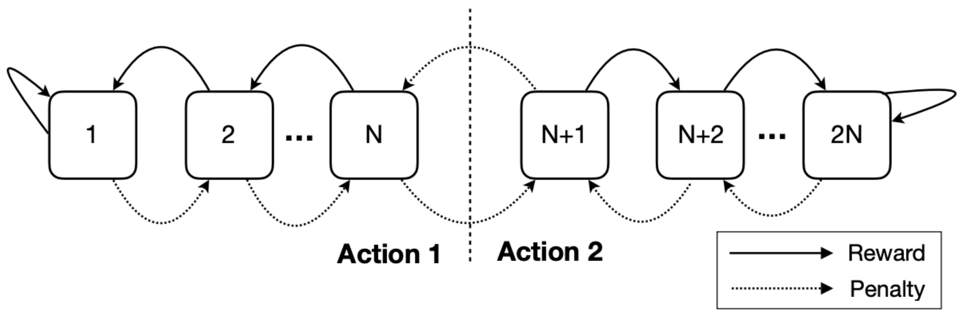

Figure 2.1: A Tsetlin Automata in a two action environment [4]

The Tsetlin Machine (TM) is a collection of TA's that evaluate data based on a set of binary values, each of which represent various boolean features in the data. These boolean features are called *literals*[3][5], and can be properties such as "has wheels" or "drives on roads" or "has wings", and one TA is responsible for each literal or its negation. The TM creates patterns of these literals based on observed data, and uses consensus among these patterns - also called *clauses*, or rules, to classify the data[3][5]. The TAs in a TM have the actions *"Include"* and *"Exclude"*, which govern whether a given literal should be a part of a clause or not[5]. During training, the TM observes a piece of data, and each clause makes a prediction[5]. The clauses then receive feedback from the TM based on whether they were correct or not. The feedback takes two forms; **Type I** and **Type II** feedback:

- **Type I feedback**, which can also be called positive feedback, is given to clauses that correctly classified the data. A clause that receives type I feedback will *increment* the TAs with the *Include* action with a high probability, and *decrement* the TAs with the *Exclude* action with a low probability, in order to positively reinforce the correct behaviour[5].

- **Type II feedback**, which can also be called negative feedback, is given to clauses that wrongly classified the data. A clause that receives type II feedback will with a high probability *decrement* the TAs with the *Include* action, and *increment* the TAs with the *Exclude* action in order to punish the wrong behaviour[5].

As the total clause output, defined as half the sum of includes plus half the negative sum of excludes, approaches a threshold T, the probability of reinforcement with both types of feedback decreases[5].

Once training is done, and the clauses are prepared, the TM can start classifying data. By comparing the literals in the incoming data to the literals in the clauses, each clause will give a vote towards a certain class, and once all the clauses have given their vote, the TM uses the majority rule to make a prediction for the data[3]. A visual representation of a Tsetlin Machine with two automata groups can be seen in figure 2.2.

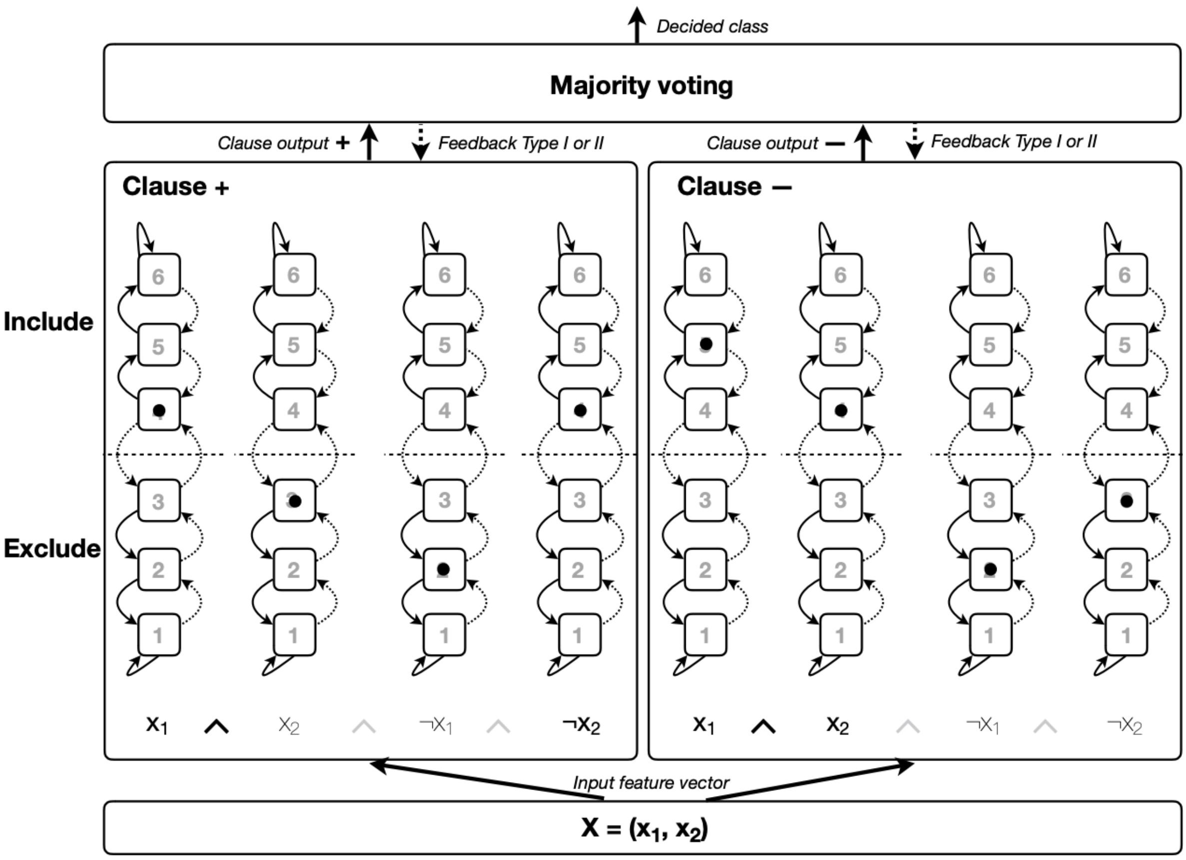

Figure 2.2: A Tsetlin Machine consisting of two groups of Tsetling automata [4]

The TM is capable of running on limited hardware, and does not require as much power or memory as neural networks or other AI/ML algorithms. This makes the Tsetlin Machine a good candidate for machine learning in Internet of Things[6].

2.2 Internet of Things

The term "Internet of Things" (IoT) was coined by Kevin Ashton in 1999, and originally described physical objects that were connected to the internet using sensors[7]. While the term is relatively new, the technology it describes has been used for decades. Today, the term is used to describe all devices that are connected to the internet[7]. Originally, Ashton made the term to explain the advantages of RFID-tags to track goods without human interaction[7]. Since then, IoT has steadily advanced, and commonplace items such as light bulbs, refrigerators, watches and doorbells, can be connected to the internet, in order to provide a better user experience. For example, the internet lets you see who is at the door from your couch, start a fresh pot of coffe on your way home from work, and has removed the need to manually configure clocks.

There are new uses for IoT discovered daily, and not just so-called smart-home devices. Wearable technology such as smart watches or health-related gadgets like glucose meters are also becoming more common.

While the technology solves a lot of problems and provides convenient services, it is not without risk. In 2017, the Norwegian Consumer Council (NCC) discovered an exploit in a brand of smart watches for children[8]. The exploit allowed malicious actors to acces the information sent and received by the smart watches, and could provide the location of children wearing the watches in real time[8].

In 2016, the french telecom company OVH were victims of a DoS-attack that was several times larger than similar threats[9]. This was the start of a wave of DoS-attacks that would go on to affect 175,000 websites over the course of a month. The attacks were caused by a massive DDoS-attack on a large DNS-host[9]. It was discovered that the attacks were caused by the Mirai Botnet, originally created by P. Jha, D. Norman, and J. White to scam and extort *Minecraft* server hosts[9]. The botnet spawned a number of variants after the source code was published online by an anonymous avatar thought to belong to Jha[9]. The Mirai Botnet was notably different from other botnets because it targetted IoT-devices as opposed to targeting computers[9]. Mirai serves as a prominent example of the harm that can arise from the lack of security in IoT-devices.
Each day, IoT devices are becoming more and more personal, which makes the security of these devices an increasingly important issue. New exploits and vulnerabilities are found every day, so it stands to reason that all the security challenges and issues with IoT devices is near impossible to cover in a single thesis.

2.3 Datasets

To evaluate the Tsetlin Machine's capabilities for Anomaly Detection, several datasets were trained and tested upon to give a good overview. This section details these datasets, including some of the data measures, labels, file structure and sources behind them. Part of the work done when testing the datasets was to create a custom data loader module for each dataset for the Tsetlin Machine Unified (TMU) library, maintained by the Centre for Artificial Intelligence Research (CAIR).

2.3.1 CIC-IDS2017

The CIC-IDS2017 dataset, provided by the University of New Brunswick, contains network traffic from a variety of common attack types as well as benign data[10]. The attack traffic was created by analyzing *.pcap-files* of network traffic where the attacks had taken place, and the benign data was created by profiling benign users and generating naturalistic behavior

Category	BENIGN	FTP-Patator	SSH-Patator	Bot	Infiltration	DoS slowloris	DoS Slowhttptest	Web Attack Sql Injection
Entries	2273089	7938	5897	1966	36	5796	5499	21

Category	DoS Hulk	DoS Goldeneye	Heartbleed	DDoS	PortScan	Web Attack Brute Force	Web Attack XSS
Entries	231073	10293	11	128027	158930	1507	652

Table 2.1: Data distribution of the CIC-IDS2017 dataset

data[10]. The attack data in the dataset contain entries from *Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS attacks*[10]. The dataset consists of 79 features, including Destination Port, Packet Flow Duration, Fwd and Bwd packet statistics, total flow duration, header lengths, packet sizes, and a variety of routing flags. The data is structured into seven *.csv-files*, each representing a certain day and time where real attacks were made, and for which naturalistic behaviour data was generated[10]. An example of how to use the dataloader created for CIC-IDS2017 can be seen in listing 2.1

```
CIC = tmu.datasets.CICIDS2017(self, split=0.7, shuffle=True, ...
    balance=True, binarize=True, bits_per_entry = 16, ...
    max_data_entries=450000, data_category_threshold = 5000)
paths = ["Data/"+i for i in os.listdir("Data")]
dataset = CIC.retrieve_dataset(paths)
```

Listing 2.1: Using CIC-IDS2017 data loader

The distribution between the different classes can be seen in table 2.1
The full dataset is available at: https://www.unb.ca/cic/datasets/ids-2017.html

2.3.2 KDD99

The KDD99 dataset is an improved version of the DARPA98 dataset [11], which was created by simulating network traffic in an U.S Air Force Lan environment. The dataset is provided by the Information and Irvine Computer Science University of California. The data set was used for The Third International Knowledge Discovery and Data Mining Tools Competition, where participants were tasked with building both a network intrusion detector and a predictive model to distinguish between good and bad connections. [12]. The dataset contains data entries from *normal* - benign data, as well as attacks from various categories and sources such as *warezclient* , *perl* , *ftp_write* , *neptune* , *rootkit* , *spy* , *land* , *multihop* , *teardrop* , *satan* , *smurf* , *normal* , *imap* , *loadmodule* , *pod* , *buffer_overflow* , *phf* , *ipsweep* , *guess_passwd* , *nmap* , *warezmaster* , *portsweep* and *back*. Among these, by far the largest categories are *normal, neptune, ipsweep and warezclient.*
The dataset consists of three main types of features:

- Basic features of individual TCP connections. These features contain information such as flow duration, internet protocol, number of bytes transferred and various flags[13].
- Content features within a connection suggested by domain knowledge. These features contain information such as number of failed login attempts, if root access was achieved, if a guest user was used, and more[13].
- Traffic features computed using a two-second time window. These features contain information such as total and server rates for SYN and REJ errors, total number of connections and number of servers connected to[13].

The complete distribution of the KDD99 dataset can be seen in table 2.2
The data was loaded using the dataset version hosted by OpenML

OpenML provides an API for loading the data directly from the code through the SKlearn-library as seen in listing 2.2:

Category	spy	guess_passwd	nmap	pod	warezmaster	neptune	ipsweep	ftp_write	buffer_overflow	rootkit	normal	multihop
Entries	2	53	231	264	20	107201	1247	8	30	10	97277	7

Category	loadmodule	imap	teardrop	land	smurf	perl	phf	back	satan	portsweep	warezclient
Entries	9	12	979	21	280790	3	4	2203	1589	1040	1020

Table 2.2: Data distribution of the KDD99 dataset

Category	guess_passwd	nmap	pod	warezmaster	neptune	ipsweep	ftp_write	buffer_overflow	rootkit	normal	loadmodule
Entries	1284	1566	242	964	45871	3740	11	50	23	77054	11

Category	imap	teardrop	land	smurf	perl	phf	back	satan	portsweep	multihop
Entries	12	904	25	3311	5	6	1315	4368	3088	25

Table 2.3: Data distribution of the NSL-KDD dataset

```
from sklearn.datasets import fetch\_openml
kdd = fetch\_openml(name='KDDCup99', version=1)
```

Listing 2.2: Downloading KDD99 dataset

The implementation in the TMU library can be loaded as such, with every variable in the function call being a parameter for how the data is preprocessed. The code for the implementation can be seen in listing 2.3

```
kdd = tmu.datasets.KDD99(split=0.7, shuffle=True, binarize=True, ...
    max_bits_per_literal=max_literals, balance=True, class_size_cutoff=500)
dataset = kdd.retrieve_dataset()
```

Listing 2.3: Using KDD99 data loader

2.3.3 NSL-KDD

The NSL-KDD dataset is an attempt to improve the KDD99 dataset by removing redundant and duplicate entries, as well as reducing the number of records to make it more affordable to run[14]. As NSL-KDD is a modified version of the KDD99 dataset, it contains the same type of data and classes, only less entries in total - see chapter 2.3.2. From KDD99, the NSL-KDD dataset has reduced the number of normal - benign data by 16.44%, and the number of attacks by 93.32% for the training set, and 88.26% for attacks and 20.92% for benign data in the test set[14]. The researchers at University of New Brunswick also trained 21 learners of unspecified nature on the dataset, and found an average accuracy of 98% for the training set, and 86% for the test set[14]. The NSL-KDD dataset is structured into a variety of files, some containing subsets of the data, and some containing the full set, which multiple files per subset - only in different file formats; .txt and .arff. There is also a summary of the dataset description in a separate file. The data distribution of the NSL-KDD dataset can be seen in table 2.3

Listing 2.4 shows how the NSL-KDD dataset can be loaded with every variable in the function call being a parameter for how the data is preprocessed. The loader requires the .csv-files be extracted into a subfolder called *NSL* in the working directory.

```
nsl = tmu.datasets.NSLKDD(shuffle = False, binarize = True, ...
    balance\_train_set = True, balance_test_set = True, ...
    max_bits_per_literal = 32, class_size_cutoff = 30000, ...
    limit_to_classes_in_train_set = True, limit_to_classes_in_test_set = True)
dataset = nsl.retrieve\_dataset()
```

Listing 2.4: Using NSLKDD data loader

Category	Normal	DoS	DDoS	Reconnaissance	Theft
Entries	477	1650259	1926622	91082	78

Table 2.4: Data distribution of the UNSW-Bot-IoT dataset

The dataset is available at https://www.unb.ca/cic/datasets/nsl.html

2.3.4 UNSW-NB15

The UNSW-NB15 dataset[15, 16, 17, 18, 19] contains 2,540,044 entries are stored in four csv files, as well as a subset of training and testing data with 175,341 and 82,332 entries respectively[20]. The data was created by using a tool called the *IXIA PerfectStorm*, which generated raw network packets described as *"a hybrid of real modern normal activities and synthetic contemporary attack behaviours"[20].* The packets were then captured using the *tcpdump* tool to generate 100GB of raw traffic. The dataset contains nine attack types, specifically *Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode* and *Worms*[20], as well as benign, normal traffic.

