TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Gerlin Vainomäe 221346IAPM

BALANCING CLASSES WITH SYNTHETIC DATA FOR IOT INTRUSION DETECTION SYSTEMS

Master's Thesis

Supervisor: Zhe Deng
MSc

Co-supervisor: Ants Torim
PhD

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gerlin Vainomäe 221346IAPM

KLASSIDE TASAKAALUSTAMINE SÜNTEETILISTE ANDMETEGA IOT SISSETUNGI TUVASTAMISE SÜSTEEMIDE JAOKS

Magistritöö

Juhendaja: Zhe Deng
MSc

Kaasjuhendaja: Ants Torim
PhD

Tallinn 2025

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Gerlin Vainomäe

06.01.2025

Abstract

This study addresses a critical challenge in cybersecurity related to class imbalance in intrusion detection systems for IoT datasets. It explores how generative models, particularly CTGAN, can be used to create synthetic data that helps balance class distributions, thereby improving model performance. While the results show promise, the study identifies a key issue: the synthetic data created tends to be too similar to the original, limiting its effectiveness in enhancing model accuracy. The author suggests that future work should focus on adding noise to synthetic data to mitigate this issue. Additionally, advancing generative models and refining training techniques could help reduce data drift and ensure the relevance of synthetic data over time.

The findings contribute not only to IoT cybersecurity but also to general intrusion detection systems across various domains. This research highlights the significance of synthetic data in improving intrusion detection systems and advancing cybersecurity solutions.

The thesis is written in English and is 49 pages long, including 5 chapters, 24 figures and 23 tables.

Annotatsioon

See lõputöö käsitleb olulist küsimust küberturbes, mis on seotud klasside tasakaalustamisega rünnakutuvastussüsteemides IoT-andmestike kontekstis. See uurib, kuidas generatiivsed mudelid, täpsemalt CTGAN, saavad luua sünteetilisi andmeid, mis aitavad tasakaalustada klasside jaotusi, parandades seeläbi mudeli jõudlust ning täpsust tuvastada rünnakuid. Lõputöö käigus on avastatud ka probleeme: loodud sünteetilised andmed on tihtipeale liiga sarnased algsetele andmetele, piirates seeläbi mudeli täpsuse parandamise efektiivsust. Autor pakub, et tulevikus tuleks keskenduda müra lisamisele mudeli genereerimise või andmete töötlemise käigus, et seda probleemi leevendada. Lisaks võivad generatiivsete mudelite arendamine ja treeningmeetodite täiendamine aidata saavutada mudeli effektiivsemat toimimist.

Uuring ei puuduta mitte ainult IoT küberturvalisust, vaid ka üldises pildis rünnakutuvastusüsteeme erinevates valdkondades, kus on võimalik seda sarnastel andmestikel rakendada. Lõputöö rõhutab sünteetiliste andmete tähtsust rünnakutuvastusüsteemide parandamisel ja küberjulgeoleku lahenduste edendamisel.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 49 leheküljel, 5 peatükki, 24 joonist, 23 tabelit.

List of Abbreviations and Terms

AI	Artificial Intelligence
GAN	Generative Adversarial Network, a machine learning framework consisting of two neural networks that compete against each other to create and distinguish realistic data
IDS	Intrusion Detection System, a cybersecurity tool designed to monitor network or system activity for suspicious or malicious behaviour
IoT	Internet of Things
LLM	Large Language Model
ML	Machine Learning
PCA	Principal Component Analysis
SMOTE	Synthetic Minority Oversampling Technique, an oversampling technique used in machine learning to address class imbalance by generating synthetic examples of the minority class

Table of Contents

List of Figures

List of Tables

1. Introduction

Imbalanced class distribution in training data poses a significant challenge to the effective performance of AI (Artificial Intelligence) models. This issue is particularly pronounced when dealing with IoT (Internet of Things) datasets intended for intrusion detection (classes in this context are different attack types). Specifically, the problem arises from the presence of minority classes, which lack sufficient data representation and thus the IDSs (Intrusion Detection System) often struggle to accurately predict the presence and classification of intrusions.

For instance, in the case of the Edge-IIoTset dataset [1], the minority classes, such as fingerprinting and man-in-the-middle (MITM) attacks, are severely underrepresented with only 1001 and 1214 samples, respectively, while the rest of the classes have around 10,000 samples each. Moreover, similar challenges may be encountered in other datasets like TON_IoT [2], CICIoT2023 [3], and UNSW_NB15 [4].

1.1 Importance

Cybersecurity is a highly critical concern, and neglecting it can lead to a wide range of issues, from minor inconveniences, for example, spam emails and slow network performance, to severe crises, such as identity theft and ransomware attacks. Therefore, the accuracy of IDSs is of major importance; increasing the accuracy by finding a way to create high-quality synthetic data is the aim of the thesis.

Over the past years, numerous publications have explored methods for creating synthetic data to tackle the problem of class imbalance amongst IoT datasets. However, given the multitude of possibilities, there isn't a single definitive solution yet, as there is ongoing room for improvement. Moreover, several research papers have suggested alternative solutions worth investigating [5] [6] [7] [8] [9] [10] [11]. In the context of the ever-evolving field of ML, with the continuous publication of new technologies, there remains a constant opportunity for bettering IDSs.

Class imbalance in IoT security datasets can significantly affect the performance of IDSs. For instance, if an IDS is trained on a dataset where malicious activities are underrepresented, it may fail to detect real-world attacks effectively. This can lead to severe consequences, such as undetected data breaches or unauthorized access to sensitive infor-

mation. Real-world examples include the Mirai botnet attack [12], which exploited IoT devices to launch large-scale DDoS attacks, taking down a lot of websites and services, and the Stuxnet worm [13], which targeted industrial control systems, more precisely Iran's nuclear program. Addressing class imbalance can help in creating more robust IDSs that can detect such attacks more accurately, thereby mitigating potential threats.

The main beneficiaries of the given work are cybersecurity professionals, who rely on accurate detection of malicious activities in IoT networks, and businesses that deploy IoT devices.

1.2 Research questions

The research questions for this problem are the following:

1. What's the current most effective way to create synthetic data for IoT intrusion classifiers?
2. How to design a model that can generate high-quality synthetic IoT data?
3. How effective are GANs and LLMs in generating high-quality synthetic data for minority classes in IoT intrusion detection datasets, and how do their results compare with other data augmentation techniques like SMOTE?

1.3 Goal of the work

This thesis primarily focuses on enhancing the performance of multi-class IDSs, specifically by addressing the prevalent issue of class imbalance in IoT datasets. The main goal is to address class imbalance by introducing synthetic data into the minority classes. By doing so the bias of ML models can be reduced and their overall performance would be improved. The goal will be achieved by creating a GAN model. For the baseline some under- and oversampling methods shall be used and later on the results will also be compared to other developed GANs.

1.4 Methods

To accomplish the thesis objectives, the initial step involves a comprehensive analysis of existing IoT datasets to precisely identify minority classes and address potential data quality issues. Once the imbalance issues are thoroughly understood, various techniques will be explored to determine the optimal approach for generating synthetic data.

Based on the findings of the analysis, a model will be developed. This model will generate synthetic data, which will then be used to train classifiers. The accuracy of these classifiers will be assessed to measure the quality and utility of the synthetic data created by the model. Results will be compared against the baseline and other classifiers found in different research papers. Furthermore, additional evaluation metrics are employed to assess the quality of synthetic data by comparing it to real data. All the chosen metrics are specified under chapter 2.1.3.

1.5 Thesis structure

In this chapter, the overall structure of the thesis is outlined. The thesis is structured into different chapters, each of which plays a distinct role in the research narrative.

The first major part, background and related work, provides a solid foundation for the study by reviewing relevant background literature and existing approaches to handling class imbalance in IoT datasets, particularly focusing on GANs. This chapter also highlights related works that have previously addressed the issue of data augmentation in the context of IDSs.

The methods chapter is focused on the methodological aspects of the research. It outlines the data source, more precisely the UNSW-NB15 dataset. The techniques employed for assessing the data and synthetic data generation are presented. It contains information about the training process, evaluation metrics, and experimental configurations. Following the methodology, an overview of the experiments is conducted, relating back to the research questions.

The findings are discussed and interpreted in the discussion chapter, contrasting them with previous research outcomes. The chapter provides insights into the effectiveness of the synthetic data generated by GANs and discusses the implications of these results for the field of cybersecurity. It also analyzes the limitations encountered throughout the study and proposes possibilities for future research.

2. Background and related work

2.1 Internet of Things Intrusion Detection System

The Internet of Things (IoT) is a network of physical objects embedded with sensors and software, designed primarily to establish connections for the purpose of sharing and collecting data with other internet-connected devices and systems. This extensive range of devices spans from everyday household items, like smart thermostats, to highly advanced industrial tools. [14]

As the number of IoT devices grows, our reliance on them increases, making security a more pressing concern. To enhance the security of IoT, various solutions are being used such as antivirus software, firewalls and intrusion detection systems (IDSs) [15]. The aim of an IDS is to spot various forms of malicious network traffic and computer activities that elude detection of firewalls, thus playing an important role in offering protection against threats that put at risk the availability, integrity, or confidentiality of computer systems [16]. IDSs detect unauthorized activities, like access, replication, alteration, and destruction of information systems and these breaches encompass both external (outside the organization) and internal (within the organization) intrusions [15].

In the last decade, IDSs have been improved by using machine learning (ML) [16], but ML is not flawless as it will not always detect attacks perfectly. One specific challenge is dealing with imbalanced data, where some classes have less data compared to others.

When the training dataset contains imbalanced sample sizes for different classes, it often leads to poorer performance. Therefore, it is essential to apply data or class-balancing techniques to avoid biased results. [5]

Synthetic data is often employed to address the problem of minority classes. This approach is adopted because acquiring real data can be costly, time-intensive, or simply because the data is unavailable. At a conceptual level, synthetic data differs from real data in that it is generated based on statistical properties derived from real data. The extent to which a synthetic dataset faithfully represents real data is a measure of its utility. This process of creating synthetic data is known as synthesis and can be categorized into three types: the first type is generated using actual real datasets, the second type is independent of real data, and the third type combines elements of both approaches. [17]

2.1.1 Evaluation

Evaluation will involve comparing the newly created classifiers with those previously trained on the analysed datasets, requiring the selection of appropriate metrics. The current chosen metrics, based on existing solutions [5] [6] [7] [8] [9] [10] [11], are the following:

1. Accuracy: The number of correct predictions, including true positive and true negative predictions, divided by the total number of predictions.
2. Precision: The number of positive predictions divided by the total number of positive class values predicted.
3. Recall: The number of positive predictions divided by the number of positive class values in the test data.
4. F1-score: A weighted average of precision and recall.

For multi-class classification, the following metrics should additionally be considered [15]:

1. Overall accuracy: The percentage of exemplars correctly classified across all classes.
2. Class detection rate: The proportion of exemplars from a given class that are correctly classified out of all exemplars in that class.
3. Class FAR (False Alarm Rate) or class FP (False Positive) rate: The proportion of exemplars from a given class that are incorrectly classified out of all exemplars not from that class.

To better assess the quality of the created synthetic data, comparing it to real data is necessary. The following metrics are suggested for this evaluation:

1. Histogram similarity: Measures the resemblance of the marginal distributions of individual features between the original and synthetic data. [14]
2. Hellinger distance: The difference in univariate distribution between each variable in the real and synthetic data. [15]
3. Mutual information: The mutual dependence between features, revealing how much information can be obtained from one feature by observing another. [14]
4. Pairwise correlation difference: The similarity between real and synthetic data in terms of linear correlations across variables. Alternatively, pairwise correlation plots (e.g., heat maps) could be used for measuring. [18]
5. Propensity score: The distinguishability between real and synthetic data, indicating how difficult it is to tell them apart. [15]

2.2 Generative AI

Methodologies like under-sampling, over-sampling, Synthetic Minority Oversampling Technique (SMOTE) and artificial intelligence (AI) techniques are often used to make synthetic data for minority classes. For the past few years, the Generative Adversarial Network (GAN) has been researched and used a lot. On the other hand, Large Language Models (LLM) are also something that could be potentially used for tabular data creation, however as it is a novel concept, it hasn't been explored a lot. In this chapter, an overview of GANs will be given.

2.2.1 Generative Adversarial Network

A GAN uses an adversarial approach to create a generative model and it consists of two main parts: the generator and the discriminator. The generator is responsible for making synthetic data from random noise, and it's guided by the discriminator, which helps it learn how to mimic the patterns of minority classes. The discriminator, often referred to as the adversary network, assesses the likelihood of a data sample belonging to either the actual training data or the data generated by the generator. This assessment is continuously given to the generator until the loss either stops changing or shows minimal improvement over iterations. After the generator has been trained, it can generate data that closely matches the distribution of the training data, enabling the creation of data points not present in the original set, which aids in the detection of previously unseen attacks. [6]

Generative Adversarial Network for Intrusion Detection Systems

There are quite a lot of different GANs. In this work, the focus will be on tabular data. Even when drawing the boundary at GANs for tabular data, there are many different types and there are plenty of different research papers written about those GANs. Hence to limit the scope, in this chapter the focus will be on the most recent and best-performing GAN models meant for tabular data and centralized systems.

The names of some of the most known tabular GANs that shall be introduced are the following:

- AC-GAN Auxiliary Classifier GAN
- CGAN Conditional GAN
- CTGAN Conditional Tabular GAN
- DCGAN Deep Convolution GAN
- WCGAN Wasserstein Conditional GAN

- WCGAN-GP Wasserstein Conditional GAN with Gradient Penalty
- WGAN Wasserstein GAN
- WGAN-GP Wasserstein GAN with Gradient Penalty

Mostly used datasets in the below works are NSL-KDD, UNSW-NB15, CICIDS2017 and BoT-IoT. Most used classifiers are XGBoost, Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM).

AC-GAN stands for Auxiliary Classifier Generative Adversarial Network. It is a type of generative adversarial network (GAN) that uses an auxiliary classifier to improve the quality of the generated data. The auxiliary classifier is a separate neural network that is trained to classify the generated data into different categories.

In (Jianyu Wang et al., 2021), they propose Def-IDS, an ensemble defence mechanism specially designed for NIDS, against both known and unknown adversarial attacks. It is a two-module training framework that integrates multi-class GANs and multi-source adversarial retraining to improve model robustness. They propose a novel GAN framework to learn the data distribution of all classes of traffic in one model. The trained MGAN is able to generate new samples that could be regarded as the perturbed data but still belongs to the original classes. In order to build a multi-class model, two variant techniques, AC-GAN and SGAN, are integrated. [19])

In (Danni Yuan et al., 2020) they designed an IDS to be deployed on edge nodes. They convert network traffic to images which are applied to train a CNN to classify the categories of network traffic. Furthermore, AC-GAN is adopted to generate synthesized samples to expand the intrusion detection dataset. To balance the number of data between the minor classes and the major ones in the intrusion detection dataset, AC-GAN is used to generate synthesized network traffic samples which would be used to train the classifier. [20]

CGAN stands for Conditional GAN. This means that the generator can be given some additional information about the data that it is generating, such as the class labels or the values of certain features.

In (Hasan A. et al., 2022) they introduce a three-level intrusion detection system conditional generative adversarial network (3LIDS-CGAN) model. In third-level IDS, adversary packets are detected using CGAN which automatically learns the adversarial environment and detects adversary packets. [21]

(Ayesha S. D. et al., 2023) introduced a data generative model (DGM) to improve the

minority class presence in the anomaly detection domain. The approach is based on a CGAN to generate synthetic samples for minority classes. Kullback-Leibler divergence is used during the training stage. The quality of data generation is observed in case of ADASYN, SMOTEENN, and Borderline-SMOTE for benchmark datasets. [22]

To enhance the IMIDS's detection performance, (Kim-Hung Le et al., 2022) proposes a novel attack data generator leveraging a conditional generative adversarial network. This network is constituted of conditional generators that could learn the conditional distribution of samples in the dataset. [23]

Anomaly detection methods suffer from unbalanced and missing sample data, thus causing IDS training to be complicated. In (Hafsa Benaddi et al, 2022), they propose using CGANs to enhance the training process by handling the unbalanced data and coping with the lack of specific class samples, which may succeed in evading their Convolutional Neural Network-Long Short-Term Memory (CNNLSTM) based IDS model. CGAN is employed to learn the minority data class to balance the dataset and generate adversarial malware examples. [24]

(Imtiaz Ullah and Qusay H. Mahmoud, 2021) proposes a novel framework for detecting anomalies in IoT networks utilizing CGANs to build realistic distributions for a given feature set to overcome the issue of data imbalance. [25]

CTGAN stands for Conditional Tabular GAN. This is a type of CGAN that is specifically designed to generate synthetic tabular data. CTGAN can handle a variety of different data types compared to CGAN, including numerical, categorical, and mixed data. It is better able to capture the complex relationships between different features in the data compared to CGAN.

Two critical problems are the limits of classical oversampling methods while generating samples and the limits of understanding complex datasets and modelling real tabular data by the existing GAN models. The aim of (Omar H. et al., 2023) is to implement the CTGAN model, the state-of-the-art of GAN model in tabular data modelling and generation, to overcome all previously mentioned limits. The results of this paper show the highly quality of synthetic samples that can be generated using CTGAN. [26]

(Basim A. A. et al., 2023) proposes an IDS based on a CTGAN for detecting DDoS and DoS attacks on IoT networks. The CGAN-based IDS utilizes a generator network to produce synthetic traffic that mimics legitimate traffic patterns, while the discriminator network learns to differentiate between legitimate and malicious traffic. Addressing the

issue of extreme class imbalance in the Bot-IoT dataset through the utilization of synthetic data generation. [27]

DCGAN combines the original GANs and CNN. DCGAN is introduced in (Guosheng Zhao et al., 2023). In DCGAN discriminator and generator, a convolutional neural network (CNN) is used to replace the multilayer perceptron in GANs. The pooling layer in CNN is removed. By using batch normalization (BN) in the layer, the generator can learn stably, so that the model can learn the sample data distribution better. Both the generator and the discriminator use the BN layer, and the full convolution layer is replaced by the full connection layer. It is based on a Gated Recurrent Unit (GRU) and Residual Network (ResNet). GRU extracts the time series features of the sample data, then using ResNet and softmax functions for intrusion detection classification. [28]

The Wasserstein GAN, or **WGAN**, is a variant of a GAN which aims to improve the stability of learning and get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches. The WGAN uses weight clipping to enforce the Lipschitz constraint and a different loss function than regular GANs, which measures the Wasserstein distance between the real and generated data distributions. This loss function is more stable and less prone to exploding gradients than the traditional Jensen-Shannon divergence loss used in GANs.

In (Paulo Freitas de Araujo-Filho et al., 2023) it is talked about how while unsupervised IDSs are required to detect zero-day attacks, they usually present high false positive rates. Moreover, most existing IDSs rely on long short-term memory (LSTM) networks to consider time-dependencies among data. In the paper, they investigate GANs, an unsupervised approach to detecting attacks by implicitly modelling systems, and alternatives to LSTM networks to consider temporal dependencies among data. They have proposed an unsupervised GAN-based IDS that uses temporal convolutional networks (TCNs) and selfattention to detect cyber-attacks. The GAN is used to detect both known and zero-day attacks. [29]

(Pradeepkumar Bhale et al., 2023) proposes a transparent, optimally placed, distributed IDS solution, namely, OPTIMIST, which can handle both high-rate and low-rate DDoS attacks with good accuracy. The placement problem is formulated as the weighted minimum vertex cover problem of a K-uniform hypergraph and solved with an approximation algorithm. The IDS module is based on a long short-term memory (LSTM) model where a novel offline training method for LSTM is proposed using Wasserstein GAN-generated artificial flows. A training method is proposed for OPTIMIST where WGAN-generated artificial flows from the datasets are mixed with the original flows to reduce the biases of the datasets. [30]

(Yalong Song et al., 2023) presents an intrusion detection model, SEW-MBiGD, which integrates data processing and fusion neural networks to address data imbalance and insufficient feature learning in existing models. Firstly, to balance the dataset and mitigate the influence of edge data, the model employs SMOTE and Edited Nearest Neighbors (ENN) algorithms for data preprocessing, while also utilizing WGAN to generate minority class data. Additionally, the model employs the use of a transformer. [31]

(Cheolhee Park et al., 2023) focused on the reconstruction error and Wasserstein distance-based generative adversarial networks and autoencoder-driven deep learning models. [32]

(Shuang Zhao et al., 2021) design a new loss function to achieve an effective attack against the black-box intrusion detection system on the premise of ensuring network traffic functionality. They proposed an improved adversarial attack model based on WGAN called attackGAN. By adding the feedback of the intrusion detection system, it can effectively evade the attack, and at the same time guarantee the functionality of network traffic. [33]

In (Mustafizur R. Shahid et al., 2020), first, the autoencoder is trained to learn to convert sequences of packet sizes (sequences of categorical data) into a latent vector in a continuous space. Then a WGAN is trained on the latent space to learn to generate latent vectors that can be decoded into realistic sequences, through the decoder of the autoencoder. They also show that the synthetic bidirectional flows are close enough to the real ones that they can fool anomaly detectors into labelling them as legitimate. [34]

WCGAN (Wasserstein Conditional Generative Adversarial Network) is a type of WGAN that uses additional conditional information to guide the generation process.