Listing 2.5 shows how UNSW NB15 can be loaded with every variable in the function call being a parameter for how the data is preprocessed. When calling the *retrieve_dataset*-function, make sure to pass a list of relative file paths to each of the four csv-files. In the example below, the .zip containing the dataset downloaded from the source was placed in a folder called *"UNSW"* before being extracted.

```
UNSWNB15 = tmu.datasets.UNSW_NB15(split=0.7, shuffle=True, balance=True, ...
    binarize=True, bits_per_entry = 32, max_data_entries=450000, ...
    data_category_threshold = class_size_cutoff)
paths = ["UNSW/UNSW-NB15 - CSV Files/UNSW-NB15_"+str(fi)+".csv" for fi in ...
    [1,2,3,4]]
dataset = UNSWNB15.retrieve_dataset(paths)
```

Listing 2.5: Using UNSW NB15 data loader

The dataset is available at https://research.unsw.edu.au/projects/unsw-nb15-dataset

2.3.5 UNSW-Bot-IoT

The UNSW-Bot-IoT dataset[21, 22, 23, 24, 25, 26] is supplied by the University of New South Wales, and contains more than 72.000.000 records of data from normal IoT traffic as well as various botnet attacks[27]. The data is spread across 74 .csv-files. The categories in the dataset include *Denial of Service (DoS), Distributed Denial of Service (DDoS), Normal - benign data from non-attacks, Reconnaissance and Theft*[27]. The dataset was created by simulating a realistic network in the Cyber Range Lab of UNSW Canberra. The traffic was then recorded into pcap files, resulting in 69.3 GB of raw .pcap files. After extracting the information the resulting .csv-files were 16.7 GB in size. To make handling the dataset easier, UNSW extracted the 5% subset from the data using SQL queries. The subset of the dataset is roughly 1GB, and is divided into four .csv-files[27]. The complete distribution of the dataset can be seen in table 2.4

Below is how the implementation in the TMU library can be loaded with every variable in the function call being a parameter for how the data is preprocessed. When calling the *retrieve_dataset*-function, make sure to pass a list of relative file paths to each of the four csv-files. In the example below, the .zip containing the dataset downloaded from the source was placed in a folder called *"Bot_IoT"* before being extracted.

```
BotIoT = tmu.datasets.Bot_IoT(split=0.7, shuffle=True, balance=True, ...
   binarize=True, bits_per_entry = 32, max_data_entries=450000, ...
   data_category_threshold = class_size_cutoff)
paths = ["Bot_IoT/Entire ...
   Dataset/UNSW_2018_IoT_Botnet_Dataset_"+str(fi)+".csv" for fi in ...
   list(range(1,75))]  # Use this for the entire dataset. Modify the ...
   range to select dataset files
paths = ["Bot_IoT/All ...
   features/UNSW_2018_IoT_Botnet_Full5pc_"+str(fi)+".csv" for fi in ...
   list(range(1,5))]  # Use this for the 5% subset dataset
dataset = BotIoT.retrieve_dataset(paths)
```

Listing 2.6: Using Bot_IoT data loader

The dataset is available at https://research.unsw.edu.au/projects/bot-iot-dataset

Chapter 3

State of the art

This chapter aims to inform the reader of the latest research within anomaly detection in IoT devices using AI. This includes recent studies in the field of network security and machine learning as well as conventional intrusion detection and prevention systems. In addition the most well known systems for intrusion detection and prevention will be described in short.

3.1 Recent Studies

This section dives into other studies into IoT anomaly detection using AI and similar techniques. Given the rising presence of IoT devices, and the data they gather about our day to day patterns and routines, IoT device security is an important and new subject. Many of the active Iot devices are poorly protected, and attackers might have an easy way into any users life [28]. This has resulted in a number of AI implementations aimed at protecting devices from attacks, some of them are described below. Some models, particularly those based on Neural Networks, are, despite boasting high performance when classifying IoT data, unfit for deployment in IoT devices. Neural Networks are notoriously resource-intensive, and are difficult to scale. They also need large amounts of memory and computational power to train[6], something not found in the typical IoT-device.

3.1.1 Kitsune

Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai released a paper titled "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection" in 2018[29], where they sought to evaluate the use of autoencoders to handle the challenge of IoT security. In their study, they created a model where multiple autoencoder neural networks make predictions on the data, and a consensus among the autoencoders is used to make the actual prediction[29]. For the study, they used data prepared specifically for the study by an expert, and after training their model on the data, they uploaded their model to Network-based Intrusion Detection Systems (NIDS) and tested against multiple forms of attack, including Man-in-the-Middle, Denial-of-Service, Recon and Botnet Malware[29]. By tweaking a threshold cutoff value, they were able to get a False Positive Rate (FPR) of as low as 0, but they measured False Negative Rates (FNR) and FPRs of both 0 and 0.001[29]. The authors compared their approach to several other intrusion detection methos, including Suricata, Iso. Forest, GMM, GMM Inc. and pcStream, and in all cases, Kitsune performed comparably or better [29].

3.1.2 Hades-IoT

In 2019, D. BreitenBacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer and Y. Elovici released a paper titled "HADES-IoT: A Practical and Effective Host-Based Anomaly Detection System for IoT Devices"[30]. Hades in this context is short for Host-bAsed DEtection System. Hades-IoT has two modes of operation; Profiling Mode and Enforcing Mode. In

Profiling Mode, Hades-IoT simply observes ordinary, benign traffic, and builds a whitelist of what the packets of this type of traffic looks like[30]. After viewing enough traffic, Hades-IoT can enter Enforcing Mode. In Enforcing Mode, Hades-IoT outright drops any packet that doesn't fit the patterns on the whitelist. The authors state in their paper that requiring malicious traffic to train can be viewed as a drawback as it limits the kinds of malware a model can detect, and that by focusing only on benign traffic, that problem goes away[30]. Hades-IoT was tested on a variety of known malware, including IoTReaper, Persirai, and VPNFilter, and the study shows that Hades-IoT was able to detect all malicious activity and attacks during testing. In terms of performance metrics, the paper only mentions having a low False Positive Rate (see 4.4.), without specifying a number[30].

3.1.3 N-BaIoT

Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici published a paper titled "N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders" in 2018[31]. In the paper, the authers suggested using autoencoders to classify network traffic. The dataset in the study was network traffic from pcap-files, where network packets were measured on 115 statistics, and each packet was measured at five different times ranging from 100ms to 1 minute to model behaviour[31]. The solution developed was tested on the Mirai Botnet [9], and reported a False Positive Rate (FPR) of near zero as the only performance metric.[31].

3.1.4 Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine

In 2020, K. D. Abeyrathna, H. S. G. Pussewalage, S. N. Ranasinghe, V. A. Oleshchuk and O. C. Granmo published a study titled "Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine" in which a Tsetlin Machine was used to classify data from the KDD99 dataset (see chapter 2.3.2)[2]. In the paper, the authors explain that while other machine learning- and artificial intelligence- solutions are able to classify IoT traffic to a satisfying degree, their rules of detection and classification are obscured and uninterpretable to humans[2], often referred to as a *"black box"*. In terms of raw performance, the study finds that a Tsetlin Machine algorithm was able to classify the KDD99-dataset with an accuracy of 97.36%, with an F1-score of 0.9827[2]. In comparison, the other algorithms evaluated in the study, such as ANN-1, ANN-2, ANN-3, SVM, DT, RF and KNN, had accuracies ranging from 95.93% to 97.03%, and F1-scores between 0.9729 and 0.9812, which means the study found the Tsetlin Machine algorithm to be superior to a large number of known machine learning algorithms[2]. In terms of interpretability of the classification rules, the study found that certain parameters in the dataset such as "logged in" were used to create clear and powerful rules such as "IF logged in == TRUE then Attack"[2].

3.1.5 REDRESS

In 2023, S. Maheshwari et al. published a study titled "REDRESS: An Embedded Machine Learning Methodology Using Tsetlin Machines" in which the capabilities of the Tsetlin Machine in an embedded system were examined[6]. Maheshwari begins by explaining that neural network solutions in IoT security often have steep costs in both storage, runtime memory and computation[6]. The study goes on to explain how Tsetlin Machines could potentially be a superior alternative as they can run directly in hardware[6]. In the study, the authors compared the performance of the Tsetlin Machine with that of binary neural networks and found that not only was the Tsetlin Machine able to achieve competitive classification scores, but it was also faster by a factor of anywhere from 5 to 1900[6].

3.2 Existing Solutions

The existing solutions for anomaly detection in IoT consists largely of intrusion detection- and intrusion prevention systems, IDS and IPS for short. This section will look at the main features of both systems, as well as one system designed specifically for intrusion detection in embedded devices.

3.2.1 Intrusion Detection- (IDS) and Intrusion Prevention Systems (IPS)

The goal for an IDS, as the name suggests, is to detect intrusions or anomalies in network traffic [32]. An IDS can be both hardware and software systems, and looks for the anomalies and intrusions that a firewall is not able to detect. The main two categories for IDS is Anomaly-based (AIDS) and Signature-based (SIDS). An intrusion Prevention System (IPS) is similar to an IDS, as it is made to detect anomalies in network traffic, but in addition an IPS should take steps to prevent the anomalies from ever happening.

Signature intrusion systems use pattern matching to detect anomalies in the network traffic, and are dependant on a large set of known signatures [32]. The signatures in this case is known patterns of previous anomalies that are stored in a database for future lookup. When traffic passes through the IDS, the system is constantly checking for pattern matches in the data, and if a match is detected it triggers an alarm to notify operators of the threat. A major issue with SIDS is the dependency on previous signatures to identify an anomaly. The systems needs to be frequently updated, and are close to useless if there is a zero-day attack.

Anomaly-based intrusion systems is the result of trying to overcome the limitations of SIDS, and uses machine learning to classify the network traffic. A machine learning model is trained on normal network traffic, and set to raise an alarm if it detects anything abnormal [32]. AIDS is not dependant on a large knowledge base of signatures, meaning that it is more resilient against new anomalies and zero-day attacks. However, the classification of "normal" traffic can be challenging and the definition of normal user behavior can be constricting to the users. An example would be that a new routine is established, not yet recognized by the machine learning model. This would result in this new routine to be flagged as an anomaly, and raise the alarm.

3.2.2 Zeek

Zeek [33] is an open-source network monitoring tool used for security and performance analysis, with special emphasis on high speed high volume network monitoring. Zeek is classified as a signature based IDE, but scripting compatability broadens the possibilities in network monitoring, and functions as a hybrid IDE. The high speed and volume support allows clients to run Zeek on 10 gigabit ethernet and even 100 gigabyte ethernet networks.
A great advantage of Zeek over other security monitoring software, is that it generates high quality transaction logs. The logs contains protocols and activity as seen on the network, and display these in a neutral way. These logs become useful when analyzing threats and how they differ from normal traffic.

3.2.3 Suricata

As with Zeek, Suricata [34] is also a open source network analysis tool used for threat detection. The software is developed and released by the Open Information Security Foundation (OISF) [35], whose main goal is to provide community driven open source security software. Suricata is a combo of a signature- IDS and IPS, and is used to analyze and prevent security threats in networks. It offers IDS alerts, protocol transactions, network flow analysis, pcap recordings and extracting files. The IDS and IPS part, supports use of two spcialized rulesets for the signatures, the "Emerging Threats Suricata Ruleset" and the "VRT ruleset".

3.2.4 Snort

Snort is a open source signature based IPS widely used for network intrusion prevention and analysis [36]. Snort supports TCP dumping, packet logging and an IPS. As with Zeek and Suricata, Snort uses rulesets for signatures in order to detect threats on the network. Snort is highly configurable, and users can specify packages, warnings, modes and signatures to fit their system.

3.2.5 Embedded anomaly detection

Casillo et al. [37] examined a bayesian network capable of detecting potential cyber attacks on the CAN-BUS of vehicles. The solution was tested in a simulated environment where CAN messages is sent to the bayesian network running embedded, where the prediction results in a warning if the message is deemed as an intrusion. The study shows that the bayesian network model shows promise in detecting attacks in the CAN-BUS of vehicles.

Chapter 4

Method

This section details some of the methods used when working on this thesis. It includes the tools that were used along the way, how the datasets were pre-processed to fit the Tsetlin Machine, and how the training and testing pipeline worked.

4.1 Tools

4.1.1 Jupyterlab

Jupyterlab [38] functions as a in-browser IDE and runtime environment, and allows the user access to code and outputs regardless of which computer is used. By running jupyterlabs on a server, the user can start longer code runs such as AI training/testing without the need of keeping their main computer running. For this thesis, a high performance hardware server with jupyterlabs was used to perform early tests on the datasets and configuring runs for Weights and Biases (see 4.1.2.) The use of Jupyterlabs and the server hosting it, drastically shortened run times allowing for more runs per day.

4.1.2 Weights and Biases

Weights and Biases (W&B) [39] is a machine learning platform created to ease the development of AI models. It allows for easy logging and versioning of runs, and automatically creates useful graphs and tables from the logged parameters. The ease of logging makes setup of the testing and training pipeline efficient.

4.1.3 Lazy Predict

Lazy predict [40] is a python library developed for easy implementation and comparison of multiple machine learning algorithms. Lazy predict has models for both classification and regression, with 30 models each. The relevant models for this thesis are the classification models, where some examples are RandomForest, LinearSVC and DecisionTree. The easy implementation allows for quick testing and comparisons to compare the different models against ones own implementation. However, the output of the Lazy predict test only returns accuracy and F1 score, which does not provide the best insight into the model performance. A complete list of the models used in Lazy Predict can be found in Appendix A.

4.1.4 Scikit Learn Classifier Comparison

The Scikit Learn Classifier Comparison [41][42] is developed by Scikit learn to provide an illustration to the nature of decision boundaries between different classifier models. The benefits of Scikit Learn over lazy predict is the ability to add performance metrics to the runs, allowing for a better insight into the model performance.

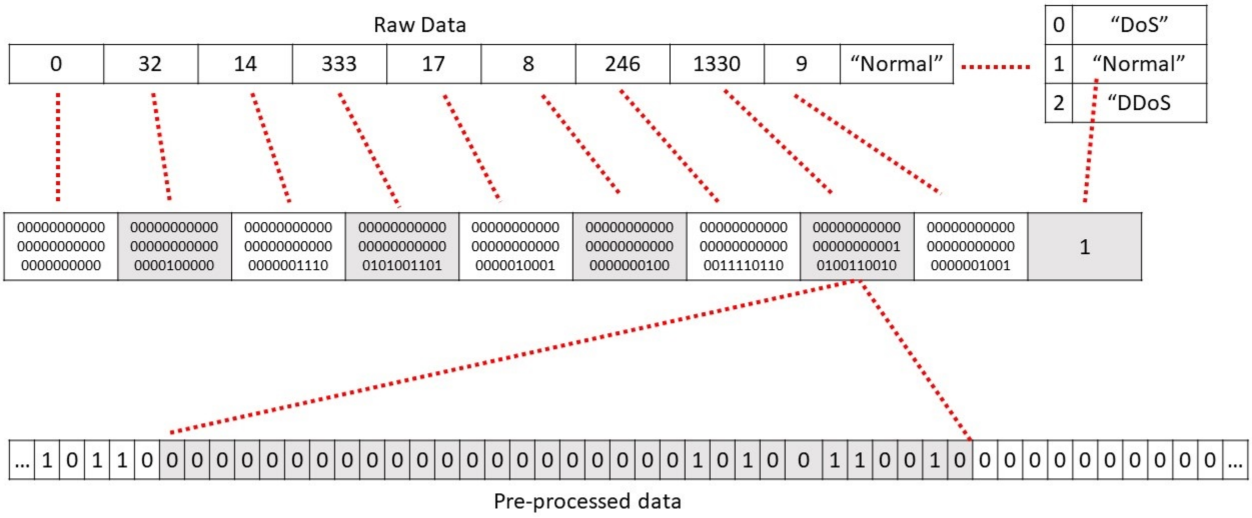

Figure 4.1: The raw data is converted to 32-bit binary numbers, which are split into individual bits. Those bits are then used as literals for the Tsetlin Machine.

4.1.5 Docker

Docker is a virtualization software used to run anything from simple python scripts to entire linux distributions in isolated virtual machines. A docker container can be used to deploy web applications, coordinate multiple programs in a shared enviroment, and give users a sandbox environment to work in without risk to the host machines[43]. For this thesis, Docker was used to host the python instances responsible for performing the hyperparameter sweeps with W&B.

4.2 Data Preprocessing

The datasets arrive in the form of comma-separated values, also called CSV-files. When read by the pandas library, they are converted into dataframes, a custom data structure similar to python dictionaries. Each row in the dataframe contains the raw data from one row of data, often in the form of numbers or strings. To prepare the data for the Tsetlin Machine, each element of each row is converted into a 32-bit binary number. For data types where directly casting to a number does not make as much sense, such as with strings, the different values for that dataset feature are put into a list. The index for each value in that list is then used as the numeric value instead, before the binary conversion. Once the values are represented by ones and zeros, they are added to a python list that represents that data entry. The process of converting a single data entry into a binary representation can be seen in figure 4.1

After this process, also called binarization, the data is normalized. Two python dictionaries are used for this purpose; One to keep all the data for each category, and one to keep track of the number of elements per category. To limit the number of the largest classes without limiting the smaller classes, an upper bound of number of entries per category is imposed on each category, resulting in a much more balanced distribution of data. Next, the balanced dataset is shuffled before being split into training and testing. Both the upper bound threshold and the training/test split values are hyperparameters that can be adjusted per run. The class size cutoff method is illustrated in figure 4.2

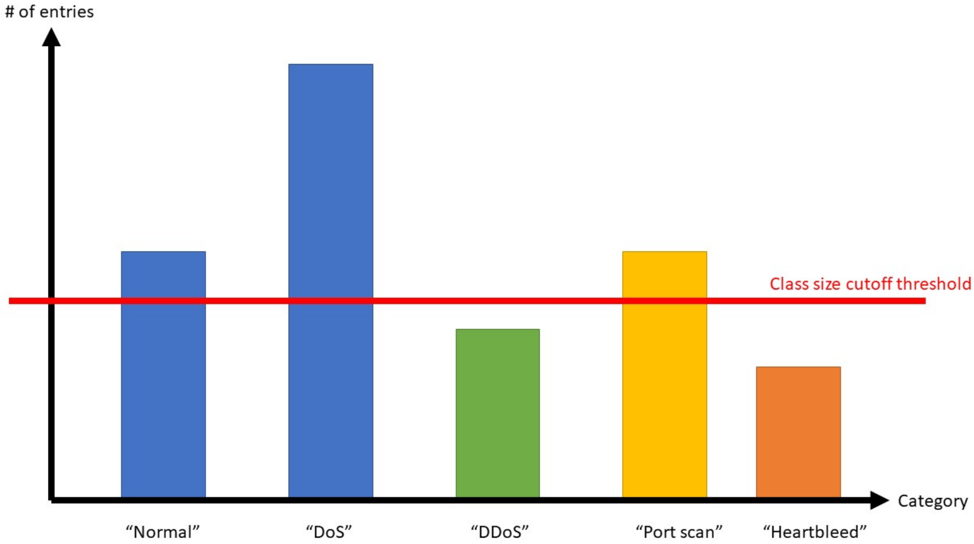

Figure 4.2: How the cutoff threshold is applied to balance classes

4.3 Training and Testing

When training and testing the Tsetlin Machine on the different datasets, the following steps were followed:

1. The dataset is split into two sets; training and testing with a 70/30 split.
2. The hyperparameters are set in W&B config dictionary to ensure logging.
3. During the run, the accuracy, loss, f1-score, recall, as well as testing and training time are logged.
4. After the run, the perfomance metrics recall, specificity, precision and F1 score are calculated.
5. Finally a confusion matrix is created to examine the performance over the different classes in the datasets.

After manual testing and training is done by following the steps above, a W&B sweep is started to find the best possible hyperparameters for each of the datasets.

4.3.1 Hyperparameter optimization with Sweeps

Weight and Biases(W&B) provides functionality to optimize hyperparameters for any model in what is called a Sweep[44]. A Sweep is similar to a grid-search in that a large number of runs with different hyperparameters are performed. A sweep is fundamentally different from a grid-search in that the user specifies only a range for each hyperparameter, before W&B handles the selection of each parameter used in a given run. W&B provides three selection methods; grid - iterate through all combinations, random, and bayes - random, only based on probabilities of improving performance[44]. W&B also provides a dashboard for each Sweep, showing any and all logged data for each run in the Sweep in the same graphs[44]. An important feature is that the Sweep is able to determine how important each hyperparameter is towards the final score for each run, as well as ranking the correlation for each parameter. This way, it is easy to figure out what combinations of hyperparameters will provide the best performance for any given model[44]. Figure 4.3 shows a visual representation of a sweep table displaying each hyperparameter for each run, as well as the accuracy of the individual runs.

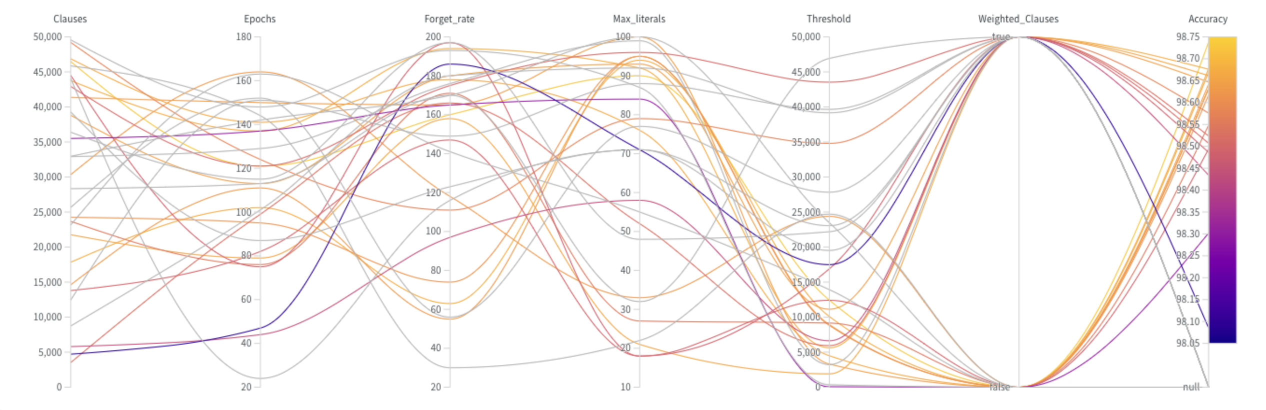

Figure 4.3: Visual representation of a sweep table

4.4 Performance Metrics

To understand how well the Tsetlin Machine performs on different datasets, it is important to establish ways to evaluate the models. While Accuracy - percentage of test data correctly classified, can give an indication of how well the Tsetlin Machine performs on a given dataset, there are other metrics that can give a more detailed and nuanced picture.

4.4.1 Confusion Matrix

One such metric is the confusion matrix [45]; a table where the rows are predicted classes and columns are true classes. In a confusion matrix, it is easy to see which classes the model is good at classifying, and where it struggles. The confusion matrix is based on comparing the predicted and true class of each data entry. The difference between what the model predicts and the truth can be described in four base values for each class. Consider here class A to represent the current class, and class B representing any class that isn't A:

- True Positive (TP)[45]: The model correctly predicts that a given data entry from class A belongs to class A.
- True Negative (TN)[45]: The model correctly does not predict that a given data entry from class B belongs to class A.
- False Positive (FP)[45]: The model wrongly predicts that a given data entry from class B belongs to class A.
- False Negative (FN)[45]: The model wrongly does not predict that a given data entry from class A belongs to class A.

4.4.2 Recall

Recall[45], also called Sensitivity, describes a model's ability to detect positive samples in data. Recall is defined as the total number of positives correctly identified as such, and since the total number of positive data consists of True Positives(TP) and False Negatives(FN), the formula to calculate Recall is as such:

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

4.4.3 Precision

The Precision[45] value represents a model's ability to corectly detect positive samples (True Positive, TP) in data without also falsely flagging negative data (False Positive, FP). Precision is defined as the percentage of positive predictions that are correct, and can be expressed as such:

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

4.4.4 Accuracy

As mentioned in the intro to this chapter, Accuracy[45] is perhaps the most basic performance metric. It is defined by the total number of correctly identified samples divided by the total number of samples. Since the correctly identified samples are the True Positives (TP) and True Negatives(TN), the formula for accuracy is as follows:

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

4.4.5 F1-Score

Another metric is the F1-score[45]. The F1 score is a combination of a few separate measures, and is regarded as the gold standard in performance metrics when dealing with machine learning algorithms. To calculate the F1-score, the Recall and Precision values are needed, with this formula:

$$F1score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

For multi-class datasets like the ones evaluated here, the F1-score is calculated per class.

4.5 Publishing a paper in ISTM

The results discovered when working on this thesis were submitted as a scientific paper for ISTM - the International Symposium on the Tsetlin Machine. As of the submission of this thesis the paper should still be pending publishing. Several of the tables containing results are also hosted on github at https://github.com/Nidaige/TM-IoT/tree/main/docs

4.6 Github

The code for the data loaders is hosted on github at https://github.com/Nidaige/TM-IoT. The code for instancing and running the Tsetlin Machine is found in TMU.py, and the data loaders for all five datasets can be found in datasets.py. Apart from support for non-binarized data, which we added later, the data loaders are also integrated into the TMU repository, and can be found at https://github.com/cair/tmu/tree/main/tmu/data.

Chapter 5

Results

This chapter aims to inform the reader of the performance of the Tsetlin Machine when used on the datasets covered in section 2.3. The data pre-prosessing and testing pipeline will also be covered. A thorough analysis of the results will be covered in chapter 6

5.1 Implementation

The final implementation of the data pre-processing and Tsetlin Machine consisted of the following steps:

1. Binarize data using the method described in chapter 4.2
2. Initialize the W&B configuration with the chosen hyperparameters for the current run
3. For each epoch:
   (a) Fit the Tsetlin Machine to the training data using the built-in tm.fit()-function
   (b) Have the Tsetlin Machine make a prediction on the test data using the built-in tm.predict()-function
   (c) Evaluate the performance over the test data using the sklearn metrics functions
   (d) Log the performance to W&B
4. At the end, log a confusion matrix to W&B

5.2 Model Perfomance

This section covers the achieved model performance for all datasets covered in section 2.3, and compare the Tsetlin Machine to a range of classifiers from the lazy predict and scikit classifier comparison python libraries. It is important to note that Lazy predict or Scikit learn Classifier comparison does not allow for changing hyperparameters, meaning that hyperparameters are not tuned for the data sets.

5.2.1 Sweeps

One sweep of roughly hyperparameter combinations was performed for each of the datasets, in an attempt to find the best possible hyperparameters for all datasets. The sweeps was configured to search within a certain range for each parameter, with the goal of optimizing the accuracy of the model. The optimization method used is Bayesian optimization, using probability to find the best combination of hyperparameters. Bayesian optimization is preferable when trying to find the best combination of many parameters, as it uses mathematics to select the next set of parameters in a way that maximizes the probability of a higher performance[44].

-	TM	ETC	RFC	SVC	LR	BC	CV
Accuracy	0.9873	0.9736	0.9730	0.9711	0.9669	0.9655	0.9652
F1 score	0.9860	0.9688	0.9683	0.9661	0.9654	0.9649	0.9639

Table 5.1: Perfomance of Lazy Predict and Tsetlin on CICIDS2017

Method	Accuracy	Loss	F1 Score	Recall	Precision
KNeighborsClassifier (B)	0.9509	0.0490	0.9492	0.9509	0.9492
LinearSVC (B)	0.9712	0.0288	0.9670	0.9711	0.9737
DecisionTreeClassifier (B)	0.5767	0.4231	0.5683	0.5768	0.8249
RandomForestClassifier (B)	0.6011	0.3989	0.5671	0.6010	0.6561
Neural Network (B)	0.9673	0.0327	0.9619	0.9672	0.9613
AdaBoostClassifier (B)	0.3030	0.6970	0.2817	0.3029	0.3215
GaussianNB (B)	0.7228	0.2771	0.7371	0.7228	0.8072
QuadraticDiscriminantAnalysis (B)	0.8254	0.1745	0.8239	0.8254	0.8702
KNeighborsClassifier	0.9447	0.0552	0.9430	0.9447	0.9443
LinearSVC	0.9680	0.0319	0.9638	0.9680	0.9697
DecisionTreeClassifier	0.5791	0.4208	0.5713	0.5791	0.8347
RandomForestClassifier	0.5077	0.4922	0.4727	0.5077	0.5956
Neural Network	0.9648	0.0351	0.9634	0.9648	0.9633
AdaBoostClassifier	0.1878	0.8121	0.0917	0.1878	0.8256
GaussianNB	0.7014	0.2985	0.7102	0.7014	0.7852
QuadraticDiscriminantAnalysis	0.8122	0.1877	0.8154	0.8122	0.8608
Tsetlin Machine	0.9873	0.0012	0.9860	0.9873	0.9847

Table 5.2: Performance of Scikit and Tsetlin on CICIDS2017

After running the sweeps for all datasets, the end results was a range of different combinations that provided the best accuracy for the individual datasets. This shows that there is no magical combination that works best for all datasets, but that each dataset require different combinations. In addition to the different combinations of hyperparameters, the sweeps provided useful insight in the importance and correlation of the hyperparameters.

5.2.2 CICIDS-2017

The Tsetlin Machine was able to achieve consistant results of above 98% accuracy, with F1-scores above 0.98. This in comparison with the models run through lazy predict, shows that the Tsetlin Machine achieves the highest accuracy and f1-score. Table 5.1 and 5.2 display the highest scores achieved by the Tsetlin Machine and some of the models in lazy predict and scikit learn. The full results from the sweeps on the CIC-IDS2017-dataset can be found in appendix B.1, note that a mistake was made when configuring the class size cutoff in the main sweep for CIC-IDS2017. The cutoff was first set to 5000, resulting in lower performance from the TM. After the mistake was found, a new sweep with fewer runs was done with the correct class size cutoff of 12000. The sweep with class size cutoff of 12000 can be seen in table B.1 and the sweep with 5000 in table B.2. The confusion matrix from the best performing run of CIC-IDS2017 can be found in appendix C, in table C.4.

5.2.3 KDD99

The best achieved accuracy by the Tsetlin Machine on the KDD99 dataset is 99.871% , with an F1 score of 0.9987. This again is higher than all models from Lazy Predict and Scikit Learn Classifier Comparison. Table 5.3 shows the results with descending accuracy. The

-	TM	ETC	RFC	LSVC	SVC	LR	PAC
Accuracy	99.29	99.613	99.613	99.291	98.969	98.905	98.776
F1 Score	0.9939	0.9961	0.9964	0.9945	0.9899	0.9906	0.9892

Table 5.3: Performance of Lazy Predict and Tsetlin on KDD-99

Method	Accuracy	Loss	F1 Score	Recall	Precision
KNeighborsClassifier (B)	0.9684	0.0315	0.9714	0.9684	0.9755
LinearSVC (B)	0.9916	0.0083	0.9932	0.9916	0.9948
DecisionTreeClassifier (B)	0.6536	0.3464	0.5901	0.6535	0.8988
RandomForestClassifier (B)	0.8950	0.1049	0.8962	0.8950	0.9071
Neural Network (B)	0.9923	0.0077	0.9938	0.9922	0.9955
AdaBoostClassifier (B)	0.2209	0.7791	0.1352	0.2208	0.9030
GaussianNB (B)	0.9504	0.0495	0.9575	0.9504	0.9705
QuadraticDiscriminantAnalysis (B)	0.8995	0.1004	0.9449	0.8995	0.9991
KNeighborsClassifier	0.9832	0.0167	0.9861	0.9832	0.9890
LinearSVC	0.9774	0.0225	0.9797	0.9774	0.9821
DecisionTreeClassifier	0.6677	0.3322	0.6109	0.6677	0.8348
RandomForestClassifier	0.9484	0.0515	0.9514	0.9484	0.9585
Neural Network	0.9800	0.0199	0.9822	0.9800	0.9844
AdaBoostClassifier	0.2215	0.7784	0.1352	0.2215	0.9030
GaussianNB	0.9182	0.0817	0.9317	0.9182	0.9548
QuadraticDiscriminantAnalysis	0.9755	0.0244	0.9758	0.9755	0.9772
Tsetlin Machine	0.9929	0.0071	0.9939	0.9929	0.9949

Table 5.4: Performance of Scikit and Tsetlin on KDD99

results from Scikit classifier comparison can be seen in table 5.4. The complete sweep table from KDD99 can be seen in appendix B.2. The confusion matrix from the best performing run of KDD99 can be found in appendix C, in table C.3.