The model proposed by (Mustafizur R. Shahid et al., 2020) is compared with the DGM technique which uses the CGAN model for its framework in contrast with WCGAN with gradient penalty. The proposed framework shows improved performance over that of DGM. This paper proposes an XGBoost stacked GAN approach using Auto-Encoder (AE) as a feature extraction to enhance the performance of intrusion detection with an optimal feature vector. At first, this approach implements a Deep Auto-Encoder (DAE) to derive the subset of the most significant features. After that, different GAN models are trained to synthesize the minority attack data samples. [6]

WGAN-GP is similar to WGAN, however it uses Gradient Penalty (GP). The gradient penalty is a way to enforce Lipschitz continuity of the discriminator. This means that the discriminator's gradients should be bounded in a certain range. When the discriminator's gradients are bounded, it is less likely to learn a trivial mapping from real to generated

data.

(Shiming Li et al., 2023) proposes the evaluation of an IDS with function discarding adversarial attacks in the IIoT (EIFDAA), a framework that can evaluate the defence performance of machine learning-based IDSs against various adversarial attack algorithms. This framework is composed of two main processes: adversarial evaluation and adversarial training. Adversarial evaluation can diagnose IDS that is unfitting in adversarial environments. Then, adversarial training is used to treat the weak IDS. In this framework, five well-known adversarial attacks, the fast-gradient sign method (FGSM), basic iterative method (BIM), projected gradient descent (PGD), DeepFool and WGAN-GP are used to convert attack samples into adversarial samples to simulate the adversarial environment. The WGAN was employed as a generation model for creating adversarial samples to attack a target IDSprotecting SDN-enabled models. [35]

WCGAN-GP is essentially a mixture of the previously mentioned WGAN-GP and WCGAN. The suggested model in the paper (Arpita S., 2023) also has a GA (Genetic Algorithm) attached to it, which is used for dimensionality reduction. It applies GA for selecting optimal feature subsets by filtering out the irrelevant features for efficient and accurate attack classification. [36]

3. Methods

3.1 Work environment

In this section, the work environment is detailed, encompassing the software environment and development tools.

3.1.1 Software environment

The implementation was carried out using Python version 3.10. Python was chosen due to its versatility, extensive library ecosystem, and overall widespread usage in data science and machine learning tasks. The main packages and frameworks used include:

- **scikit-learn [37]:** A machine learning library for implementing feature selection, model training, and evaluation. It provides tools for handling data preprocessing and performance metrics.
- **TensorFlow [38]:** A deep learning framework which is used to design and train the model for synthetic data generation.
- **Pandas [39]:** A library for data manipulation and analysis, essential for preprocessing the dataset.
- **NumPy [40]:** A package for numerical computing, used for mathematical operations.
- **Matplotlib [41] and Seaborn [42]:** Visualization libraries to create plots for data analysis and results presentation.
- **CTGAN [43]:** A library specifically designed for tabular GAN-based synthetic data generation.

3.1.2 Development tools

The development process involved two primary tools:

- **Jupyter Notebook:** Commonly used for machine learning tasks, it was utilized for data analysis, synthetic data generation, and result visualization.
- **Visual Studio Code:** A versatile code editor used for managing code and debugging, was chosen due to familiarity with the tool.

3.2 Dataset

The main focus is on using the UNSW-NB15 dataset, which is a network security dataset that is commonly used for research and evaluation of IDSs. It was created by the Network Security Lab at the University of New South Wales (UNSW) in Australia in 2015. The dataset is designed to simulate a real-world network environment and contains a variety of normal and malicious network traffic. The dataset includes a diverse set of network traffic, including normal traffic and nine types of attack families. The exact attack types and number of records can be seen in table 1, where the dataset imbalance can also be noticed. The dataset provides detailed information at the packet level. The training set contains 175,341 records and the testing set has 82,332 records. [4] (Note: originally, the dataset has more samples, namely 2,540,044 records, however on the UNSW-NB15 website, a partition of this dataset is available, which is also generally used in other works.)

Table 1. UNSW-NB15 dataset record distribution

Type	Training subset	Testing subset	Total
Normal	56,000	37,000	93,000
Generic	40,000	18,871	58,871
Exploits	33,393	11,132	44,525
Fuzzers	18,184	6,062	24,246
DoS	12,264	4,089	16,353
Reconnaissance	10,491	3,496	13,987
Analysis	2,000	677	2,677
Backdoors	1,746	583	2,329
Shellcode	1,133	378	1,511
Worms	130	44	174

3.2.1 Machine Learning Algorithms

The algorithms selected for evaluation in this study are Random Forest and XGBoost (eXtreme Gradient Boosting). These models have gained widespread popularity due to their proven performance and robustness in various machine learning tasks, particularly in classification and regression problems. Choosing widely-used models like Random Forest and XGBoost, allows for straightforward comparisons with results from other research papers.

3.3 Feature engineering

Feature engineering plays a crucial role in preparing data for machine learning models, particularly when dealing with complex datasets like UNSW-NB15. A comprehensive feature engineering has been employed to a pipeline to preprocess and transform the UNSW-NB15 dataset, ensuring that the features are suitable for training effective intrusion detection models.

To establish a baseline for comparison, the results of multiclass classification using the Random Forest and XGBoost classifiers before applying any feature engineering are presented in Table 2.

Table 2. Multiclass classification before feature engineering

Method	**Accuracy**	**Precision**	**Recall**	**F1-score (micro)**	**F1-score (macro)**
Random Forest	0.755	0.84	0.76	0.78	0.47
XGBoost	0.712	0.82	0.71	0.74	0.49

The first step in the process was label encoding, which involved converting categorical features into numerical values. The original dataset consists of a total of 45 different columns, from which four of them are categorical and the rest numerical. The categorical features are proto, service, state and attack_cat.

Next highly correlated features are removed because multicollinearity can otherwise increase computation time and lead to overfitting models. Features are considered highly correlated when their value is higher than 0.98. Keeping only one feature from a correlated pair simplifies the model without significant loss of information. As a result, features like sloss, dloss, dwin, ct_srv_dst and ct_ftp_cmd are dropped due to how closely they are correlated.

Table 3. Highest feature correlations

Feature 1	**Feature 2**	**Correlation coefficient**
sbytes	sloss	0.9961094729148002
dbytes	dloss	0.9965035947623478
swin	dwin	0.9901399299415929
ct_srv_src	ct_srv_dst	0.9803230099911133
is_ftp_login	ct_ftp_cmd	1.0

Table 3 provides a visual representation of the correlation between various features. Lighter

colours on the figure, indicating values closer to 1.0, signify a stronger similarity between the features, while the darker colours, indicating values closer to 0.6, signify a lower similarity.

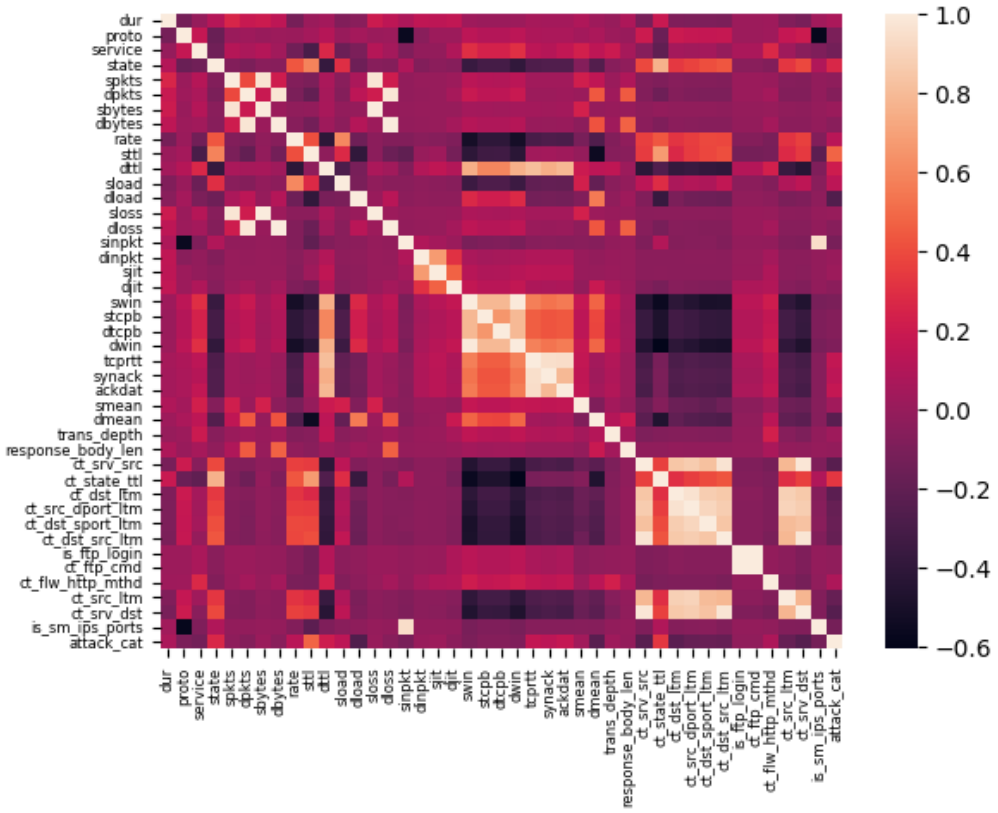

Figure 1. Feature correlations

To further refine the feature set, an Extra Trees Classifier (sklearn.ensemble) was trained on the training data and extracted feature importance scores. Features with importance scores lower than 0.01 were considered less relevant and were subsequently removed from the dataset. This step aimed to reduce the dimensionality of the feature space and focus on the most informative features for the intrusion detection task.

Table 4. Extra Trees Importance

Feature	Importance Score
sttl	0.111586
dttl	0.074821
ct_src_dport_ltm	0.067877
ct_srv_src	0.066188
service	0.060049
ct_dst_src_ltm	0.057570
ct_state_ttl	0.049908
smean	0.049526
ct_dst_ltm	0.047443
sbytes	0.033832
swin	0.032253
ct_dst_sport_ltm	0.031403
ct_src_ltm	0.026956
dmean	0.026436
dload	0.021061
proto	0.020687
sload	0.020438
rate	0.020216
stcpb	0.018474
state	0.016808
synack	0.013630
dur	0.012917
sinpkt	0.012859
dtcpb	0.011868
tcprtt	0.011505
sjit	0.010460
dbytes	0.010236
ackdat	0.009965
dpkts	0.009175
spkts	0.008444
djit	0.008185
dinpkt	0.008175
is_sm_ips_ports	0.006912
trans_depth	0.005125
ct_flw_http_mthd	0.004079
response_body_len	0.002356
is_ftp_login	0.000578

After dimensionality reduction through feature importance scores, different methods of feature reduction were tested out, an additional feature selection step was implemented using the SelectKBest method from scikit-learn. This method selects the top 20 features based on their Analysis of Variance (ANOVA) F-value scores, which measure the degree

of separation between classes for each feature. By retaining only the top-ranked features, the feature set was further optimized for improved model performance and interpretability. [37] Additionally, Boruta and Fisher score were experimented with, which later showed to yield better results compared to SelectKBest, though Boruta and Fisher score seemed to perform similarly as the methods chose the same features. Since various ways to incorporate Boruta were tested, there is Boruta V1 and V2, with one version having more features and the other less.

Table 5. Chosen features

Feature	**SelectKBest**	**Boruta V1**	**Boruta V2**	**Fisher score**
sttl	X	X	X	X
ct_dst_src_ltm	X	X	X	X
dttl	X	X	X	X
ct_dst_sport_ltm	X	X	X	X
ct_srv_src	X	X	X	X
service	X	X	X	X
smean		X	X	X
ct_state_ttl	X	X	X	X
ct_src_dport_ltm	X	X	X	X
sbytes		X	X	X
ct_dst_ltm	X	X	X	X
ct_src_ltm	X	X	X	X
swin	X		X	X
dmean	X	X	X	X
state	X		X	X
proto		X	X	X
sload		X	X	X
rate	X	X	X	X
dtcpb	X		X	X
stcpb	X		X	X
tcprtt	X	X	X	X
sinpkt		X	X	X
dload	X	X	X	X
dur		X	X	X
synack	X	X	X	X
sjit		X	X	X
dbytes		X	X	X
ackdat	X		X	X

For categorical features (such as proto, service, and state), one-hot encoding was applied. This technique converts categorical variables into binary columns, representing the presence or absence of each category.

Next, the feature variables were standardized by ensuring that they have a mean of 0 and a standard deviation of 1. Standardization is an important step, as it helps to prevent features with larger scales from dominating those with smaller scales.

Finally, Principal Component Analysis (PCA) was tried out to further reduce the dimensionality of the feature space. PCA is a widely used technique for dimensionality reduction that transforms the original features into a new set of uncorrelated variables called principal components.

By applying this feature engineering pipeline, the aim was to enhance the quality and suitability of the UNSW-NB15 dataset for training intrusion detection models. The combination of feature encoding, dimensionality reduction, feature selection, and standardization techniques enabled the extraction of the most relevant and informative features while mitigating potential issues such as multicollinearity and scale differences.

3.4 Experiments and results

This paragraph describes the experiments conducted to evaluate the effectiveness of various techniques and methods for improving intrusion detection on the UNSW-NB15 dataset. The experiments aim to assess the impact of feature selection, dimensionality reduction, data balancing strategies, and synthetic data generation on model performance.

3.4.1 Experiments with PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique. It is a statistical procedure that transforms a high-dimensional dataset into a lower-dimensional space while retaining the maximum possible variance from the original data. PCA can be beneficial when working with the UNSW-NB15 dataset, as it can reduce computational complexity and potentially improve the performance of machine learning models by eliminating redundant or irrelevant features. [44]

Additionally, truncated Singular Value Decomposition (SVD) was evaluated. Unlike PCA, SVD does not centre the data before performing decomposition, making it well-suited for working with sparse matrices. This enables efficient dimensionality reduction in specific scenarios, especially when the dataset's structure contains many zero values. [37]

PCA and SVD were applied in conjunction with feature engineering methods such as SelectKBest and one-hot encoding. The experiments were conducted using different

numbers of components, notably PCA with 3 and 6, as well as SVD with 3, 6, and 10 components. Tables 6 and 7 summarize the results across these scenarios, showing the key metrics of accuracy, precision, recall, and F1-score.

Table 6. Feature engineering PCA tests with Random Forest Classifier

Method	**Accuracy**	**Precision**	**Recall**	**F1-score**
None	0.76	0.84	0.76	0.78
SelectKBest	0.72	0.81	0.72	0.74
SelectKBest + PCA(3)	0.61	0.66	0.61	0.62
SelectKBest + PCA(6)	0.52	0.71	0.53	0.55
SelectKBest + SVD(3)	0.56	0.66	0.56	0.59
SelectKBest + SVD(6)	0.53	0.70	0.53	0.55
One-Hot Encoding	0.76	0.81	0.77	0.77
One-Hot Encoding + PCA(3)	0.32	0.39	0.32	0.28
One-Hot Encoding + PCA(6)	0.35	0.46	0.35	0.28
One-Hot Encoding + SVD(3)	0.40	0.51	0.40	0.41
One-Hot Encoding + SVD(6)	0.32	0.34	0.32	0.26
One-Hot Encoding + SVD(10)	0.30	0.44	0.30	0.23

Table 7. Feature engineering PCA tests with Random Forest Classifier, binary classification

Method	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Time (s)**
Normal	1.0	1.0	1.0	1.0	18.61
Normal, PCA(3)	0.78	0.80	0.78	0.77	35.14
Generic	0.99	0.99	0.99	0.99	22.14
Generic, PCA(3)	0.98	0.98	0.98	0.98	26.38
Exploits	0.92	0.92	0.92	0.92	30.91
Exploits, PCA(3)	0.88	0.86	0.88	0.87	27.78
Fuzzers	0.88	0.90	0.88	0.89	29.03
Fuzzers, PCA(3)	0.87	0.89	0.87	0.88	28.50
DoS	0.95	0.94	0.95	0.93	28.94
DoS, PCA(3)	0.95	0.91	0.95	0.93	29.71
Reconnaissance	0.99	0.99	0.99	0.99	19.81
Reconnaissance, PCA(3)	0.95	0.93	0.95	0.94	26.22
Analysis	0.99	0.98	0.99	0.99	34.09
Analysis, PCA(3)	0.99	0.98	0.99	0.99	21.57
Backdoor	0.97	0.99	0.97	0.98	21.79
Backdoor, PCA(3)	0.99	0.99	0.99	0.99	26.55
Shellcode	1.0	1.0	1.0	1.0	20.81
Shellcode, PCA(3)	0.99	0.99	0.99	0.99	22.87
Worms	1.0	1.0	1.0	1.0	17.54
Worms, PCA(3)	1.0	1.0	1.0	1.0	21.54

As evident from the results, the application of PCA in conjunction with SelectKBest and one-hot encoding yielded mixed outcomes. SVD in this case didn't produce very different results either. Not using any of the methods provided better results across all metrics. However, this approach would significantly increase training time due to the high dimensionality of the UNSW-NB15 dataset, and it can lead to overfitting. Therefore, despite the current contradictory results, it remains necessary to select one of these methods to balance performance and computational efficiency.

While SelectKBest alone achieved an accuracy of 0.72 and an F1-score of 0.74, incorporating PCA with SelectKBest led to a decrease in performance, with an accuracy of 0.61 and an F1-score of 0.62. Increasing the number of PCA components to 6 further reduced performance, with accuracy and F1-scores of 0.52 and 0.55, respectively.

Similarly, one-hot encoding alone performed reasonably well, with an accuracy of 0.76 and an F1-score of 0.77, but the combination of one-hot encoding and PCA(3) resulted in significant performance degradation, with an accuracy of 0.32 and an F1-score of 0.28, with PCA(6) yielding similar outcomes.

These findings suggest that while PCA can be an effective dimensionality reduction technique in certain scenarios, its application to the UNSW-NB15 dataset may not always be beneficial, particularly when combined with specific feature engineering methods like one-hot encoding. The high-dimensional nature of one-hot encoded features posed significant challenges, leading to a preference for feature engineering approaches without PCA in many scenarios. However, the use of PCA can still play a critical role in reducing training time under specific circumstances. Further fine-tuning of the dimensionality reduction parameters and preprocessing steps may yield more favourable results.

3.4.2 Experiments with synthetic data

In this section, the impact of synthetic data generated using the CTGAN is evaluated on the performance of a machine learning-based intrusion detection model. This analysis aims to understand how various configurations of synthetic data - generated with different sample sizes and with or without dimensionality reduction techniques - affect model performance across different dataset balancing strategies. The evaluation primarily focuses on assessing the effectiveness of these methods in terms of accuracy, precision, recall, and F1-score. Additionally, the impact on training time (computational efficiency) is examined, given the potentially high computational costs associated with synthetic data generation and model training.

To further enhance the analysis, different feature selection and dimensionality reduction techniques were employed to observe their influence on model performance. When using PCA with the Random Forest classifier, the training time increased notably compared to scenarios without PCA. However, in the CTGAN training phase, incorporating PCA preprocessed data led to a substantial reduction in computational time—reducing the training duration to under an hour, compared to approximately three to four hours for data without PCA preprocessing. Despite these computational gains, the results with PCA did not meet the expected performance levels, suggesting that PCA might not capture the most relevant features for intrusion detection in this context.

Further optimization using the Boruta algorithm demonstrated improved results over both PCA and SelectKBest, showing higher accuracy and stability across synthetic datasets. Given these outcomes, Boruta was selected as the preferred feature selection method for subsequent training phases, as it demonstrated the most promising balance of computational efficiency and performance.

It was decided to generate synthetic data with a size that is a fraction of the original dataset size. For the UNSW-NB15 dataset, which contains 175,341 rows, synthetic datasets of the following sizes were generated:

1. 10% of 175,341 rows = 17,534 synthetic rows
2. 20% of 175,341 rows = 35,068 synthetic rows
3. 30% of 175,341 rows = 52,602 synthetic rows

This systematic variation in synthetic dataset sizes allows us to assess how dataset size influences model performance and to evaluate the balance between computational efficiency and classification effectiveness.

3.4.3 Experiments with SMOTE, undersampling, oversampling

Tables 8 and 9 present a comparison of traditional balancing techniques, including SMOTE, oversampling, and undersampling, evaluated using the SelectKBest and Boruta feature selection methods. These techniques were assessed in terms of accuracy, precision, recall, F1-score, and computational time.

When using SelectKBest, as shown in Table 8, all balancing methods performed similarly across most metrics. However, the TOMEK undersampling technique turned out to be the most balanced and efficient option. It achieved the highest values for accuracy, precision, recall, and F1-score without compromising performance. Additionally, TOMEK

undersampling demonstrated significantly reduced computational time compared to other techniques, completing in 25.63 seconds, which is substantially faster than oversampling methods. This combination of stable performance and reduced processing time shows that TOMEK undersampling is a highly effective technique when computational efficiency is a priority.

Table 8. SMOTE testing (with SelectKBest)

Method	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Time (s)**
Oversampling	0.75	0.83	0.74	0.77	55.72
Oversampling (ADASYN)	0.75	0.83	0.74	0.77	54.71
Undersampling	0.62	0.83	0.62	0.68	187.92
Undersampling (ENN)	0.75	0.81	0.75	0.76	11.16
Undersampling (TOMEK)	0.75	0.83	0.75	0.77	25.63

When Boruta was employed for feature selection, the overall results improved compared to those with SelectKBest. The results are brought out in Table 9. In this case, ENN undersampling outperformed other techniques, achieving the best results in accuracy, precision, recall, and F1 score. ENN also maintained an efficient runtime of only 7.49 seconds, further emphasizing its effectiveness for balancing while reducing computational overhead.