5.2.4 NSL-KDD

The best achieved accuracy for the NSL-KDD dataset was 85.559% with an F1 score of 0.804, which is comparable to the models in Lazy Predict and higher than the models in Scikit Learn Classifier Comparison. Table 5.5 shows the results with descending accuracy. We also performed tests using scikit learn Classifier comparison. The results can be seen in table 5.6. The complete sweep table for NSL-KDD can be seen in appendix B.3. The confusion matrix from the best performing run of NSL-KDD can be found in appendix C, in table C.1.

5.2.5 UNSW NB15

The highest accuracy achieved by the TsetTsetlinMalin Machinechine on the UNSW NB15 dataset was 79.77%, performing better than all models in Scikit Classifier comparison, but falling short of the models in Lazy Predict. The top results of Lazy Predict and the Tsetlin

-	TM	RC	RCV	LDA	XGBC	PC	PAC
Accuracy	0.8532	0.8734	0.8732	0.8691	0.8487	0.8470	0.8445
F1 Score	0.8006	0.8430	0.8426	0.8416	0.7969	0.7954	0.7964

Table 5.5: Performance of Lazy Predict and Tsetlin on NSL-KDD

Method	Accuracy	Loss	F1 Score	Recall	Precision
KNeighborsClassifier (B)	0.8297	0.1703	0.7820	0.8296	0.7791
LinearSVC (B)	0.8391	0.1609	0.7898	0.8390	0.8556
DecisionTreeClassifier (B)	0.7869	0.2130	0.7252	0.7869	0.8391
RandomForestClassifier (B)	0.7590	0.2409	0.6720	0.7590	0.8099
Neural Network (B)	0.8448	0.1552	0.7967	0.8447	0.8745
AdaBoostClassifier (B)	0.7482	0.2517	0.6539	0.7482	0.8184
GaussianNB (B)	0.7858	0.2142	0.7522	0.7857	0.7656
QuadraticDiscriminantAnalysis (B)	0.7933	0.2066	0.7492	0.7933	0.8068
KNeighborsClassifier	0.8177	0.1822	0.7733	0.8177	0.8554
LinearSVC	0.8322	0.1677	0.7884	0.8322	0.8384
DecisionTreeClassifier	0.6480	0.3519	0.6419	0.6480	0.8771
RandomForestClassifier	0.8349	0.1650	0.7797	0.8349	0.8662
Neural Network	0.8260	0.1739	0.7818	0.8260	0.8008
AdaBoostClassifier	0.5880	0.4199	0.4817	0.5880	0.7567
GaussianNB	0.4629	0.5370	0.4854	0.4629	0.6991
QuadraticDiscriminantAnalysis	0.5834	0.4165	0.5960	0.5834	0.7613
Tsetlin Machine	0.8532	0.1468	0.8006	0.8532	0.8794

Table 5.6: Performance of Scikit and Tsetlin on NSL-KDD

-	TM	XGBC	SVC	CCCV	BC	LR	PC
Accuracy	0.7977	0.8290	0.8208	0.8125	0.8124	0.8110	0.8098
F1 Score	0.7950	0.8248	0.8124	0.8025	0.8082	0.8058	0.8046

Table 5.7: Performance of Lazy Predict and Tsetlin on UNSW NB15

Machine can be seen in table 5.7, while all results from scikit can be seen in table 5.8. The full sweep table from UNSW NB15 can be seen in appendix B.4. The confusion matrix from the best performing run of UNSW NB15 can be found in appendix C, in table C.5.

5.2.6 UNSW Bot-IoT

The highest achieved accuracy and f1 scores for the Bot-IoT dataset were 100% and 1.00 respectively. These values are discussed deeper in chapter 6.2.2. In comparison, the performance achieved by the various models with Lazy Predict and Scikit learn classifier comparison can be seen in table 5.9 and table 5.10. The full results from the sweep on UNSW Bot-IoT can be seen in appendix B.5. The confusion matrix from the best performing run of UNSW Bot-IoT can be found in appendix C, in table C.2.

5.3 Hyperparameter Importance

The sweep functionality of W&B provides an overview of the hyperparameters and their importance in achieving the highest accuracy. In appendix B.6 table B.7 shows all hyperparameters and their importance and correlation for each dataset. It is important to note that correlation between above +- 0.6 are considered high correlation.

Method	Accuracy	Loss	F1 Score	Recall	Precision
KNeighborsClassifier (B)	0.7621	0.2378	0.7585	0.7621	0.7637
LinearSVC (B)	0.8124	0.1876	0.8037	0.8123	0.8306
DecisionTreeClassifier (B)	0.7177	0.2822	0.6952	0.7177	0.7878
RandomForestClassifier (B)	0.5683	0.4316	0.5164	0.5683	0.6435
Neural Network (B)	0.8211	0.1789	0.8164	0.8210	0.8443
AdaBoostClassifier (B)	0.4784	0.5215	0.4119	0.4784	0.7797
GaussianNB (B)	0.3790	0.6210	0.4247	0.3789	0.8006
QuadraticDiscriminantAnalysis (B)	0.6687	0.3313	0.6613	0.6686	0.7469
KNeighborsClassifier	0.7685	0.2314	0.7709	0.7685	0.7829
LinearSVC	0.8087	0.1912	0.8022	0.8087	0.8230
DecisionTreeClassifier	0.7034	0.2965	0.6748	0.7034	0.7635
RandomForestClassifier	0.5010	0.4989	0.4441	0.5010	0.5904
Neural Network	0.8196	0.1803	0.8175	0.8196	0.8394
AdaBoostClassifier	0.5032	0.4967	0.4816	0.5032	0.6989
GaussianNB	0.4112	0.5887	0.4618	0.4112	0.7856
QuadraticDiscriminantAnalysis	0.5724	0.4275	0.5583	0.5724	0.7041
Tsetlin Machine	0.7977	0.2022	0.7950	0.7977	0.8412

Table 5.8: Performance of Scikit and Tsetlin on UNSW NB15

-	TM	ABC	BC	CCCV	LSVC	LR	PAC
Accuracy	1.0	1.0	1.0	0.9998	0.9998	0.9997	0.9997
F1 Score	1.0	1.0	1.0	0.9998	0.9998	0.9997	0.9997

Table 5.9: Performance of Lazy Predict and Tsetlin on UNSW Bot-Iot

Method	Accuracy	Loss	F1 Score	Recall	Precision
KNeighborsClassifier (B)	0.9993	0.0007	0.9992	0.9992	0.9992
LinearSVC (B)	1.0	0.0	1.0	1.0	1.0
DecisionTreeClassifier (B)	1.0	0.0	1.0	1.0	1.0
RandomForestClassifier (B)	0.9417	0.0582	0.9397	0.9417	0.9423
Neural Network (B)	0.9998	0.0001	0.9998	0.9998	0.9998
AdaBoostClassifier (B)	0.9979	0.0020	0.9968	0.9979	0.9979
GaussianNB (B)	0.9997	0.0002	0.9997	0.9997	0.9997
QuadraticDiscriminantAnalysis (B)	0.9919	0.0081	0.9927	0.9918	0.9946
KNeighborsClassifier	0.9997	0.0002	0.9997	0.9997	0.9997
LinearSVC	1.0	0.0	1.0	1.0	1.0
DecisionTreeClassifier	1.0	0.0	1.0	1.0	1.0
RandomForestClassifier	0.9404	0.0595	0.9331	0.9404	0.9427
Neural Network	1.0	0.0	1.0	1.0	1.0
AdaBoostClassifier	1.0	0.0	1.0	1.0	1.0
GaussianNB	0.9999	0.0001	0.9999	0.9999	0.9999
QuadraticDiscriminantAnalysis	0.9921	0.0078	0.9928	0.9921	0.9944
Tsetlin Machine	1.0	0.0182	1.0	1.0	1.0

Table 5.10: Performance of Scikit and Tsetlin on UNSW Bot-IoT

Chapter 6

Discussions

This chapter aims to examine and highlight the most important findings from the paper, and provide a thorough analysis from the results covered in chapter 5. The chapter will go over the results in general, as well as how they compare to those achieved with Lazy Predict and Scikit Learn Classifier Comparison. Some dataset-specific issues, the quality of the implementation and proposed further research are also covered.

6.1 About the results

As a general observation, the Tsetlin Machine performs as well, if not better, than the models featured in Lazy Predict and Scikit Learn Classifier Comparison. This confirms both hypotheses put forward in the thesis definition (see 1.3.) The models were able to reach a high performance level with f1-scores on the high end of 0.9, with the exception of UNSW-NB15 and NSL-KDD, where all models were struggling to improve beyond a f1-score of 0.85.

6.1.1 Lazy Predict & Scikit Learn

Comparing the results achieved by the Tsetlin Machine, to those of the models from Lazy predict and Scikit learn provided useful insight as to where the Tsetlin Machine performes in comparison to other models. However, the Tsetlin Machine was run with optimized hyperparameters and with the data pre-processing tailored to fit. The models from Lazypredict and Scikit learn does not allow for changing hyperparameters, possibly making the results skewed with an advantage to the Tsetlin Machine.

6.1.2 Feature extraction and interpretability

Due to how the data is processed, each feature in the original dataset is represented by multiple literals after preprocessing. This introduces an interpretability issue where the clauses discriminate on literals that represent various powers of 2 rather than complete features in the data.
Take for example the packet header length. This feature is typically an integer representing the number of bits or bytes in the http header for a given packet. Assume in this case that one of the literals represent whether or not the packet length is longer than 64.
An interpretable clause could say "IF packetHeaderLength>=64 THEN DDoS". This is simple to understand, as the clause simply states that any packet with a header longer than 64 bits or bytes is likely to be a DDoS attempt.
The actual clause would, if packet header length was the first feature in the data, read as "IF x24 THEN DDoS" which not only makes no sense to a human observer, but also requires post-processing to map the literals back to the original values before any meaningful insight can be gained.

6.1.3 Binarized vs Non-binarized data

From the results, several of the models tested achieved greater performance scores on the data preprocessed and binarized for the Tsetlin Machine, compared to reading directly from the .csv-files. This might be because the binarization process breaks each value into 32 bits, which increases the number of inputs, which could enable a finer, more nuanced classification. While the classification performance is increased, the processing also massively increases the memory requirement both for the data structure holding the data set, as well as for the models themselves.

6.2 Dataset-specific Issues

A recurring issue regarding the data sets is the unbalanced nature of network traffic. The number of entries in each class vary greatly and created problems early on in the testing of the Tsetlin Machine. The most notable example was the UNSW-NB15 dataset, which has a wast majority of benign entries, with the anomalies being smaller classes, resulting in high accuracy without predicting anomalies correctly. Creating a class size cut of for each dataset and limiting the largest class improved the Tsetlin Machines ability to predict correct labels, and improved the performance metrics noticeably.

6.2.1 USNW-NB15

Inconsistent labelling

In the USNW-NB15 dataset, there are some labels that essentially have the same name. "Fuzzers" and " Fuzzers " (note the added spaces) are two of the multiple classes that share names, but looking at the confusion matrix (see table C.5) from the final run of the dataset reveals that while the names are similar, the classes are distinct enough for the TM to separate between them like it does any other class in the dataset. A test to verify this conclusion was done, with a custom script combining the categories with the same names, and then calculating performance metrics, and there was an improvement of less than 1%, which lends credibility to the idea that these are different categories with very similar names.

6.2.2 Bot-IoT

Differences between the sample dataset and the full dataset

Bot-IoT is a very large dataset, and the host website provides two versions; a 5% subset split into four files and the full set split into 74 files. However, there is a more fundamental difference between the two versions; the features are different, which means a Tsetlin Machine trained on data from one should not be used to classify data from the other. For instance, a run on the 5% subset that scored 98% accuracy on test data from the same set scored roughly 1% accuracy on data randomly selected from the full set version.

Suspected overfitting

While the results from testing on the Bot-IoT dataset reaching 100% accuracy and an F1-score of 1.0 could lead one to believe the model was heavily overfitted to the training data, another possible explanation is that the dataset itself is simply very easy to classify. Some subsequent tests to verify this theory were done, with a 30/70 split; 30% training data and 70% test data, both with a cutoff threshold of 100, where all categories are practically equal in size, and with 12000, where the larger categories dominate, and both resulted in accuracies above 99.5%. Looking at a confusion matrix of the results, it was clear that there were many samples from all categories present in the test data, which meant it was not simply a case of smaller categories all ending up in the training data by pure chance.

Dataset	Epochs	Clauses	Forget Rate	Max Literals	Threshold	Weighted Clauses
CICIDS2017	121	46476	160	90	12416	False
KDD99	65	49883	20	25	12289	True
NSL-KDD	20	2183	115	10	49662	False
UNSW-NB15	53	15302	133	10	3016	False
UNSW-Bot-IoT	91	36982	184	17	1037	True

Table 6.1: Hyper parameters for all data sets

6.3 About the implementation

Overall, the implementation of the data pre-processing and the actual Tsetlin Machine has been successful and served its purpose. However there are a few tweaks we would have liked to make earlier. The most notable of these is the requirement to pre-process the data for each run of the algorithm.
As of now, every time a new run is initiated, the first step is to process the data as described in section 4.2. This is a time consuming process that could be reduced by simply processing the data once, and then saving the data with its new format to a new file that can be used in the subsequent runs. This method would also introduce some downsides, and the integrity of the data would need to be checked thoroughly before the new data could be accepted. The biggest hazard by saving the data to a new file, would be that entire classes could be present in the training data, but not once in the training set, or the other way around.

6.4 Hyperparameters

Table 6.1 displays the best hyperparameters used by the Tsetlin Machine for the different datasets, and were found by using W&B sweeps. All results made by the Tsetlin Machine were achieved by using the hyperparameters in the table.
In addition to the found hyperparameters, W&B provide a table displaying the importance and correlation of each hyperparameter. The full table can be seen in table B.7. Our initial beliefs were that the hyperparameters, importance and the correlation would be similar across the different datasets, and only differ slightly. This proved to be wrong, and the hyperparameters has a wide range of values across the datasets. Even more surprising is the importance and correlation of the hyperparameters. For CICIDS-2017, the forget rate has a high importance, and a high negative correlation, but in the other datasets, the forget rate is of low importance and has lower correlation. This variance of importance and correlation can be seen in all the datasets, and none seem to be common with each other. The reason for this variance could be the different formats and states of the datasets used in this thesis, and a future test on real network traffic could provide a clearer picture as to what importance and correlation the hyperparameters have in a non simulated setting.

6.5 Future research

From the results achieved and experiences earned while working on the thesis, there are a few leads that could be pursued further in future work. Firstly, testing the Tsetlin Machine in a realistic, real-time simulated network with realistic conditions to understand how it performs on real world data in real time. This could then be further explored by running the Tsetlin Machine in a real IoT-device in real conditions. Secondly, to develop a hardware-compatible implementation, and explore the performance of the Tsetlin Machine on a chip. Thirdly, to automate the selection of the threshold value and hyperparameter tuning. The implementation used in this thesis had a set threshold of 12000 samples per class and used pre-determined hyperparameters, which means the actual best combination for each dataset could remain undiscovered. Fourthly, to investigate the interpretability of the Tsetlin Machine in a IoT security context, as a better understanding of the clauses created could indicate potential improvements to the data processing and model. Finally, type 3 feedback could be explored to further reduce the computation time and resource requirements of the Tsetlin Machine. Type 3 feedback aims to include features that are relevant, and exclude irrelevant features in the data reducing the total number of features to be considered when predicting a class.

Chapter 7

Conclusions

Smart IoT-devices are often built with limited performance components and have little in the ways of built-in cybersecurity. An AI model could potentially classify incoming traffic and distinguish benign from malign. However, most of today's models are resource-intensive, especially in ways of memory and computation, and require powerful hardware to train. The Tsetlin Machine is a potential candidate here, as the binary, propositional logic model is compatible with hardware, and can perform inference in near real-time. In this thesis, the Tsetlin Machine was assessed in its ability to detect anomalies in network traffic to determine its viability as a intrusion detection system in IoT devices.

To provide the best possible evaluation we developed custom data loaders for five benchmark datasets; CICIDS 2017, KDD99, NSL-KDD, UNSW-NB15 and UNSW-Bot-IoT. We then preprocessed the data, and ran experiments and hyperparameter searches for each individual dataset. After optimizing the hyperparameters, we compared the achieved performance of the Tsetlin Machine to various classifiers from scikit learn classifier comparison and lazy predict.