Table 9. SMOTE testing (with Boruta V1)

Method	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Time (s)**
Oversampling	0.75	0.84	0.76	0.78	45.60
Oversampling (ADASYN)	0.75	0.84	0.76	0.78	54.71
Undersampling	0.64	0.84	0.64	0.70	0.25
Undersampling (ENN)	0.77	0.81	0.77	0.78	7.49
Undersampling (TOMEK)	0.76	0.84	0.76	0.78	29.56

The findings bring out the advantages of undersampling methods, particularly TOMEK and ENN, in scenarios where training time is a consideration without sacrificing model performance. TOMEK was most effective with SelectKBest, while ENN demonstrated best performance with Boruta. These results suggest that the choice of balancing technique and feature selection method should be aligned with the specific goals, whether prioritizing model performance, computational efficiency, or both.

3.4.4 Experiments with neural network

For further analysis, neural network architectures were also tested, including Multilayer Perceptron (MLP), Dense Neural Networks, GRU, LSTM, and a Bi-GRU CNN model using Boruta feature selection. As shown in Table 10, the Bi-LSTM CNN model with Boruta selection achieved the highest performance, reaching an accuracy of 79%, a recall of 79%, and an F1-score of 78%, outperforming all other methods.

Interestingly, while the GRU and LSTM networks performed moderately, adding synthetic data did not drastically improve their results, which suggests that certain architectures may benefit differently from synthetic data augmentation. (Requires data to be added to the tables)

Table 10. Results with Neural Network, Random Forest

Method	Accuracy	Precision	Recall	F1-score
MLP NN (SelectKBest)	0.732	0.81	0.73	0.74
Dense NN (SelectKBest)	0.714	0.78	0.71	0.72
GRU (SelectKBest)	0.607	0.65	0.60	0.61
LSTM (SelectKBest)	0.674	0.70	0.67	0.68
Bi-GRU CNN (Boruta)	0.757	0.81	0.76	0.76
Bi-LSTM CNN (Boruta)	0.790	0.81	0.79	0.78

3.4.5 Experiments with CTGAN

Categories trained together

The evaluation results suggest that synthetic data generated by CTGAN, particularly in combination with Boruta feature selection, can improve model performance when used to balance an imbalanced dataset. The primary advantage lies in enhancing accuracy and F1-score without relying exclusively on oversampling techniques like SMOTE, which can sometimes lead to overfitting.

However, a limitation of using synthetic data alone (without real data) is the potential for diminished performance. This shows the importance of incorporating both real and synthetic data to maintain predictive accuracy. Additionally, while increasing synthetic data size beyond 10% did not produce better results, optimizing GAN parameters such as embedding dimensions and batch sizes had a noticeable impact on training efficiency and outcome consistency.

In Table 11, CTGAN, using the SDV library, was trained with SelectKBest and PCA, running for 100 epochs.

Table 11. Results with CTGAN, SelectKBest and PCA, Random Forest

Method	Accuracy	Precision	Recall	F1-score
SelectKBest	0.72	0.81	0.72	0.74
SelectKBest, synthetic data only	0.65	0.77	0.65	0.67
SelectKBest, synthetic data 10%	0.70	0.81	0.70	0.73
SelectKBest, synthetic data 20%	0.71	0.81	0.71	0.73
SelectKBest, synthetic data 30%	0.70	0.80	0.70	0.72
SelectKBest + PCA	0.61	0.66	0.61	0.62
SelectKBest + PCA, synthetic data only	0.58	0.67	0.58	0.60
SelectKBest + PCA, synthetic data 10%	0.60	0.66	0.60	0.61
SelectKBest + PCA, synthetic data 20%	0.60	0.66	0.60	0.61
SelectKBest + PCA, synthetic data 30%	0.61	0.66	0.61	0.62

The performance of CTGAN was evaluated using several configurations, varying parameters such as batch size, number of epochs, embedding dimensions, and generator/discriminator settings. Table 12 summarizes the key configurations tested.

Table 12. CTGAN Training Configurations

Version	Training Time	Batch Size	Epochs	Embedding Dim	Additional Details
Version 1	90m49.7s	500	50	128	Sample size = 10,000
Version 2	18m27.1s	500	10	512	Sample size = 20,000
Version 3	25m15.2s	500	10	512	Increasing generator and discriminator dimensions from (256, 256) to (512, 256, 128)
Version 4	156m44.8s	1,000	100	512	Generator LR = 1e-4, PAC = 10
Version 5	593m54.6s	2,000	150	512	GPU acceleration, PAC = 10

Each configuration was tailored to test specific aspects of the CTGAN model:

- **Version 1:** Baseline configuration with minimal embedding dimensions and a smaller sample size.
- **Version 2:** Reduced training epochs to optimize runtime while increasing embedding dimensions.
- **Version 3:** Modified generator and discriminator dimensions for improved representation.
- **Version 4:** Larger batch size and more advanced hyperparameters to enhance generator stability.
- **Version 5:** High computational power using GPU acceleration and extensive training epochs.

Upon evaluating the synthetic data generated by the CTGAN model, no significant data drift was observed, indicating reliable alignment with the original dataset. The best results in terms of accuracy were achieved by ver3, as shown in Table 13 which trained CTGAN for 10 epochs with optimized generator and discriminator dimensions (increasing from (256, 256) to (512, 256, 128)). While this version yielded high accuracy when combining real and synthetic data, the performance of the synthetic data alone was comparatively lower than other versions.

Increasing the training epochs beyond 10 did not lead to noticeable improvements in combined performance with real data. However, training for additional epochs did improve evaluation metrics when assessing synthetic data independently. Even so, synthetic data alone could not outperform the original data, with real data achieving an accuracy of 75.5% and the highest synthetic-only accuracy reaching 64.1%. This indicates that synthetic data, while useful in combination with real data, remains limited in its standalone performance.

Table 13. Results with CTGAN, Boruta V1, Random Forest

Method	**Accuracy**	**Precision**	**Recall**	**F1-score**
Boruta	0.755	0.83	0.76	0.78
Version 1				
Boruta, synthetic data only	0.593	0.61	0.59	0.58
Boruta, synthetic data 10%	0.760	0.83	0.76	0.78
Boruta, synthetic data 20%	0.758	0.83	0.76	0.78
Boruta, synthetic data 30%	0.755	0.83	0.76	0.78
Version 2				
Boruta, synthetic data only	0.557	0.56	0.56	0.54
Boruta, synthetic data 10%	0.758	0.83	0.76	0.78
Boruta, synthetic data 20%	0.759	0.83	0.76	0.78
Boruta, synthetic data 30%	0.759	0.83	0.76	0.78
Version 3				
Boruta, synthetic data only	0.428	0.33	0.43	0.32
Boruta, synthetic data 10%	0.759	0.83	0.76	0.78
Boruta, synthetic data 20%	0.759	0.83	0.76	0.78
Boruta, synthetic data 30%	0.761	0.84	0.76	0.78
Version 4				
Boruta, synthetic data only	0.641	0.80	0.64	0.66
Boruta, synthetic data 10%	0.756	0.84	0.76	0.78
Boruta, synthetic data 20%	0.755	0.84	0.75	0.78
Boruta, synthetic data 30%	0.755	0.84	0.76	0.78
Version 5				
Boruta, synthetic data only	0.636	0.75	0.64	0.65
Boruta, synthetic data 10%	0.758	0.84	0.76	0.78
Boruta, synthetic data 20%	0.755	0.84	0.75	0.78
Boruta, synthetic data 30%	0.755	0.84	0.75	0.78

Testing Version 5 with XGBoost revealed that incorporating synthetic data significantly boosts model accuracy. While the original data alone yields a lower accuracy in XGBoost than in Random Forest, adding 10% synthetic data rows to the original dataset produced the best results so far, achieving an accuracy of 77.7%. Additionally, the macro F1 score is measured to see how the model performs in all classes. This is useful, particularly in imbalanced datasets, where minority classes should not be ignored. (This metric should be considered for other tests as well.)

Table 14. Results using CTGAN, Boruta V1, XGBoost

Method	**Accuracy**	**Precision**	**Recall**	**F1-score, micro**	**F1-score, macro**
Boruta	0.732	0.83	0.73	0.75	0.50
Version 5					
Boruta, synthetic data only	0.677	0.81	0.68	0.69	0.30
Boruta, synthetic data 10%	0.777	0.84	0.78	0.78	0.52
Boruta, synthetic data 20%	0.769	0.83	0.77	0.78	0.52
Boruta, synthetic data 30%	0.770	0.83	0.77	0.78	0.52

Additionally, Fisher score was briefly tested; however, it produced results that were similar to those of Boruta, leading to it not being investigated in this study any further.

Table 15. Results using CTGAN, Fisher score, Random Forest

Method	**Accuracy**	**Precision**	**Recall**	**F1-score, micro**	**F1-score, macro**
Only original data	0.651	0.76	0.65	0.67	0.27
Only synthetic data	0.677	0.81	0.68	0.69	0.30
Original data + synthetic data 10%	0.755	0.84	0.75	0.78	0.48
Original data + synthetic data 20%	0.755	0.84	0.76	0.78	0.47
Original data + synthetic data 30%	0.755	0.84	0.75	0.78	0.47

Table 16. Results using CTGAN, Fisher score, XGBoost

Method	**Accuracy**	**Precision**	**Recall**	**F1-score, micro**	**F1-score, macro**
Only original data	0.732	0.83	0.73	0.75	0.50
Only synthetic data	0.672	0.79	0.67	0.68	0.29
Original data + synthetic data 10%	0.744	0.83	0.74	0.76	0.51
Original data + synthetic data 20%	0.783	0.83	0.78	0.79	0.50
Original data + synthetic data 30%	0.754	0.82	0.75	0.77	0.49

Analyzing Version 5 The density plot, shown in Figure 2, visualizes the distribution of values for each feature in both the real and synthetic datasets. The overall high overlap in the distributions suggests the two sets are very similar to each other. However, noticeable differences can be observed in cases where the real data has a very low density. For instance,

features such as sbytes, sjit, and dbytes show distinct discrepancies, with the synthetic data showing higher density in these areas. This could indicate that there are certain features where the synthetic data might not fully capture the lower-density behaviour of the real dataset.