Initially we had two hypotheses related to the performance of the Tsetlin Machine. Our first hypothesis, *"The Tsetlin Machine can provide a high accuracy on real IoT network traffic."*, was thoroughly confirmed for three of the datasets, reaching accuracies of 0.9929, 0.9873, and 1.0 for KDD99, CIC-IDS2017 and UNSW-Bot-IoT respectively. The second hypothesis, *"The Tsetlin Machine can perform as well or better than other machine learning methods when classifying IoT traffic."* was also confirmed by the experimental results, where the Tsetlin Machine performed the highest out of all the tested models for four out of five datasets, with UNSW-NB15 being the sole exception where the TM still ranked 5th. To give a clear answer to our research question:

Research Question: How can Tsetlin Machines be effectively trained to detect anomalies in IoT device data streams, and if so, how does the performance compare to that of other models?

Answer: The Tsetlin Machine can be effectively trained on IoT network data by converting each data value to a 32-bit binary number and imposing an upper bound on class sizes. The performance achieved after hyperparameter optimization was competitive or superior to the other classification models tested.

Given the experimental results and their conclusions, the Tsetlin Machine presents itself as a powerful tool for IoT anomaly detection. However, further work is required to deploy it as a real-world solution.

Bibliography

[1] Liang Xiao et al. "IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?" In: *IEEE Signal Processing Magazine* 35.5 (2018), pp. 41–49.

[2] Kuruge Abeyrathna et al. "Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine." In: Oct. 2020. DOI: 10.1109/SSCI47803.2020.9308206.

[3] Ole-Christoffer Granmo. *An Introduction to Tsetlin Machines: Your First Tsetlin Machine.* 2021. URL: http://tsetlinmachine.org/wp-content/uploads/2021/09/Tsetlin_Machine_Book_Chapter_1-4.pdf. (accessed: 04.10.2022).

[4] Ole-Christoffer Granmo. "The Tsetlin Machine–A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic." In: *arXiv preprint arXiv:1804.01508* (2018).

[5] Ole-Christoffer Granmo. *Tsetlin Machine.* 2023. URL: https://github.com/cair/TsetlinMachine/blob/master/README.md. (Accessed 25.05.2023.

[6] S. Maheshwari et al. "REDRESS: Generating Compressed Models for Edge Inference Using Tsetlin Machines." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023), pp. 1–16. DOI: 10.1109/TPAMI.2023.3268415.

[7] Karen Rose, Scott Eldridge, and Lyman Chapin. "The internet of things: An overview." In: *The internet society (ISOC)* 80 (2015), pp. 1–50.

[8] The Norwegian Consumer Counsil. *Significant security flaws in smartwatches for children.* 2017. URL: https://www.forbrukerradet.no/side/significant-security-flaws-in-smartwatches-for-children/. (accessed: 14.12.2022).

[9] Malwarebytes. *What was the Mirai botnet?* 2023. URL: https://www.malwarebytes.com/what-was-the-mirai-botnet. (Accessed 24.05.2023.

[10] Canada Canadian Institute for Cybersecurity - University of New Brunswick. *Intrusion Detection Evaluation Dataset (CIC-IDS2017).* 2017. URL: https://www.unb.ca/cic/datasets/ids-2017.html. (Accessed 01.03.2023).

[11] Lincoln Laboratory. *1998 DARPA INTRUSION DETECTION EVALUATION DATASET.* 2023. URL: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset. (Accessed 24.05.2023.

[12] Information and Irvine Computer Science University of California. *KDD Cup 1999 Data.* 1999. URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. (Accessed 01.03.2023).

[13] Information and Irvine Computer Science University of California. *INTRUSION DETECTOR LEARNING.* 1999. URL: http://kdd.ics.uci.edu/databases/kddcup99/task.html. (Accessed 01.03.2023).

[14] Canada Canadian Institute for Cybersecurity - University of New Brunswick. *NSL-KDD dataset.* URL: https://www.unb.ca/cic/datasets/nsl.html. (Accessed 03.03.2023).

[15] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In: *Military Communications and Information Systems Conference (MilCIS)* 6.5 (2015), pp. 8182–8201.

[16] Nour Moustafa and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set." In: *Information Security Journal: A Global Perspective* 25.1-3 (2016), pp. 18–31. DOI: 10.1080/19393555.2015.1125974. eprint: https://doi.org/10.1080/19393555.2015.1125974. URL: https://doi.org/10.1080/19393555.2015.1125974.

[17] Nour Moustafa, Jill Slay, and Gideon Creech. "Novel Geometric Area Analysis Technique for Anomaly Detection Using Trapezoidal Area Estimation on Large-Scale Networks." In: *IEEE Transactions on Big Data* 5.4 (2019), pp. 481–494. DOI: 10.1109/TBDATA.2017.2715166.

[18] Nour Moustafa, Jill Slay, and Gideon Creech. "Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models." In: Cham: Springer International Publishing, 2017, pp. 127–156. ISBN: 978-3-319-59439-2. DOI: 10.1007/978-3-319-59439-2_5. URL: https://doi.org/10.1007/978-3-319-59439-2_5.

[19] Mohanad Sarhan et al. "NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems." In: *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer International Publishing, 2021, pp. 117–135. DOI: 10.1007/978-3-030-72802-1_9. URL: https://doi.org/10.1007%5C%2F978-3-030-72802-1_9.

[20] Australia Dr Nour Moustafa - Intelligent Security Group University of New South Wales Canberra. *The UNSW-NB15 Dataset*. 2021. URL: https://research.unsw.edu.au/projects/unsw-nb15-dataset. (Accessed 03.03.2023).

[21] Koroniotis et al. "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset." In: *Future Generation Computer Systems* 100 (2019), pp. 779–796.

[22] Koroniotis et al. "Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques." In: *International Conference on Mobile Networks and Management* 235 (2017), pp. 30–44.

[23] Koroniotis et al. "A new network forensic framework based on deep learning for Internet of Things networks: A particle deep framework." In: *Future Generation Computer Systems* 110 (2020), pp. 91–106.

[24] Koroniotis et al. "Enhancing network forensics with particle swarm and deep learning: The particle deep framework." In: *arXiv preprint arXiv* 2005.00722 (2020).

[25] Koroniotis et al. "A Holistic Review of Cybersecurity and Reliability Perspectives in Smart Airports." In: *IEEE Access* 8 (2020), pp. 209802–209834.

[26] Nickolaos Koroniotis. "Designing an effective network forensic framework for the investigation of botnets in the Internet of Things." In: *PhD diss.* (2020).

[27] Australia Dr Nour Moustafa - Intelligent Security Group University of New South Wales Canberra. *The UNSW-Bot-IoT Dataset*. 2021. URL: https://research.unsw.edu.au/projects/bot-iot-dataset. (Accessed 07.03.2023).

[28] Francesca Meneghello et al. "IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices." In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8182–8201.

[29] Y. Mirsky et al. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection." In: *Network and Distributed Systems Security Symposium* (2018).

[30] D. Breitenbacher et al. "HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices (Extended Version)." In: *IEEE Internet of Things Journal* 9 (12 2022).

[31] Y. Meidan et al. "N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders." In: *IEEE PERVASIVE COMPUTING* 13 (9 2018).

[32] Ansam Khraisat et al. "Survey of intrusion detection systems: techniques, datasets and challenges." In: *Cybersecurity* 2.1 (2019), pp. 1–22.

[33] The Zeek Project. *About Zeek*. 2023. URL: https://docs.zeek.org/en/master/about.html. (Accessed 24.05.2023.

[34] OISF. *Suricata*. 2023. URL: https://suricata.io/. (Accessed 24.05.2023.

[35] OISF. *OISF Community Driven Open Source*. 2023. URL: https://oisf.net/. (Accessed 24.05.2023.

[36] Cisco. *What is Snort?* 2023. URL: https://www.snort.org/#documents. (Accessed 09.05.2023).

[37] Mario Casillo et al. "Embedded Intrusion Detection System for Detecting Attacks over CAN-BUS." In: *2019 4th International Conference on System Reliability and Safety (ICSRS)*. 2019, pp. 136–141. DOI: 10.1109/ICSRS48664.2019.8987605.

[38] Jupyter Team. *Jupyter Project Documentation*. 2023. URL: https://docs.jupyter.org/en/latest/. (Accessed 24.05.2023.

[39] Weights and Biases. *Weights and Biases Documentation*. 2023. URL: https://docs.wandb.ai. (Accessed 15.05.2023).

[40] Shankar Rao Pandala. *Lazy Predict Documentation*. 2023. URL: https://pypi.org/project/lazypredict/. (Accessed 20.03.2023).

[41] F. Pedregosa et al. *Classifier Comparison*. 2023. URL: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py. (Accessed 15.05.2023).

[42] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[43] Docker. *Docker*. 2023. URL: https://www.docker.com/. (Accessed 22.05.2023.

[44] Weights and Biases. *Tune Hyperparameters*. 2023. URL: https://docs.wandb.ai/guides/sweeps. (Accessed 08.03.2023).

[45] Tutorialspoint. *Machine Learning - Performance Metrics*. URL: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_algorithms_performance_metrics.htm. (Accessed 03.06.2023).

Appendix A

Lazy Predict Models

The following table A.1 contains all models used in lazy predict classification, and their abbreviation.

Full Name	Abbreviation
Linear SVC	LSVC
Stochastic Gradient Descent Classifier	SGDC
Perceptron	ptron
Logistic Regression	LR
Support Vector Classification	SVC
Calibrated Classifier CV	CCCV
Passive Aggressive Classifier	PAC
Label Propagation	LP
Label Spreading	LS
Random Forrest Classifier	RFC
Gradient Boosting Classifier	GBC
Quadratic Discriminant Analysis	QDA
Hist Gradient Boosting Classifier	HGBC
Ridge Classifier CV	RCCV
Ridge Classifier	RC
Ada Boost Classifier	ABC
Extra Trees Classifier	ETC
KNeighbours Classifier	KNC
Bagging Classifier	BC
Bernouli Naive Bayes	BNB
Linear Discriminant Analysis	LDA
Gaussian Naive Bayes	GNB
Nu-Support Vector Classification	NUSVC
Decision Tree Classifier	DTC
Nearest Centroid	NC
Checking Classifier	CC
Dummy Classifier	DC

Table A.1: Model name abreviations

Appendix B

Dataset sweep tables

B.1 CICIDS 2017

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy	F1 score:	Loss	Recall
#1	46476	121	160	90	12416	FALSE	98.73864	0.986077	1.261355	0.987386
#2	17840	102	63	94	10086	FALSE	98.67808	0.98567	1.321917	0.986781
#3	43760	137	194	92	5634	TRUE	98.67599	0.985188	1.324005	0.98676
#4	46873	141	178	76	4485	FALSE	98.66973	0.9851	1.33027	0.986697
#5	21758	79	158	21	1907	TRUE	98.66138	0.985309	1.338624	0.986614
#6	38809	113	180	93	11129	TRUE	98.64676	0.984852	1.353242	0.986468
#7	41367	150	165	100	3298	FALSE	98.64258	0.984835	1.357419	0.986426
#8	14617	111	55	95	8925	FALSE	98.63214	0.984978	1.36786	0.986321
#9	24256	95	74	95	8816	FALSE	98.61752	0.984703	1.382479	0.986175
#10	49205	124	111	79	34799	TRUE	98.57575	0.983991	1.424246	0.985758
#11	23670	76	171	27	9151	FALSE	98.54861	0.983938	1.451394	0.985486
#12	3520	99	166	52	5942	TRUE	98.52355	0.983562	1.476454	0.985235
#13	42923	121	175	96	43537	TRUE	98.50684	0.983145	1.493161	0.985068
#14	13787	82	147	18	12397	FALSE	98.49431	0.983819	1.505691	0.984943
#15	44499	75	197	18	16898	TRUE	98.48596	0.982579	1.514044	0.98486
#16	5793	44	97	58	6626	TRUE	98.43375	0.98235	1.566252	0.984337
#17	35498	137	165	84	67	FALSE	98.30218	0.982094	1.697818	0.983022
#18	4714	47	186	71	17490	TRUE	98.085	0.978774	1.915005	0.98085

Table B.1: Results of sweep on CIC-IDS2017 with a class size cutoff threshold of 12000

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy	F1 score:	Loss	Recall
#1	28008	91	44	93	7652	FALSE	97.65926	0.972455	2.340743	0.976593
#2	27977	100	30	149	9585	TRUE	97.62821	0.973909	2.371787	0.976282
#3	29634	71	20	67	3839	FALSE	97.53508	0.973741	2.46492	0.975351
#4	22164	76	51	57	2586	TRUE	97.49783	0.972012	2.502173	0.974978
#5	25455	73	29	76	7405	TRUE	97.48541	0.971149	2.514591	0.974854
#6	18878	66	143	75	3703	FALSE	97.44195	0.969476	2.558053	0.974419
#7	26441	89	17	170	2023	FALSE	97.42332	0.972597	2.576679	0.974233
#8	29922	98	218	62	6011	FALSE	97.37365	0.969217	2.62635	0.973736
#9	15555	98	59	23	6287	FALSE	97.3364	0.969617	2.663604	0.973364
#10	19379	93	129	102	7196	FALSE	97.33019	0.968828	2.669812	0.973302
#11	20062	75	69	83	3408	FALSE	97.29914	0.969248	2.700857	0.972991
#12	29362	46	83	174	4499	FALSE	97.28673	0.968398	2.713275	0.972867
#13	17206	78	125	33	1903	TRUE	97.26189	0.970062	2.73811	0.972619
#14	27772	93	395	14	3193	TRUE	97.26189	0.96985	2.73811	0.972619
#15	28393	100	193	180	3088	TRUE	97.25568	0.967551	2.744319	0.972557
#16	22656	88	187	65	828	FALSE	97.24947	0.968842	2.750528	0.972495
#17	29556	77	289	63	1997	TRUE	97.18117	0.968175	2.818825	0.971812
#18	27057	62	293	20	5237	FALSE	97.15634	0.966705	2.843661	0.971563
#19	25076	94	269	121	3367	FALSE	97.13771	0.966742	2.862287	0.971377
#20	29367	92	352	39	1265	TRUE	97.11909	0.967446	2.880914	0.971191
#21	22605	69	396	43	2905	FALSE	97.10667	0.965995	2.893332	0.971067
#22	24490	70	379	30	1312	FALSE	97.08804	0.966091	2.911958	0.97088
#23	27855	50	39	37	165	TRUE	97.08804	0.967057	2.911958	0.97088
#24	26771	61	242	73	1350	TRUE	97.08183	0.966038	2.918167	0.970818
#25	27456	65	206	10	8770	FALSE	97.07562	0.966238	2.924376	0.970756
#26	26900	53	247	28	4570	TRUE	97.03216	0.965559	2.967838	0.970322
#27	23607	36	316	36	2230	TRUE	97.01354	0.96646	2.986465	0.970135
#28	29100	19	21	46	2317	FALSE	97.00112	0.965675	2.998882	0.970011
#29	27670	79	260	169	5114	FALSE	96.9204	0.964353	3.079598	0.969204
#30	18102	52	231	13	2594	FALSE	96.90798	0.966034	3.092015	0.96908
#31	18914	79	264	48	3679	FALSE	96.89557	0.963637	3.104433	0.968956

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy	F1 score:	Loss	Recall
#32	28724	51	325	132	577	FALSE	96.87694	0.964205	3.12306	0.968769
#33	26319	94	186	41	7214	TRUE	96.86452	0.963922	3.135477	0.968645
#34	18075	70	725	19	1786	FALSE	96.8521	0.963496	3.147895	0.968521
#35	21898	65	582	37	1648	FALSE	96.83348	0.963453	3.166522	0.968335
#36	28006	90	567	48	681	FALSE	96.79002	0.964715	3.209984	0.9679
#37	23586	59	658	60	3060	TRUE	96.43611	0.959348	3.563889	0.964361
#38	27579	75	375	157	4345	FALSE	96.4299	0.959748	3.570098	0.964299
#39	28194	67	1108	53	1363	FALSE	96.29331	0.958074	3.706693	0.962933
#40	28820	61	692	116	2800	FALSE	96.04495	0.955882	3.955048	0.96045
#41	29282	82	467	186	2729	FALSE	95.77797	0.953316	4.222029	0.95778
#42	21872	81	547	250	4383	FALSE	94.87148	0.943643	5.128524	0.948715
#43	28857	53	695	229	2305	TRUE	94.69763	0.943166	5.302372	0.946976
#44	9791	61	1546	53	6167	FALSE	92.91568	0.923898	7.084316	0.929157
#45	3490	39	1370	42	8140	TRUE	92.69837	0.922614	7.301627	0.926984
#46	26625	14	670	36	9135	TRUE	92.55557	0.921099	7.444431	0.925556
#47	8074	30	1472	55	61	TRUE	92.0899	0.920066	7.910096	0.920899
#48	24572	49	5736	38	5718	TRUE	89.43872	0.887709	10.56128	0.894387
#49	13422	72	1505	190	3581	TRUE	88.66882	0.88378	11.33118	0.886688
#50	12183	64	1315	218	5097	TRUE	86.72544	0.86485	13.27456	0.867254
#51	17744	62	8079	47	8550	FALSE	86.6944	0.859332	13.3056	0.866944
#52	4280	86	6569	53	595	FALSE	84.93108	0.849003	15.06892	0.849311
#53	13084	73	8317	35	6850	FALSE	84.04321	0.830509	15.95679	0.840432
#54	12596	38	3746	66	3778	TRUE	80.46691	0.798077	19.53309	0.804669
#55	28374	65	2436	186	8161	TRUE	79.21272	0.792139	20.78728	0.792127
#56	22694	54	5902	99	2511	FALSE	77.44319	0.778826	22.55681	0.774432
#57	3334	74	7011	63	8925	TRUE	76.15174	0.775968	23.84826	0.761517
#58	8172	91	2761	168	6629	TRUE	75.17695	0.77718	24.82305	0.75177
#59	28951	56	6740	93	4071	TRUE	72.43884	0.712204	27.56116	0.724388
#60	22952	23	7362	24	2434	FALSE	71.37713	0.6845	28.62287	0.713771
#61	20066	31	5696	87	6912	TRUE	68.25407	0.658527	31.74593	0.682541
#62	29014	68	7875	114	8415	TRUE	67.65802	0.669478	32.34198	0.67658

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy	F1 score:	Loss	Recall
#63	9216	89	5316	138	3299	FALSE	66.92537	0.66047	33.07463	0.669254
#64	25882	79	9057	128	1700	TRUE	66.77015	0.662109	33.22985	0.667701
#65	21369	81	8753	115	3216	TRUE	65.26139	0.649192	34.73861	0.652614
#66	25911	54	6337	123	8503	FALSE	64.2183	0.631919	35.7817	0.642183
#67	19622	47	4207	167	9952	FALSE	63.43599	0.636963	36.56401	0.63436
#68	2340	55	7999	57	860	TRUE	62.41152	0.638722	37.58848	0.624115
#69	13752	95	4748	193	8067	TRUE	62.27493	0.647102	37.72507	0.622749
#70	21026	48	8388	124	267	FALSE	59.25742	0.598463	40.74258	0.592574
#71	2291	64	9223	67	5310	TRUE	54.54489	0.541548	45.45511	0.545449
#72	11656	95	8203	205	1966	TRUE	50.4222	0.523001	49.5778	0.504222
#73	5846	33	8134	96	7389	FALSE	48.03179	0.5228	51.96821	0.480318
#74	11780	34	6943	157	6590	FALSE	47.87036	0.467053	52.12964	0.478704
#75	5121	68	7198	208	5781	TRUE	40.6184	0.480552	59.3816	0.406184
#76	3390	70	9820	167	4168	TRUE	37.90513	0.388902	62.09487	0.379051
#77	2252	34	7041	239	7062	TRUE	36.65094	0.371645	63.34906	0.366509
#78	2098	11	3904	77	1119	TRUE	35.57059	0.398953	64.42941	0.355706
#79	1768	12	5956	221	7805	FALSE	13.20005	0.187566	86.79995	0.132

Table B.2: Results from the initial sweep on CICIDS-2017 with class size cutoff 5000

B.2 KDD99

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#1	49883	65	20	25	12289	TRUE	99.87122	0.128783	0.999033	0.998712
#2	19005	67	24	9	43811	TRUE	99.80683	0.193175	0.998066	0.998068
#3	31078	51	5	39	13756	TRUE	99.74243	0.257566	0.998066	0.997424
#4	45747	74	88	14	33817	TRUE	99.74243	0.257566	0.997743	0.997424
#5	46718	67	22	33	22668	TRUE	99.74243	0.257566	0.998386	0.997424
#6	24184	64	12	13	43198	FALSE	99.74243	0.257566	0.998066	0.997424
#7	43332	50	85	9	24583	FALSE	99.67804	0.321958	0.997421	0.99678
#8	46086	64	25	29	14505	TRUE	99.67804	0.321958	0.997422	0.99678
#9	28209	74	210	10	18708	TRUE	99.67804	0.321958	0.996782	0.99678
#10	48282	63	44	10	5913	TRUE	99.67804	0.321958	0.997424	0.99678
#11	24830	63	45	10	4532	TRUE	99.67804	0.321958	0.997096	0.99678
#12	27124	49	26	12	15210	FALSE	99.67804	0.321958	0.997096	0.99678
#13	48780	68	13	31	21048	TRUE	99.61365	0.386349	0.996456	0.996137
#14	39279	69	35	34	41858	TRUE	99.61365	0.386349	0.996454	0.996137
#15	38533	72	113	9	14489	TRUE	99.61365	0.386349	0.996456	0.996137
#16	42787	42	21	11	20109	TRUE	99.61365	0.386349	0.997092	0.996137
#17	29826	63	26	26	1223	TRUE	99.61365	0.386349	0.996136	0.996137
#18	47465	54	18	22	32719	TRUE	99.61365	0.386349	0.996769	0.996137
#19	30296	18	10	21	2501	FALSE	99.61365	0.386349	0.996459	0.996137
#20	35700	48	147	17	9644	TRUE	99.61365	0.386349	0.997417	0.996137
#21	24184	71	48	30	11990	TRUE	99.61365	0.386349	0.996133	0.996137
#22	41682	73	15	30	4347	TRUE	99.61365	0.386349	0.996448	0.996137
#23	36066	64	12	35	41318	TRUE	99.61365	0.386349	0.996777	0.996137
#24	44919	73	300	8	7346	TRUE	99.61365	0.386349	0.996459	0.996137
#25	45121	57	24	67	49077	TRUE	99.61365	0.386349	0.997098	0.996137
#26	27896	61	25	48	1701	FALSE	99.61365	0.386349	0.996773	0.996137
#27	24571	66	30	43	29255	TRUE	99.54926	0.450741	0.995808	0.995493
#28	15141	68	75	28	469	TRUE	99.54926	0.450741	0.99709	0.995493
#29	43350	70	57	21	1139	TRUE	99.54926	0.450741	0.996771	0.995493
#30	33704	74	11	13	33800	FALSE	99.54926	0.450741	0.997417	0.995493
#31	48411	69	105	14	39276	TRUE	99.54926	0.450741	0.996118	0.995493

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#32	23806	65	5	46	1687	TRUE	99.54926	0.450741	0.996767	0.995493
#33	20863	51	20	47	13772	TRUE	99.54926	0.450741	0.99611	0.995493
#34	44423	55	12	42	6767	TRUE	99.54926	0.450741	0.995821	0.995493
#35	42735	60	187	18	39981	TRUE	99.54926	0.450741	0.996129	0.995493
#36	20337	35	19	22	1266	FALSE	99.54926	0.450741	0.996127	0.995493
#37	13566	68	7	28	25459	FALSE	99.54926	0.450741	0.995802	0.995493
#38	24145	75	35	49	3806	TRUE	99.54926	0.450741	0.996133	0.995493
#39	33922	75	64	16	31990	FALSE	99.54926	0.450741	0.99645	0.995493
#40	41188	63	16	30	23444	TRUE	99.54926	0.450741	0.996136	0.995493
#41	45632	61	76	22	45497	TRUE	99.54926	0.450741	0.997086	0.995493
#42	18833	73	6	26	8226	TRUE	99.54926	0.450741	0.995806	0.995493
#43	42796	65	85	23	1195	TRUE	99.48487	0.515132	0.996098	0.994849
#44	36389	74	7	35	43164	TRUE	99.48487	0.515132	0.995162	0.994849
#45	40021	60	26	12	28804	TRUE	99.48487	0.515132	0.995184	0.994849
#46	27713	17	7	19	11163	FALSE	99.48487	0.515132	0.995488	0.994849
#47	15422	71	13	12	46604	TRUE	99.48487	0.515132	0.994831	0.994849
#48	49942	40	19	41	5198	TRUE	99.48487	0.515132	0.996437	0.994849
#49	42205	64	26	19	37218	TRUE	99.48487	0.515132	0.995164	0.994849
#50	9501	49	35	26	3105	FALSE	99.48487	0.515132	0.995468	0.994849
#51	37031	75	129	9	19149	TRUE	99.48487	0.515132	0.995475	0.994849
#52	41171	53	22	63	22526	FALSE	99.48487	0.515132	0.995481	0.994849
#53	38555	67	37	23	14610	TRUE	99.42048	0.579524	0.994533	0.994205
#54	40967	65	8	8	32809	TRUE	99.42048	0.579524	0.994822	0.994205
#55	46208	73	45	26	47251	FALSE	99.42048	0.579524	0.994191	0.994205
#56	33945	58	52	43	37011	TRUE	99.42048	0.579524	0.99452	0.994205
#57	8080	63	11	27	13202	TRUE	99.42048	0.579524	0.994524	0.994205
#58	23690	70	140	30	6913	TRUE	99.42048	0.579524	0.994845	0.994205
#59	42705	75	22	28	26540	FALSE	99.42048	0.579524	0.99612	0.994205
#60	32073	73	16	21	16655	TRUE	99.42048	0.579524	0.994512	0.994205
#61	44460	61	5	29	14319	TRUE	99.42048	0.579524	0.994516	0.994205
#62	13770	67	34	14	27504	FALSE	99.42048	0.579524	0.995158	0.994205

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#63	29351	54	33	15	10152	FALSE	99.42048	0.579524	0.995158	0.994205
#64	49732	73	139	15	47556	FALSE	99.42048	0.579524	0.994518	0.994205
#65	41739	56	32	54	34794	TRUE	99.42048	0.579524	0.994841	0.994205
#66	31207	46	28	95	1111	FALSE	99.42048	0.579524	0.994847	0.994205
#67	38265	69	32	38	28094	TRUE	99.42048	0.579524	0.99517	0.994205
#68	45818	31	5	48	1259	FALSE	99.42048	0.579524	0.994843	0.994205
#69	16402	69	26	12	3909	TRUE	99.42048	0.579524	0.995139	0.994205
#70	35655	64	66	49	7537	TRUE	99.42048	0.579524	0.995487	0.994205
#71	47634	54	28	35	47330	TRUE	99.42048	0.579524	0.99485	0.994205
#72	16110	73	130	20	34035	TRUE	99.42048	0.579524	0.994839	0.994205
#73	25865	68	18	25	3930	TRUE	99.42048	0.579524	0.995152	0.994205
#74	7987	75	22	34	46115	TRUE	99.42048	0.579524	0.995149	0.994205
#75	14362	74	80	25	25832	TRUE	99.42048	0.579524	0.994204	0.994205
#76	23522	69	23	51	3817	FALSE	99.42048	0.579524	0.994524	0.994205
#77	4874	72	59	13	2402	FALSE	99.42048	0.579524	0.994175	0.994205
#78	43023	61	157	30	13190	TRUE	99.35608	0.643915	0.994193	0.993561
#79	23296	48	69	12	9732	TRUE	99.35608	0.643915	0.994516	0.993561
#80	2928	68	29	50	1274	FALSE	99.35608	0.643915	0.995143	0.993561
#81	17461	68	25	9	32235	TRUE	99.35608	0.643915	0.994818	0.993561
#82	16989	75	52	16	17407	TRUE	99.35608	0.643915	0.994186	0.993561
#83	16558	65	14	18	7545	TRUE	99.35608	0.643915	0.994203	0.993561
#84	47448	39	79	26	27111	TRUE	99.35608	0.643915	0.993885	0.993561
#85	39494	74	131	29	7267	TRUE	99.35608	0.643915	0.99388	0.993561
#86	41066	69	108	12	8649	TRUE	99.35608	0.643915	0.994192	0.993561
#87	37363	48	20	56	23658	FALSE	99.35608	0.643915	0.995164	0.993561
#88	39251	45	20	20	28110	FALSE	99.35608	0.643915	0.995149	0.993561
#89	37956	56	131	17	8814	TRUE	99.35608	0.643915	0.994216	0.993561
#90	13092	70	13	11	7361	TRUE	99.35608	0.643915	0.994837	0.993561
#91	31631	74	91	40	13009	TRUE	99.35608	0.643915	0.993557	0.993561
#92	37617	62	62	23	6155	TRUE	99.35608	0.643915	0.995481	0.993561
#93	29279	41	34	25	15984	TRUE	99.35608	0.643915	0.994522	0.993561

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#94	27835	67	158	16	29687	TRUE	99.35608	0.643915	0.994195	0.993561
#95	24846	55	104	18	4552	TRUE	99.35608	0.643915	0.994828	0.993561
#96	44773	74	36	46	48097	TRUE	99.35608	0.643915	0.993869	0.993561
#97	42955	51	32	35	24762	TRUE	99.35608	0.643915	0.994522	0.993561
#98	17978	73	23	81	20991	TRUE	99.35608	0.643915	0.99517	0.993561
#99	40912	75	21	55	32084	TRUE	99.35608	0.643915	0.994203	0.993561
#100	37574	72	172	11	25434	FALSE	99.35608	0.643915	0.994201	0.993561

Table B.3: Sweep table results

B.3 NSL-KDD

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score	Recall
#1	2183	20	115	10	49662	FALSE	85.55922	14.44078	0.803969	0.855592
#2	1440	18	70	13	48130	FALSE	85.28254	14.71746	0.801377	0.852825
#3	9341	44	82	11	46233	TRUE	85.20805	14.79195	0.800913	0.85208
#4	3703	56	208	8	8789	TRUE	85.19208	14.80792	0.800792	0.851921
#5	4700	10	73	11	47655	FALSE	85.1708	14.8292	0.800286	0.851708
#6	4756	56	47	23	48567	FALSE	85.13887	14.86113	0.800549	0.851389
#7	21537	56	239	13	34971	TRUE	85.13887	14.86113	0.80022	0.851389
#8	5409	57	66	21	33799	TRUE	85.11759	14.88241	0.800219	0.851176
#9	5212	52	95	12	40357	TRUE	85.10695	14.89305	0.799922	0.851069
#10	9277	23	110	13	40650	FALSE	85.08567	14.91433	0.799509	0.850857
#11	7331	14	46	21	49342	FALSE	85.05374	14.94626	0.799273	0.850537
#12	21064	60	54	27	48564	TRUE	85.04842	14.95158	0.799667	0.850484
#13	7633	26	33	21	47381	TRUE	85.02714	14.97286	0.799128	0.850271
#14	13835	50	133	8	49159	TRUE	85.01649	14.98351	0.799132	0.850165
#15	25550	27	152	41	29848	FALSE	85.00585	14.99415	0.799053	0.850059
#16	34817	39	62	20	47165	FALSE	84.97925	15.02075	0.798736	0.849792
#17	22362	59	223	9	49654	FALSE	84.96861	15.03139	0.798446	0.849686
#18	29634	32	81	19	37313	FALSE	84.96329	15.03671	0.798702	0.849633
#19	16219	32	25	15	45837	FALSE	84.95797	15.04203	0.798366	0.84958
#20	2414	5	65	33	43097	FALSE	84.93136	15.06864	0.796775	0.849314
#21	22479	59	131	44	38596	TRUE	84.92604	15.07396	0.798257	0.84926
#22	8573	41	102	11	49296	FALSE	84.88347	15.11653	0.797511	0.848835
#23	14335	18	27	38	47879	FALSE	84.87283	15.12717	0.797782	0.848728
#24	5437	26	42	10	41939	FALSE	84.86751	15.13249	0.797254	0.848675
#25	8123	41	113	13	46326	TRUE	84.86219	15.13781	0.797421	0.848622
#26	24909	23	18	18	38196	FALSE	84.85687	15.14313	0.797757	0.848569
#27	30574	36	50	39	39554	FALSE	84.84091	15.15909	0.797417	0.848409
#28	22556	33	38	35	49184	TRUE	84.83559	15.16441	0.797445	0.848356
#29	6936	58	40	8	19135	TRUE	84.82494	15.17506	0.797338	0.848249
#30	1683	60	242	11	37021	FALSE	84.81962	15.18038	0.796563	0.848196
#31	46376	21	50	9	41094	TRUE	84.81962	15.18038	0.797333	0.848196