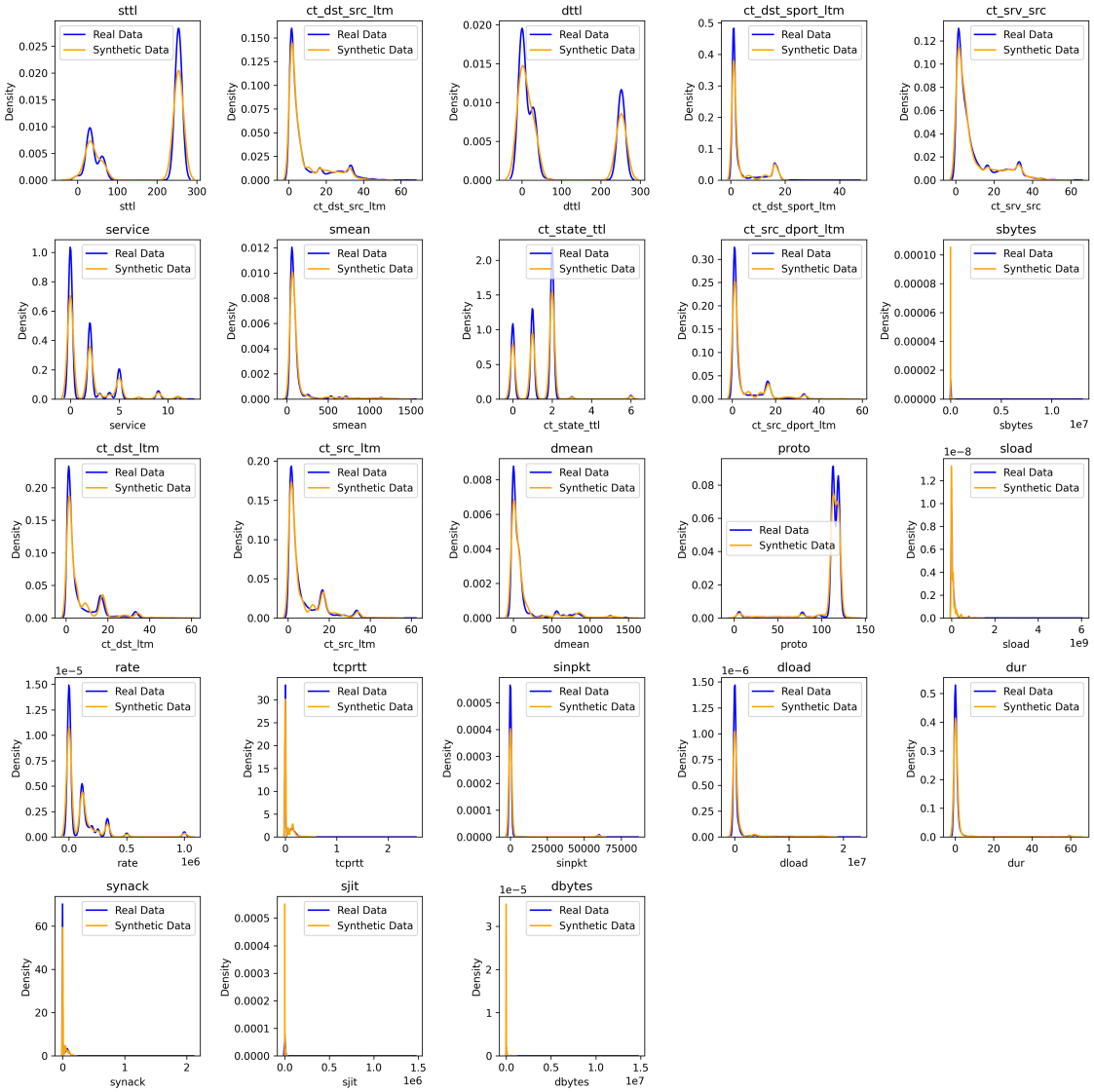

Figure 2. Version 5 density plot

As illustrated in Figure 3, no data drift is detected when comparing the real data (after feature engineering) with the synthetic data (which contains 20% of the real dataset's sample size). The distributions across the columns are largely similar. The highest drift score is 0.070683 for the sjit column, while the lowest is 0.002727 for the cs_state_ttl column. Since the drift threshold is set at 0.5, there is no significant drift between the two sets.

Figure 3. Version 5 data drift

The confusion matrix in Figure 4 shows that while the model performs well in identifying classes with a higher volume of data, it struggles with classes that are underrepresented. Specifically, a significant number of samples from these minority classes are misclassified as Exploits. This issue could be addressed by generating more targeted synthetic data for the underperforming classes, namely DoS, Analysis, and Backdoor, to better balance the dataset and improve model performance for these categories.

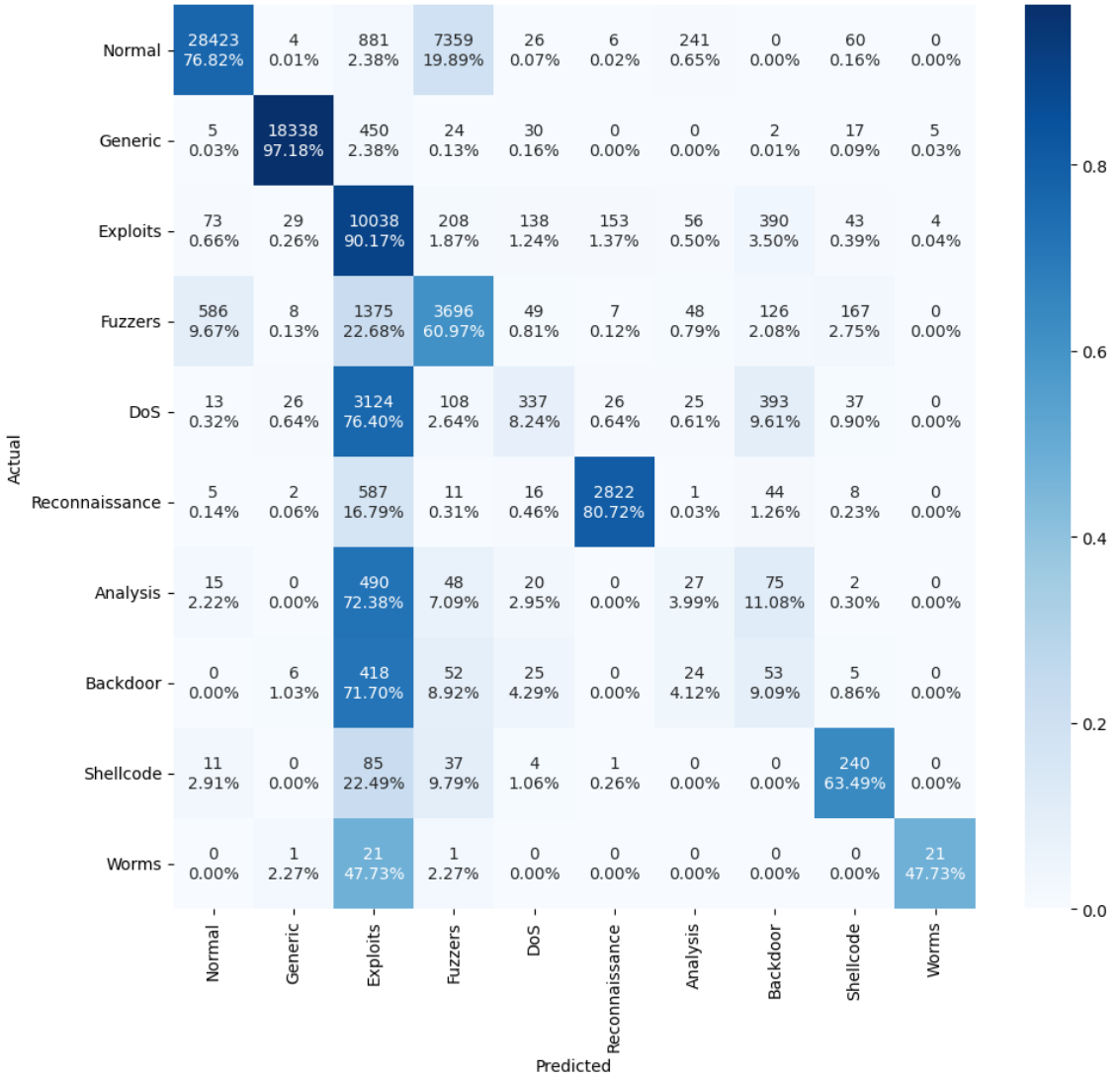

Figure 4. Version 5, 10% synthetic data, confusion matrix

- Mutual information - near 0: The features are very different, with little shared information. Higher: Features are more similar and share more information.
- Hellinger distance - 0 indicates that the two distributions are identical, and 1 indicates that they are completely dissimilar.

Table 17. Mutual Information and Hellinger Distance for each feature

Feature	Mutual Information	Hellinger Distance
sttl	0.4672	0.0042
ct_dst_src_ltm	0.4831	0.1966
dttl	0.4887	0.0499
ct_dst_sport_ltm	0.5195	0.3912
ct_srv_src	0.4785	0.1568
service	0.4831	0.4173
smean	0.9304	0.0630
ct_state_ttl	0.4893	0.0054
ct_src_dport_ltm	0.4707	0.1684
sbytes	1.1510	0.1808
ct_dst_ltm	0.4568	0.1633
ct_src_ltm	0.3889	0.2187
dmean	0.5497	0.0926
proto	0.4748	0.6083
sload	0.9053	0.3004
rate	0.5479	0.1861
tcprtt	0.3946	0.4023
sinpkt	0.4387	0.1353
dload	0.4787	0.1626
dur	0.5358	0.0635
synack	0.3964	0.3829
sjit	0.3927	0.9473
dbytes	0.6370	0.2506

Categories trained separately

In this section, CTGAN is trained separately on different categories, or in other words, attack types, as opposed to training on all categories combined. For clarity, all categories are assigned numbers, as detailed in Table 18, which maps each category number to its corresponding attack type.

Table 18. Category numbers

Category number	Attack type
0	Normal
1	Generic
2	Exploits
3	Fuzzers
4	DoS
5	Reconnaissance
6	Analysis
7	Backdoor
8	Shellcode
9	Worms

One approach is to ensure that at least half the volume of the most frequently sampled class—in this case, 'Normal' — is added to the dataset. The approach can be seen in Table 19.

Table 19. Chosen class sizes Version 1

Class	Current Size	Target Size	Deficit (Target - Current)
Normal	**56000**	**28000**	**0**
Generic	**40000**	**28000**	**0**
Exploits	**33393**	**28000**	**0**
Fuzzers	**18184**	**28000**	**9816**
DoS	**12264**	**28000**	**15736**
Reconnaissance	**10491**	**28000**	**17509**
Analysis	**2000**	**28000**	**26000**
Backdoor	**1746**	**28000**	**26254**
Shellcode	**1133**	**28000**	**26867**
Worms	**130**	**28000**	**27870**

Another strategy is to add 10,000 new samples to each minority class, as illustrated in Table 20. This approach helps balance the class distribution while maintaining a reasonable increase in the dataset size.

Table 20. Chosen class sizes Version 2

Class	Current Size	Added Samples	Final Size
Normal	**56000**	**0**	**56000**
Generic	**40000**	**0**	**40000**
Exploits	**33393**	**0**	**33393**
Fuzzers	**18184**	**10000**	**28184**
DoS	**12264**	**10000**	**22264**
Reconnaissance	**10491**	**10000**	**20491**
Analysis	**2000**	**10000**	**12000**
Backdoor	**1746**	**10000**	**11746**
Shellcode	**1133**	**10000**	**11133**
Worms	**130**	**10000**	**10130**

In Table 21 it can be seen how classes perform by themselves when a different amount of samples are taken from the synthesizer. Each class has been trained with a separate CTGAN and personalized parameters based on the data amount, the exception being Exploits, Generic and Normal, as they're the majority classes, which have been sampled from a model where all the data was trained together. The specific training parameters are shown in Table 22 for each class and confusion matrices can be seen in Appendix 2.