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score	Recall
#32	3845	54	178	14	28020	TRUE	84.81962	15.18038	0.79694	0.848196
#33	39011	46	65	44	40946	FALSE	84.80898	15.19102	0.797078	0.84809
#34	49382	11	28	19	47783	TRUE	84.80366	15.19634	0.796965	0.848037
#35	12018	57	47	32	48855	TRUE	84.80366	15.19634	0.797027	0.848037
#36	22793	53	358	12	44669	TRUE	84.79834	15.20166	0.796788	0.847983
#37	23196	58	59	25	42749	TRUE	84.77174	15.22826	0.796825	0.847717
#38	13142	48	19	9	49229	FALSE	84.76109	15.23891	0.796037	0.847611
#39	32166	60	114	9	48811	TRUE	84.75045	15.24955	0.796486	0.847505
#40	30314	60	93	17	36035	TRUE	84.75045	15.24955	0.796613	0.847505
#41	38997	55	472	53	49697	TRUE	84.74513	15.25487	0.79629	0.847451
#42	4151	50	25	20	42635	FALSE	84.74513	15.25487	0.796047	0.847451
#43	9142	19	40	40	40851	FALSE	84.73981	15.26019	0.796471	0.847398
#44	19702	50	168	14	40267	TRUE	84.72385	15.27615	0.79606	0.847238
#45	41794	43	100	28	42759	FALSE	84.71853	15.28147	0.796245	0.847185
#46	44255	15	134	20	46210	TRUE	84.70256	15.29744	0.795821	0.847026
#47	45535	48	174	14	21524	TRUE	84.69724	15.30276	0.79611	0.846972
#48	46653	50	413	33	44093	TRUE	84.6866	15.3134	0.795676	0.846866
#49	24448	20	160	14	31998	FALSE	84.67064	15.32936	0.795356	0.846706
#50	49804	25	35	13	28976	TRUE	84.67064	15.32936	0.795599	0.846706
#51	34140	40	334	17	45690	TRUE	84.66532	15.33468	0.795277	0.846653
#52	27215	29	101	17	30239	FALSE	84.66	15.34	0.795439	0.8466
#53	39673	34	36	11	49459	FALSE	84.64936	15.35064	0.795146	0.846494
#54	42439	44	428	23	43948	TRUE	84.61211	15.38789	0.794683	0.846121
#55	30132	33	126	37	39436	FALSE	84.60679	15.39321	0.795444	0.846068
#56	40301	50	49	8	43280	TRUE	84.60147	15.39853	0.794878	0.846015
#57	6344	57	345	12	48776	TRUE	84.56422	15.43578	0.794206	0.845642
#58	18410	36	98	33	39410	FALSE	84.56422	15.43578	0.794787	0.845642
#59	12599	28	15	13	41572	FALSE	84.54826	15.45174	0.793875	0.845483
#60	2279	54	157	10	17933	TRUE	84.54294	15.45706	0.793911	0.845429
#61	18007	26	154	15	40121	FALSE	84.5323	15.4677	0.793894	0.845323
#62	20252	25	233	30	46632	FALSE	84.52166	15.47834	0.79373	0.845217

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score	Recall
#63	19589	47	22	9	43751	TRUE	84.51634	15.48366	0.793347	0.845163
#64	31091	46	75	49	29871	FALSE	84.49505	15.50495	0.793981	0.844951
#65	38257	59	345	51	49653	TRUE	84.49505	15.50495	0.793792	0.844951
#66	39267	39	44	12	42185	TRUE	84.48441	15.51559	0.793646	0.844844
#67	3259	50	243	15	20432	TRUE	84.48441	15.51559	0.793245	0.844844
#68	23328	34	117	48	33978	FALSE	84.46313	15.53687	0.793453	0.844631
#69	19788	10	12	30	49605	TRUE	84.45248	15.54752	0.79288	0.844525
#70	14995	59	23	19	44270	TRUE	84.45248	15.54752	0.792891	0.844525
#71	25889	37	114	27	38536	FALSE	84.45248	15.54752	0.793306	0.844525
#72	40773	40	289	15	43456	TRUE	84.44184	15.55816	0.792871	0.844418
#73	2728	59	199	48	14642	TRUE	84.4046	15.5954	0.792851	0.844046
#74	22845	23	135	42	24886	FALSE	84.39928	15.60072	0.792975	0.843993
#75	44913	9	8	11	23586	TRUE	84.37799	15.62201	0.791993	0.84378
#76	6796	57	358	14	905	TRUE	84.36203	15.63797	0.792606	0.84362
#77	3910	17	125	29	46894	TRUE	84.34075	15.65925	0.791768	0.843407
#78	47514	45	463	12	37523	TRUE	84.32478	15.67522	0.791632	0.843248
#79	32678	20	130	54	38507	FALSE	84.31414	15.68586	0.791712	0.843141
#80	42659	56	40	53	48790	TRUE	84.23965	15.76035	0.791183	0.842397
#81	5734	18	17	53	47092	FALSE	84.22901	15.77099	0.791005	0.84229
#82	30410	31	143	73	36424	FALSE	84.20773	15.79227	0.790901	0.842077
#83	5417	56	67	55	12335	TRUE	84.19176	15.80824	0.791069	0.841918
#84	46756	30	53	59	47368	TRUE	84.18112	15.81888	0.790516	0.841811
#85	15191	58	248	22	27959	TRUE	84.1758	15.8242	0.790192	0.841758
#86	26922	13	456	46	33197	TRUE	84.11195	15.88805	0.788879	0.84112
#87	33662	31	146	47	22679	FALSE	84.10663	15.89337	0.789865	0.841066
#88	36704	56	23	66	43921	FALSE	84.10131	15.89869	0.789529	0.841013
#89	13175	8	163	44	46337	FALSE	84.08003	15.91997	0.788494	0.8408
#90	40387	50	495	73	45977	TRUE	84.0747	15.9253	0.788563	0.840747
#91	39490	34	30	94	47007	FALSE	84.0481	15.9519	0.789289	0.840481
#92	33553	59	311	54	23999	TRUE	84.02682	15.97318	0.789412	0.840268
#93	44187	25	335	21	49092	TRUE	83.98425	16.01575	0.788006	0.839843

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score	Recall
#94	22499	17	476	57	32337	TRUE	83.92572	16.07428	0.786709	0.839257
#95	22849	27	311	30	26296	FALSE	83.84059	16.15941	0.78628	0.838406
#96	1706	51	82	78	47735	FALSE	83.72885	16.27115	0.785836	0.837288
#97	13874	60	73	17	74	TRUE	83.70225	16.29775	0.789943	0.837022
#98	9505	34	57	85	44267	TRUE	83.68096	16.31904	0.785778	0.83681
#99	49448	14	284	181	21634	FALSE	82.94669	17.05331	0.772306	0.829467
#100	2821	5	492	255	6274	TRUE	77.57795	22.42205	0.735649	0.77578

Table B.4: Results from hyperparameter Sweep on NSL-KDD dataset

B.4 UNSW NB-15

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy	F1 score:	Loss	Recall
#1	44985	41	347	112	1769	FALSE	82.10986	0.817357	17.89014	0.821099
#2	30525	66	481	170	4629	FALSE	81.22101	0.806783	18.77899	0.81221
#3	28107	90	278	133	3711	FALSE	81.94222	0.816793	18.05778	0.819422
#4	17983	69	60	83	2969	TRUE	81.89934	0.82115	18.10066	0.818993
#5	15026	23	126	59	4125	FALSE	81.92273	0.814492	18.07727	0.819227
#6	21247	67	321	144	1879	TRUE	81.9968	0.816739	18.0032	0.819968
#7	33429	60	66	253	4872	TRUE	82.2541	0.82187	17.7459	0.822541
#8	20446	38	37	30	3707	FALSE	82.23461	0.818815	17.76539	0.822346
#9	38518	46	347	199	4863	FALSE	80.81946	0.803873	19.18054	0.808195
#10	22344	42	107	68	3161	FALSE	82.43343	0.81667	17.56657	0.824334
#11	15302	53	133	10	3016	FALSE	82.49581	0.820695	17.50419	0.824958
#12	18231	93	224	155	1865	FALSE	81.87985	0.81325	18.12015	0.818798
#13	40946	41	189	126	4029	FALSE	82.24241	0.816708	17.75759	0.822424
#14	2383	47	23	76	1714	FALSE	82.09037	0.815976	17.90963	0.820904
#15	32756	52	461	153	1986	TRUE	81.42762	0.810679	18.57238	0.814276
#16	45743	8	430	40	1460	TRUE	80.05536	0.791241	19.94464	0.800554
#17	18843	19	188	119	4423	TRUE	81.64594	0.809624	18.35406	0.816459
#18	16325	46	172	53	4266	FALSE	82.17224	0.818207	17.82776	0.821722
#19	30546	74	204	23	3894	TRUE	82.527	0.820177	17.473	0.82527
#20	12792	31	27	17	3470	FALSE	82.35936	0.819247	17.64064	0.823594
#21	22118	54	207	11	3916	FALSE	82.12935	0.817615	17.87065	0.821294
#22	39923	98	345	184	2975	FALSE	82.0007	0.814888	17.9993	0.820007
#23	23689	50	114	70	2325	FALSE	82.23461	0.818945	17.76539	0.822346
#24	42324	57	23	243	1668	FALSE	81.17422	0.811305	18.82578	0.811742
#25	35776	31	17	50	3104	FALSE	81.7434	0.812555	18.2566	0.817434
#26	9696	54	138	14	4277	FALSE	82.39835	0.820622	17.60165	0.823983
#27	17905	64	57	15	3973	FALSE	82.08647	0.815781	17.91353	0.820865
#28	16997	17	47	103	3950	FALSE	82.31648	0.818757	17.68352	0.823165
#29	15497	49	193	70	4568	FALSE	82.60887	0.819861	17.39113	0.826089
#30	38478	61	296	77	2461	TRUE	81.62255	0.81189	18.37745	0.816225
#31	44621	95	331	95	2987	TRUE	81.73171	0.812896	18.26829	0.817317

#32	26562	9	10	42	3919	FALSE	81.70052	0.812942	18.29948	0.817005
#33	8342	59	162	27	3574	FALSE	82.19173	0.815344	17.80827	0.821917
#34	15661	18	123	205	3198	FALSE	81.80968	0.812343	18.19032	0.818097
#35	12333	93	342	86	2033	FALSE	82.08257	0.815959	17.91743	0.820826
#36	27397	92	76	192	2364	TRUE	81.68882	0.814244	18.31118	0.816888
#37	23122	64	35	18	2076	FALSE	82.0085	0.816491	17.9915	0.820085
#38	16783	12	240	244	2341	TRUE	79.62263	0.788766	20.37737	0.796226
#39	31235	99	146	239	1446	FALSE	81.41983	0.809846	18.58017	0.814198
#40	21981	20	33	60	2641	FALSE	81.82137	0.813386	18.17863	0.818214
#41	23849	9	255	27	2175	TRUE	80.70251	0.796039	19.29749	0.807025
#42	19092	50	91	141	4893	FALSE	82.15274	0.818582	17.84726	0.821527
#43	41624	56	368	97	963	TRUE	81.14304	0.810938	18.85696	0.81143
#44	20069	38	151	156	2128	FALSE	82.258	0.819931	17.742	0.82258
#45	17902	38	347	65	4222	FALSE	81.68492	0.810412	18.31508	0.816849
#46	15778	52	110	17	2241	FALSE	82.40614	0.822804	17.59386	0.824061
#47	36748	51	418	208	117	TRUE	78.42969	0.782542	21.57031	0.784297
#48	10756	71	176	31	1609	TRUE	82.62446	0.823803	17.37554	0.826245
#49	10405	52	148	77	4697	FALSE	81.76289	0.812204	18.23711	0.817629
#50	5859	57	344	95	1932	TRUE	81.27948	0.809106	18.72052	0.812795
#51	42156	20	244	30	4045	FALSE	81.46271	0.809485	18.53729	0.814627
#52	18077	32	162	10	4304	FALSE	81.66933	0.809074	18.33067	0.816693
#53	18137	43	74	65	4826	FALSE	81.89934	0.813854	18.10066	0.818993
#54	27035	56	132	35	3131	FALSE	82.74921	0.825193	17.25079	0.827492
#55	19277	55	46	186	4475	TRUE	81.93833	0.814199	18.06167	0.819383
#56	10969	54	73	30	4654	FALSE	82.44903	0.82076	17.55097	0.82449
#57	46926	97	128	39	4470	FALSE	81.94222	0.818011	18.05778	0.819422
#58	5924	48	254	78	4219	FALSE	81.9968	0.815975	18.0032	0.819968
#59	1903	38	162	12	4329	FALSE	81.31067	0.810009	18.68933	0.813107
#60	15342	34	29	71	4321	FALSE	82.258	0.819316	17.742	0.82258
#61	25875	37	96	48	3228	FALSE	81.97731	0.822478	18.02269	0.819773
#62	7593	37	38	8	2421	FALSE	81.97731	0.812231	18.02269	0.819773

#63	20248	43	233	33	2081	FALSE	81.44322	0.808439	18.55678	0.814432
#64	36004	84	377	75	827	TRUE	81.77849	0.812635	18.22151	0.817785
#65	27107	19	54	180	392	TRUE	81.01828	0.807501	18.98172	0.810183
#66	13745	62	80	59	3026	FALSE	82.32038	0.815708	17.67962	0.823204
#67	18347	88	214	33	4936	FALSE	81.20541	0.802473	18.79459	0.812054
#68	49762	57	364	149	4301	FALSE	81.36525	0.810925	18.63475	0.813652
#69	35770	11	90	56	366	FALSE	80.55047	0.799825	19.44953	0.805505
#70	41373	94	318	181	2787	TRUE	82.13715	0.816769	17.86285	0.821371
#71	7759	27	399	141	2618	TRUE	79.7006	0.789006	20.2994	0.797006
#72	16823	92	176	246	2164	TRUE	81.58746	0.814311	18.41254	0.815875
#73	13445	43	95	25	3580	FALSE	82.38665	0.818766	17.61335	0.823867
#74	44542	100	297	254	4904	FALSE	81.73171	0.814764	18.26829	0.817317
#75	24898	77	214	182	3121	FALSE	81.64984	0.812035	18.35016	0.816498
#76	14126	47	202	109	4074	FALSE	81.96951	0.81624	18.03049	0.819695
#77	22314	15	97	176	2774	FALSE	81.96172	0.813653	18.03828	0.819617
#78	5160	24	453	181	3960	TRUE	78.44918	0.770638	21.55082	0.784492
#79	10127	76	26	37	3485	FALSE	82.18783	0.818519	17.81217	0.821878
#80	34367	30	151	137	3952	FALSE	82.02409	0.815906	17.97591	0.820241
#81	43701	24	157	70	3761	TRUE	81.37694	0.810575	18.62306	0.813769
#82	2874	28	72	206	2679	TRUE	81.51339	0.810594	18.48661	0.815134
#83	1569	33	301	215	399	TRUE	80.23859	0.798057	19.76141	0.802386
#84	16652	85	439	209	4477	FALSE	81.01828	0.803725	18.98172	0.810183
#85	16600	81	78	241	3920	FALSE	82.41004	0.817109	17.58996	0.8241
#86	14414	75	445	225	978	TRUE	81.39644	0.808327	18.60356	0.813964
#87	20764	23	38	52	3400	FALSE	82.70243	0.82441	17.29757	0.827024
#88	8011	17	18	201	4801	TRUE	82.22292	0.814569	17.77708	0.822229
#89	17971	43	106	21	3507	FALSE	82.35936	0.81893	17.64064	0.823594
#90	1840	24	348	228	2318	FALSE	77.29913	0.759034	22.70087	0.772991
#91	17400	61	65	33	47	TRUE	77.20947	0.767337	22.79053	0.772095
#92	36813	72	418	51	1722	TRUE	81.82137	0.815118	18.17863	0.818214
#93	20026	34	30	76	3213	FALSE	82.03969	0.816276	17.96031	0.820397

#94	22413	52	124	31	3224	FALSE	82.10206	0.820402	17.89794	0.821021
#95	9683	58	115	227	4072	FALSE	82.07477	0.81735	17.92523	0.820748
#96	1880	92	256	98	345	FALSE	81.17032	0.805908	18.82968	0.811703

Table B.5: Results from Sweeping over the UNSW-NB15 dataset

B.5 UNSW Bot-IoT

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#1	32288	73	24	13	3962	FALSE	100	0	1	1
#2	42252	85	55	9	4291	FALSE	100	0	1	1
#3	34273	28	40	11	2188	FALSE	100	0	1	1
#4	33440	59	10	12	2332	FALSE	100	0	1	1
#5	39941	86	94	17	4935	FALSE	100	0	1	1
#6	33794	43	82	12	2269	FALSE	100	0	1	1
#7	48630	71	189	18	1102	TRUE	100	0	1	1
#8	49603	69	142	9	4854	TRUE	100	0	1	1
#9	42009	28	21	8	2595	TRUE	100	0	1	1
#10	36982	91	184	17	1037	TRUE	100	0	1	1
#11	46273	94	141	10	4128	TRUE	100	0	1	1
#12	45590	74	73	9	2113	TRUE	100	0	1	1
#13	43385	34	16	11	1672	TRUE	100	0	1	1
#14	35093	49	88	46	2009	TRUE	100	0	1	1
#15	48847	52	10	18	829	TRUE	100	0	1	1
#16	40440	60	132	8	1144	TRUE	100	0	1	1
#17	20793	56	52	24	354	TRUE	100	0	1	1
#18	35505	44	66	28	558	TRUE	100	0	1	1
#19	40822	76	228	8	818	TRUE	100	0	1	1
#20	10527	63	29	8	971	FALSE	99.99088	0.009118	0.999908	0.999909
#21	48452	69	82	11	4869	FALSE	99.99088	0.009118	0.999908	0.999909
#22	47007	39	48	8	1828	FALSE	99.99088	0.009118	0.999909	0.999909
#23	49384	59	203	8	1483	TRUE	99.99088	0.009118	0.999909	0.999909
#24	46441	45	30	13	1473	TRUE	99.99088	0.009118	0.999908	0.999909
#25	44380	60	169	22	2068	TRUE	99.99088	0.009118	0.999909	0.999909
#26	34857	78	54	13	2190	FALSE	99.99088	0.009118	0.999909	0.999909
#27	33110	80	10	27	4321	FALSE	99.99088	0.009118	0.999908	0.999909
#28	36230	90	280	15	446	TRUE	99.99088	0.009118	0.999908	0.999909
#29	40057	80	186	16	4587	TRUE	99.99088	0.009118	0.999908	0.999909
#30	38814	39	16	10	2529	FALSE	99.99088	0.009118	0.999908	0.999909
#31	40902	44	22	15	1090	FALSE	99.99088	0.009118	0.999908	0.999909

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#32	34261	95	159	17	1421	TRUE	99.99088	0.009118	0.999908	0.999909
#33	37306	93	41	19	4843	TRUE	99.99088	0.009118	0.99991	0.999909
#34	34516	56	106	10	2465	TRUE	99.99088	0.009118	0.999908	0.999909
#35	35015	55	33	10	3021	TRUE	99.99088	0.009118	0.999908	0.999909
#36	43600	38	72	24	1681	TRUE	99.99088	0.009118	0.999908	0.999909
#37	42042	100	7	11	1977	TRUE	99.99088	0.009118	0.999908	0.999909
#38	30241	76	173	12	382	TRUE	99.99088	0.009118	0.999908	0.999909
#39	46438	47	14	36	2805	TRUE	99.99088	0.009118	0.999909	0.999909
#40	41102	56	132	23	2143	TRUE	99.99088	0.009118	0.999908	0.999909
#41	44810	83	218	11	826	TRUE	99.99088	0.009118	0.999908	0.999909
#42	48245	51	82	28	658	TRUE	99.99088	0.009118	0.999908	0.999909
#43	42213	77	257	18	175	TRUE	99.99088	0.009118	0.999909	0.999909
#44	44415	85	12	32	695	TRUE	99.99088	0.009118	0.999909	0.999909
#45	2223	97	60	70	525	FALSE	99.98176	0.018237	0.999814	0.999818
#46	18043	26	52	91	2786	TRUE	99.98176	0.018237	0.999817	0.999818
#47	22910	64	22	21	1991	FALSE	99.98176	0.018237	0.999814	0.999818
#48	44874	89	71	20	4331	FALSE	99.98176	0.018237	0.999817	0.999818
#49	44564	95	155	16	845	TRUE	99.98176	0.018237	0.999817	0.999818
#50	34323	71	118	8	1343	TRUE	99.98176	0.018237	0.999814	0.999818
#51	29198	39	38	13	3470	TRUE	99.98176	0.018237	0.999817	0.999818
#52	48942	89	308	12	4355	TRUE	99.98176	0.018237	0.999815	0.999818
#53	45186	46	40	9	182	TRUE	99.98176	0.018237	0.999817	0.999818
#54	32710	69	30	24	3289	TRUE	99.98176	0.018237	0.999816	0.999818
#55	33706	43	33	17	683	TRUE	99.98176	0.018237	0.999817	0.999818
#56	49345	31	6	41	130	TRUE	99.98176	0.018237	0.999818	0.999818
#57	32748	44	26	12	2168	FALSE	99.98176	0.018237	0.999814	0.999818
#58	40224	31	6	51	2597	TRUE	99.98176	0.018237	0.999814	0.999818
#59	44720	56	93	47	1589	TRUE	99.98176	0.018237	0.999814	0.999818
#60	49174	30	106	21	753	TRUE	99.98176	0.018237	0.999817	0.999818
#61	42437	64	43	32	1029	TRUE	99.98176	0.018237	0.999815	0.999818
#62	49356	69	31	9	1657	TRUE	99.98176	0.018237	0.999815	0.999818

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#63	46452	67	149	18	359	TRUE	99.98176	0.018237	0.999817	0.999818
#64	36443	66	243	19	1375	TRUE	99.98176	0.018237	0.999818	0.999818
#65	49390	83	9	41	343	TRUE	99.98176	0.018237	0.999814	0.999818
#66	27793	86	92	14	48	TRUE	99.98176	0.018237	0.999818	0.999818
#67	41468	72	55	41	113	TRUE	99.98176	0.018237	0.999817	0.999818
#68	39991	58	183	31	404	TRUE	99.98176	0.018237	0.999818	0.999818
#69	43399	69	13	26	815	TRUE	99.98176	0.018237	0.999817	0.999818
#70	43019	82	283	30	1090	TRUE	99.98176	0.018237	0.999817	0.999818
#71	44355	87	236	31	1670	TRUE	99.98176	0.018237	0.999817	0.999818
#72	34442	21	72	24	1619	FALSE	99.97265	0.027355	0.999724	0.999726
#73	40895	64	94	38	1544	TRUE	99.97265	0.027355	0.999726	0.999726
#74	37448	56	124	32	1111	TRUE	99.97265	0.027355	0.999725	0.999726
#75	46931	87	139	37	753	TRUE	99.97265	0.027355	0.999725	0.999726
#76	38381	27	114	23	745	TRUE	99.97265	0.027355	0.999722	0.999726
#77	30490	16	149	40	529	TRUE	99.97265	0.027355	0.999726	0.999726
#78	33268	64	35	35	543	TRUE	99.97265	0.027355	0.999724	0.999726
#79	43067	6	46	107	4819	FALSE	99.96353	0.036473	0.999627	0.999635
#80	37454	32	145	34	2736	FALSE	99.96353	0.036473	0.999628	0.999635
#81	47186	91	130	31	3615	TRUE	99.96353	0.036473	0.999634	0.999635
#82	16459	55	103	18	612	TRUE	99.96353	0.036473	0.999626	0.999635
#83	44662	99	257	69	1878	TRUE	99.96353	0.036473	0.999636	0.999635
#84	38146	78	179	122	1353	TRUE	99.95441	0.045591	0.999545	0.999544
#85	40584	56	46	21	2993	TRUE	99.95441	0.045591	0.999547	0.999544
#86	37490	81	167	42	215	TRUE	99.95441	0.045591	0.999544	0.999544
#87	48991	18	191	23	3124	TRUE	99.95441	0.045591	0.999544	0.999544
#88	49537	77	105	42	1373	TRUE	99.95441	0.045591	0.999548	0.999544
#89	22704	39	35	39	52	TRUE	99.95441	0.045591	0.999532	0.999544
#90	26805	31	162	210	1432	FALSE	99.94529	0.05471	0.999456	0.999453
#91	41224	89	156	58	4696	FALSE	99.94529	0.05471	0.999456	0.999453
#92	43314	70	208	14	131	TRUE	99.94529	0.05471	0.99945	0.999453
#93	48955	74	276	39	220	FALSE	99.90882	0.091183	0.999089	0.999088

Run	Clauses	Epochs	Forget_rate	Max_literals	Threshold	Weighted_Clauses	Accuracy (%)	Loss	F1 score:	Recall
#94	39281	96	469	48	260	TRUE	99.8997	0.100301	0.999015	0.998997
#95	20257	54	295	97	3354	FALSE	99.89058	0.109419	0.998901	0.998906
#96	44007	95	287	247	593	FALSE	99.89058	0.109419	0.99892	0.998906
#97	5987	99	284	104	4472	TRUE	99.87234	0.127656	0.998743	0.998723
#98	34197	39	228	147	701	TRUE	99.80852	0.191484	0.998096	0.998085
#99	13391	34	187	133	4610	FALSE	99.80852	0.191484	0.998049	0.998085
#100	19536	69	366	242	4519	TRUE	99.58968	0.410322	0.995633	0.995897
#101	5246	13	189	92	3460	FALSE	99.38908	0.610924	0.993329	0.993891
#102	9294	29	434	203	2875	FALSE	99.09729	0.902708	0.990637	0.990973

Table B.6: Results from hyperparameter sweep on the UNSW-Bot-IoT dataset

B.6 Hyperparameter Importance and Correlation

	CICIDS		KDD99		NSL-KDD		UNSW-NB15		UNSW-Bot-IoT	
Parameters	Importance	Correlation	Importance	Correlation	Importance	Correlation	Importance	Correlation	Importance	Correlation
Epochs	0.047	0.378	0.115	0.258	0.048	0.198	0.183	0.203	0.143	0.354
Clauses	0.155	0.648	0.272	-0.164	0.035	-0.011	0.115	0.047	0.162	0.432
Forget Rate	0.601	-0.809	0.120	-0.436	0.062	-0.371	0.251	-0.426	0.261	-0.557
Max Literals	0.129	-0.426	0.392	-0.588	0.697	-0.840	0.119	-0.285	0.183	-0.540
Threshold	0.014	-0.209	0.077	0.028	0.086	0.394	0.331	0.391	0.085	-0.287
Weighted Clauses True	0.008	-0.214	0.015	0.184	0.026	-0.105	0.039	-0.347	0.043	0.021
Weighted Clauses False	0.007	0.214	0.009	-0.184	0.007	0.105	0.038	0.347	0.040	-0.021

Table B.7: Hyperparameter importance and correlation

Appendix C

Dataset confusion matrices

This appendix contains the confusion matrices for the final runs for each dataset.

C.1 NSL-KDD

Prediction	True Category																				
	teardrop	normal	back	land	guess_passwd	satan	phf	ftp_write	multihop	loadmodule	smurf	neptune	portsweep	ipsweep	perl	imap	pod	warezmaster	buffer_overflow	rootkit	nmap
teardrop	8	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
normal	3	9446	53	0	1231	4	2	3	17	2	67	1	4	3	2	1	5	838	20	12	0
back	0	0	306	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
land	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
guess_passwd	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
satan	1	144	0	0	0	720	0	0	0	0	0	6	0	0	0	0	1	0	0	0	0
phf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ftp_write	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
multihop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
loadmodule	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
smurf	0	9	0	0	0	0	0	0	1	0	598	0	0	0	0	0	0	0	0	0	0
neptune	0	2	0	4	0	10	0	0	0	0	0	4646	0	0	0	0	0	0	0	0	59
portsweep	0	56	0	0	0	1	0	0	0	0	0	4	153	0	0	0	0	0	0	0	0
ipsweep	0	17	0	3	0	0	0	0	0	0	0	0	0	138	0	0	28	86	0	1	1
perl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
imap	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pod	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0
warezmaster	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
buffer_overflow	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rootkit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nmap	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	13

Table C.1: Confusion matrix from the final run of NSL-KDD

Prediction	True category				
	DDoS	Reconnaissance	DoS	Theft	Normal
DDoS	3628	0	0	0	0
Reconnaissance	0	3581	0	0	0
DoS	0	0	3593	1	0
Theft	0	0	0	25	0
Normal	0	0	0	1	138

Table C.2: Confusion Matrix for the final run of UNSW Bot-IoT

C.2 UNSW Bot-IoT

C.3 KDD99

Prediction	True Category																						
	back	loadmodule	portsweep	neptune	warezmaster	phf	rootkit	perl	smurf	nmap	multihop	guess_passwd	ftp_write	warezclient	ipsweep	teardrop	pod	land	satan	normal	imap	spy	buffer_overflow
back	172	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
loadmodule	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
portsweep	0	0	172	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neptune	0	0	0	172	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
warezmaster	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rootkit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
perl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
smurf	0	0	0	0	0	0	0	0	172	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nmap	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
multihop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
guess_passwd	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ftp_write	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
warezclient	0	0	0	0	0	0	0	0	0	0	0	0	0	173	0	0	0	0	0	2	0	0	0
ipsweep	0	0	0	0	0	0	0	0	0	0	0	0	0	0	171	0	0	0	0	2	0	0	0
teardrop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	173	0	0	0	0	0	0	0
pod	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
land	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
satan	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	0	0	0	0
normal	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	168	0	0	0
imap	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
spy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
buffer_overflow	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table C.3: Confusion matrix of the final run of KDD99

C.4 CIC-IDS2017

Prediction	True Category											
	SSH-Patator	FTP-Patator	DDoS	DoS GoldenEye	Heartbleed	DoS slowloris	BENIGN	DoS Slowhttptest	Bot	PortScan	DoS Hulk	Infiltration
Heartbleed	0	0	0	0	0	0	0	0	0	0	0	0
DoS slowloris	0	0	0	0	0	1562	0	6	0	0	0	0
FTP-Patator	0	2368	0	0	0	0	0	0	0	0	0	0
Web Attack - XSS	0	0	0	0	0	0	13	0	0	0	0	2
DoS Slowhttptest	0	0	0	1	0	39	0	1529	0	0	0	0
Infiltration	0	0	0	0	0	0	0	0	0	0	0	0
Bot	0	0	0	0	0	0	0	0	517	0	0	0
Web Attack - Brute Force	0	0	0	0	0	0	31	1	0	2	1	0
BENIGN	22	11	12	30	6	132	8994	90	35	13	9	8
PortScan	0	0	0	0	0	0	0	0	0	8946	0	0
SSH-Patator	1729	2	0	0	0	0	11	0	0	0	0	0
DDoS	0	0	8904	0	0	0	0	0	0	0	0	0
DoS Hulk	0	0	0	0	0	1	8	0	0	12	9051	0
Web Attack - Sql Injection	0	0	0	0	0	0	0	0	0	0	0	0
DoS GoldenEye	0	0	0	3126	0	0	5	2	0	0	0	0

Table C.4: Confusion matrix of the final run on CIC-IDS2017

C.5 UNSW-NB15

Prediction	True category													
	Backdoor	Reconnaissance	Exploits	Worms	Reconnaissance	Analysis	DoS	Shellcode	Fuzzers	Generic	Normal	Backdoors	Fuzzers	Shellcode
Backdoor	2	0	0	0	0	0	0	0	0	0	0	0	0	0
Reconnaissance	13	2916	32	18	0	0	79	0	33	1	0	0	0	3
Exploits	14	78	2247	9	83	32	270	18	10	146	2	15	28	13
Worms	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reconnaissance	0	0	11	0	425	0	3	9	0	2	0	0	1	0
Analysis	58	78	259	0	0	369	156	0	8	1	0	120	280	0
DoS	435	557	730	13	5	440	2954	0	344	80	0	12	39	1
Shellcode	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fuzzers	6	4	79	2	0	0	79	0	3187	10	0	0	0	31
Generic	0	1	15	0	2	0	26	15	0	3394	0	0	1	0
Normal	0	0	0	0	0	0	0	0	0	0	3505	0	0	0
Backdoors	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fuzzers	0	0	82	1	20	1	31	22	0	33	48	4	1189	0
Shellcode	6	3	11	3	0	0	32	0	0	0	0	0	0	356

Table C.5: Confusion matrix for the final run of UNSW-NB15