Table 21. Testing classes one by one, Boruta V2

Class	Current Size	Accuracy	Precision	Recall	F1-score (micro)	F1-score (macro)
Worms, synthetic data only	10130	0.355	0.47	0.35	0.33	0.10
Worms, synthetic + real data	10130	0.753	0.83	0.75	0.77	0.47
Worms, synthetic data only	28000	0.372	0.51	0.37	0.35	0.10
Worms, synthetic + real data	28000	0.753	0.83	0.75	0.77	0.47
Shellcode, synthetic data only	11133	0.226	0.40	0.23	0.21	0.08
Shellcode, synthetic + real data	11133	0.759	0.84	0.76	0.78	0.49
Shellcode, synthetic data only	28000	0.263	0.47	0.26	0.25	0.11
Shellcode, synthetic + real data	28000	0.751	0.83	0.75	0.77	0.47
Backdoor, synthetic data only	11746	0.372	0.51	0.37	0.35	0.10
Backdoor, synthetic + real data	11746	0.759	0.84	0.76	0.78	0.47
Backdoor, synthetic data only	28000	0.460	0.46	0.46	0.44	0.15
Backdoor, synthetic + real data	28000	0.760	0.83	0.76	0.78	0.46
Analysis, synthetic data only	12000	0.370	0.48	0.38	0.37	0.13
Analysis, synthetic + real data	12000	0.753	0.83	0.75	0.77	0.47
Analysis, synthetic data only	28000	0.371	0.49	0.37	0.37	0.13
Analysis, synthetic + real data	28000	0.756	0.84	0.76	0.78	0.48
Reconnaissance, synthetic data only	20491	0.183	0.47	0.18	0.13	0.13
Reconnaissance, synthetic + real data	20491	0.748	0.84	0.75	0.77	0.47
Reconnaissance, synthetic data only	28000	0.219	0.49	0.22	0.20	0.14
Reconnaissance, synthetic + real data	28000	0.751	0.84	0.75	0.77	0.48
DoS, synthetic data only	22264	0.343	0.48	0.34	0.34	0.11
DoS, synthetic + real data	22264	0.753	0.84	0.75	0.77	0.48
DoS, synthetic data only	28000	0.375	0.48	0.38	0.37	0.12
DoS, synthetic + real data	28000	0.756	0.84	0.76	0.78	0.47
Fuzzers, synthetic data only	28184	0.222	0.47	0.22	0.20	0.09
Fuzzers, synthetic + real data	28184	0.754	0.84	0.75	0.78	0.48
Fuzzers, synthetic data only	28000	0.247	0.45	0.25	0.23	0.09
Fuzzers, synthetic + real data	28000	0.761	0.84	0.76	0.78	0.48
Exploits, synthetic data only	10000	0.368	0.50	0.37	0.37	0.14
Exploits, synthetic + real data	10000	0.745	0.84	0.75	0.77	0.46
Generic, synthetic data only	10000	0.653	0.63	0.65	0.59	0.22
Generic, synthetic + real data	10000	0.752	0.84	0.75	0.78	0.48
Normal, synthetic data only	10000	0.618	0.72	0.62	0.64	0.25
Normal, synthetic + real data	10000	0.758	0.83	0.76	0.78	0.48

As shown in Table 22, the training parameters for individual categories differ from those used when training the model on the entire datase. These differences arise because smaller datasets require adjustments to ensure effective training and prevent issues, such as overfitting and instability. For instance, smaller batch sizes are used with limited data to ensure the model can effectively utilize the available samples without overfitting. Additionally, the dimensions of the generator and discriminator networks are reduced to avoid over-parameterization, which can otherwise lead to training instability. Furthermore, lower learning rates are employed for individual categories to allow the model to minimize the risk of erratic updates caused by limited data variability. More training epochs are also allocated, enabling the model to make updates for learning meaningful patterns.

Table 22. Training Parameters for Each Class, Boruta V2

Class	Batch Size	Epochs	PAC	Embedding Dim	Generator LR	Discriminator LR	Generator Dim	Discriminator Dim
Worms	4	100	4	64	2×10^{-5}	2×10^{-5}	(128, 64)	(128, 64)
Shellcode	16	100	8	64	1×10^{-4}	1×10^{-4}	(128, 64)	(128, 64)
Backdoor	16	100	8	64	1×10^{-4}	1×10^{-4}	(128, 64)	(128, 64)
Analysis	16	100	8	64	1×10^{-4}	1×10^{-4}	(128, 64)	(128, 64)
Reconnaissance	250	50	10	512	1×10^{-4}	1×10^{-4}	(1024, 512, 256)	(1024, 512, 256)
DoS	250	50	10	512	1×10^{-4}	1×10^{-4}	(1024, 512, 256)	(1024, 512, 256)
Fuzzers	250	50	10	512	1×10^{-4}	1×10^{-4}	(1024, 512, 256)	(1024, 512, 256)

When combining synthetic data from different categories, although this approach alleviates the issue of data imbalance, it does not significantly improve the model's performance. As shown in Figure 5, the model achieves an accuracy of 0.753, precision of 0.84, recall of 0.75, F1-score (micro) of 0.77, and F1-score (macro) of 0.45. These metrics highlight the challenges of effectively integrating synthetic data across categories. To DoS class 15736 synthetic samples were added, to Analysis class 26000 samples were added, to Backdoor 26254 samples and to Worm 27870 samples.

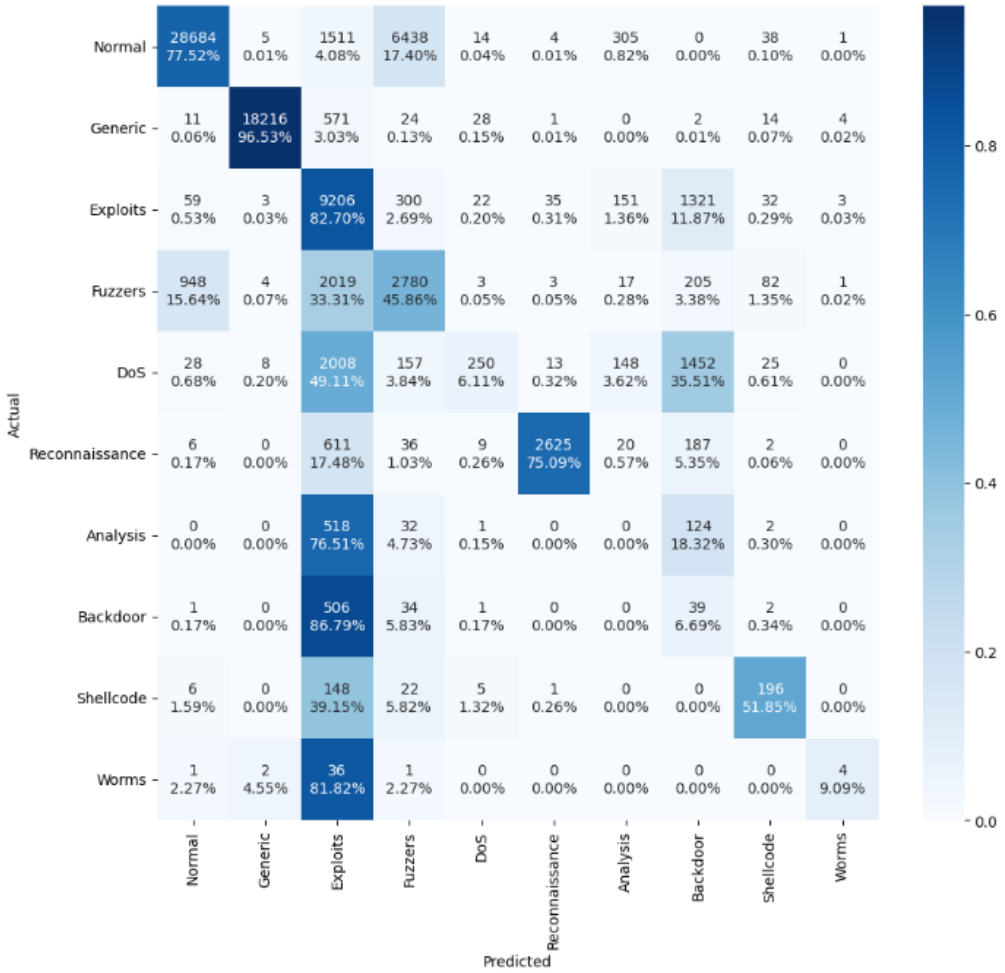

Figure 5. Confusion matrix with synthetic data from separate class models

Additionally, this combined approach increases the likelihood of data drift, as demonstrated in Figure 6. In this case, data drift was detected in 89.286% of the columns (25 out of 28), underscoring the difficulty of maintaining feature consistency when synthesizing data across multiple classes.

Figure 6. Confusion Matrix and Data Drift Analysis with Combined Classes, Boruta V2

However, a notable improvement in model performance is observed when the sample size of the Exploits class is reduced to 10,000. As shown in Figure 7, this adjustment yields an accuracy of 0.763, precision of 0.84, recall of 0.76, F1-score (micro) of 0.79, and F1-score (macro) of 0.50. Though in this case, a few classes start getting confused with the DoS

class.

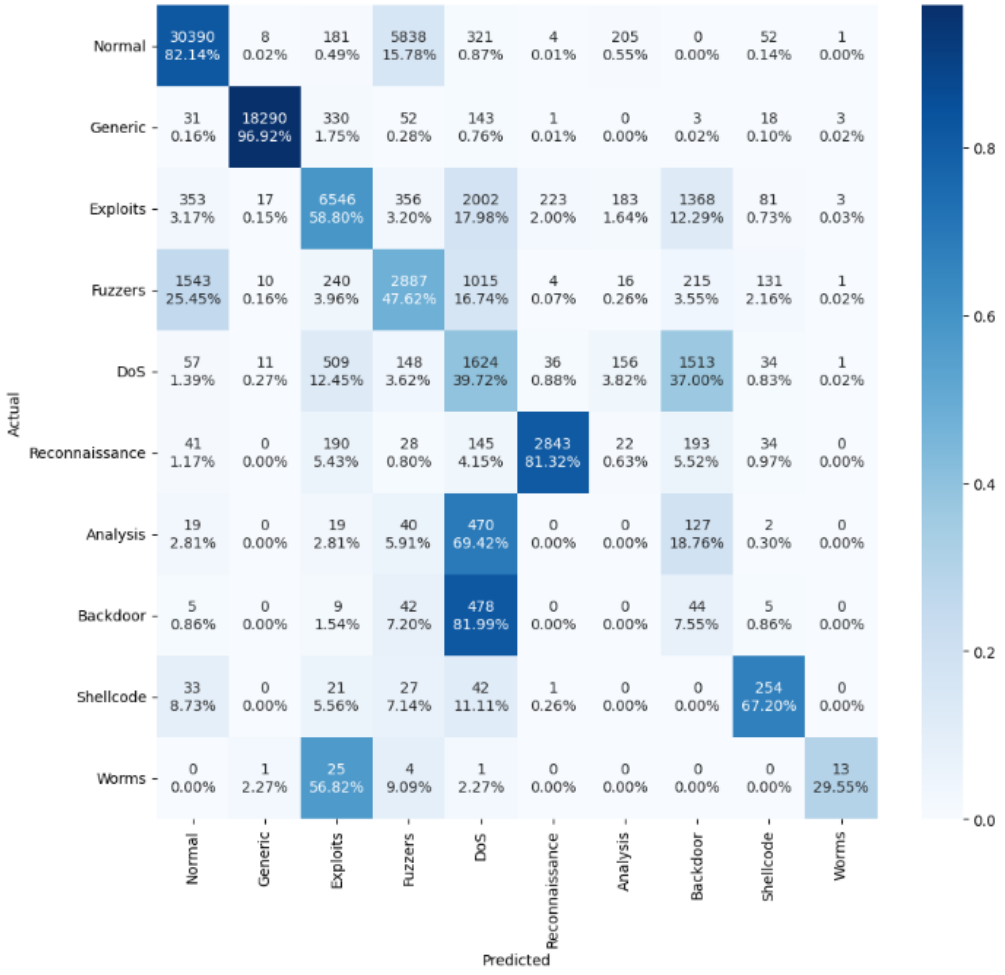

Figure 7. Confusion matrix with reduced Exploits samples, Boruta V2

In Table 23, the training data is organized by categories, and CTGAN is trained on each category separately. This approach allows for flexible customization, enabling selective addition of synthetic data to specific categories based on performance or data availability. Two main strategies were applied:

1. Performance-Based Augmentation: Synthetic samples were added only to the lowest-performing categories, which is taken from Figure 4 (categories 2, 4, and 6).
2. Data Availability-Based Augmentation: Synthetic samples were added to categories with lower data volumes (categories 4, 5, 6, 7, 8, and 9) compared to the larger categories.

Each category's dataset was trained individually for 10, 25, and 50 epochs. For the 25-epoch training, certain parameters were modified to explore whether they could improve accuracy. Results indicate that training each category independently for 10 epochs generally

suffices, as increasing the number of epochs tends to reduce accuracy.

However, training each category separately introduces some degree of data drift, especially in categories with fewer samples. In smaller categories, data drift could be observed in all columns (23 out of 23). Interestingly, while training for more epochs can decrease model accuracy, it also appears to mitigate the extent of data drift. For instance, for the generic category, data drift was observed across columns as follows:

1. 10 epochs: 52.17% of columns exhibited data drift,
2. 25 epochs (with adjusted parameters): 30.43% in the first instance, 39.13% in the second,
3. 50 epochs: 21.79%.

These results suggest that although extended training can negatively affect accuracy, it may help maintain the distribution of synthetic data closer to that of the original data, reducing data drift for categories with limited samples.

Table 23. Results with CTGAN, Boruta V1, separate categories, Random Forest

Method	Accuracy	Precision	Recall	F1-score
cat epochs10				
Added +10,000 to cat 2, 4, 6	0.758	0.84	0.76	0.78
Added +10,000 to cat 4, 5, 6, 7, 8, 9	0.759	0.83	0.76	0.78
cat epochs25 1				
Added +10,000 to cat 2, 4, 6	0.756	0.83	0.76	0.78
Added +10,000 to cat 4, 5, 6, 7, 8, 9	0.757	0.83	0.76	0.78
cat epochs25 2				
Added +10,000 to cat 2, 4, 6	0.758	0.83	0.76	0.78
Added +10,000 to cat 4, 5, 6, 7, 8, 9	0.758	0.83	0.76	0.78
cat epochs50				
Added +10,000 to cat 2, 4, 6	0.757	0.83	0.76	0.78
Added +10,000 to cat 4, 5, 6, 7, 8, 9	0.756	0.83	0.76	0.78

4. Discussion

The experiments conducted in this work demonstrate the potential of using synthetic data generated by CTGAN to enhance the performance of machine learning models tasked with intrusion detection in imbalanced datasets. Notably, the addition of synthetic data provides a viable alternative to traditional oversampling methods, such as SMOTE, mitigating common pitfalls such as overfitting and class imbalance.

Throughout the experiments, it was observed that when synthetic data generation was combined with feature selection methods like Boruta, the model performance in terms of accuracy, precision, recall, and F1-score showed some improvement. The evaluation metrics indicated the possibility of combining synthetic and real data for better classification performance, especially in underrepresented classes. The results show that tailored synthetic data generation — targeting specific categories — can fix gaps in data representation, particularly for less-represented attack types in intrusion detection scenarios.

Combining real and synthetic data consistently outperformed the use of synthetic data alone, showing the importance of blending both data sources to retain predictive accuracy. Although larger quantities of synthetic data beyond 10% offered minimal additional benefit, tuning GAN parameters - such as embedding dimensions, batch sizes, and discriminator steps - showed substantial improvements in both model consistency and training efficiency. Additionally, training each category independently revealed that shorter training durations (10 epochs) yielded optimal accuracy, though it could lead to significant data drift, especially in categories with smaller data samples.

Though the data showed to be similar to the original data, making it possible to say that the synthetic data is high quality, there isn't much variance and hence the synthetic data doesn't add much new that could benefit the model. To combat that, the author suggests, it could be beneficial to try adding noise. Noise can be added in different stadiums - 1) before training the GAN, injecting noise into real data, 2) by modifying the GAN directly (for example, the discriminator), 3) after creating the synthetic data.

However, while CTGAN with Boruta significantly aids in managing imbalances, the highest observed performance was achieved using a neural network approach, specifically a Bi-LSTM CNN with Boruta feature selection. This model achieved an accuracy of 79%, with a precision of 81%, recall of 76%, and F1-score of 78%, marking it as the preferred

method when the goal is maximum predictive accuracy.

4.1 Future work

There remains a critical need to enhance the generation process of synthetic data to better capture the complexity and characteristics of real-world intrusion detection scenarios. Future studies could focus on improving the training of GANs through more complicated loss functions or augmenting existing data to minimize data drift. It would also be advantageous to explore different configurations of GAN architectures and training strategies to reduce epoch-based degradation effects on performance metrics.

A more extensive study about the detailed dynamics of data drift, employing a wider array of metrics to quantify the drift accurately, could yield important insights. Adapting the model based on feedback from the drift reports, informed by new metrics, would make it possible to make the most out of generative data augmentation strategies.

As the current tested models create data that is too similar to the original data, it could be beneficial to investigate introducing noise, either during feature engineering, training or post-training.

To gain a more comprehensive understanding of the most efficient parameters, nested cross-validation should be employed. However, due to time constraints and limited computational resources, this approach was not pursued and remains as a potential area for future work.

To further enhance the validation process, experimenting with newer GAN variants or integrating hybrid models could provide deeper insights into the optimal parameter configurations.

Additionally, it would be valuable to include comparative results using binary classification, as well as explore the potential of creating new data using LLMs and transformer architectures. Investigating these models could provide deeper insights and further enhance the performance of IDSs.

5. Conclusion

This study addresses an important issue in the field of cybersecurity: class imbalance in IDSs used with IoT datasets. When there exists class imbalance, machine learning models struggle to accurately identify minority classes due to a lack of sufficient data. This study explores how generative models, specifically CTGAN, can help improve model performance by creating synthetic data to balance the classes.

Synthetic data of reasonable quality can be generated, though a key challenge identified in this study is that the created data tends to be too similar to the original data, limiting the model's ability to demonstrate significant improvements. Despite this, it was observed that even with less-than-ideal synthetic data, the model's performance improved due to increased data diversity. Therefore, the author suggests that future studies explore methods to introduce noise into the synthetic data, reducing its similarity to the original and enhancing its overall effectiveness.

Future studies could focus on exploring more advanced GAN setups, introducing new loss functions, and refining training approaches to address challenges like data drift, ensuring the synthetic data remains relevant and useful over time.

In conclusion, this work contributes to the understanding of synthetic data creation and intrusion detection in IoT environments, highlighting the need for solutions to handle class imbalance. While providing an overview of existing models and approaches, it also shows the advantages of utilizing synthetic data.

Potential real-world applications of these findings include improving the resilience of smart homes, industrial IoT systems, and healthcare IoT devices. For instance, in smart homes, balanced IDSs can better detect unauthorized access attempts, protecting residents' privacy and security. In industrial IoT systems, enhanced IDSs can prevent disruptions caused by cyber-attacks, ensuring the continuous operation of critical infrastructure. In healthcare IoT devices, accurate detection of malicious activities can safeguard sensitive patient data and maintain the integrity of medical devices.

Beyond IoT, the broader impact of this work on cybersecurity includes its relevance to general intrusion detection research. By addressing class imbalance, the techniques explored in this study can improve the performance of IDSs across various domains, leading

to more reliable cybersecurity solutions. This highlights the importance of synthetic data creation in enhancing the overall effectiveness of intrusion detection systems.

References

[1] Mohamed Amine Ferrag et al. "Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning". In: *IEEE Access* 10 (2022), pp. 40281–40306. DOI: 10.1109/ACCESS.2022.3165809.

[2] Nour Moustafa. "A new distributed architecture for evaluating AI-based security systems at the edge: Network TON$_{IoT datasets}$". In: *Sustainable Cities and Society* 72 (2021), p. 102994. ISSN: 2210-6707. DOI: https://doi.org/10.1016/j.scs.2021.102994. URL: https://www.sciencedirect.com/science/article/pii/S2210670721002808.

[3] Euclides Carlos Pinto Neto et al. "CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment". In: *Sensors* 23.13 (2023). ISSN: 1424-8220. DOI: 10.3390/s23135941. URL: https://www.mdpi.com/1424-8220/23/13/5941.

[4] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[5] KG Raghavendra Narayan et al. *IIDS: Design of Intelligent Intrusion Detection System for Internet-of-Things Applications*. 2023. arXiv: 2308.00943 [cs.CR].

[6] Vikash Kumar and Ditipriya Sinha. "Synthetic attack data generation model applying generative adversarial network for intrusion detection". In: *Computers Security* 125 (2023), p. 103054. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.103054. URL: https://www.sciencedirect.com/science/article/pii/S0167404822004461.

[7] Tej Kiran Boppana and Priyanka Bagade. "GAN-AE: An unsupervised intrusion detection system for MQTT networks". In: *Engineering Applications of Artificial Intelligence* 119 (2023), p. 105805. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2022.105805. URL: https://www.sciencedirect.com/science/article/pii/S0952197622007953.

[8] Ghada Abdelmoumin, Danda B. Rawat, and Abdul Rahman. "On the Performance of Machine Learning Models for Anomaly-Based Intelligent Intrusion Detection Systems for the Internet of Things". In: *IEEE Internet of Things Journal* 9.6 (2022), pp. 4280–4290. DOI: 10.1109/JIOT.2021.3103829.

[9] Jessica L. Purser. “Using Generative Adversarial Networks for Intrusion Detection in Cyber-Physical Systems”. In: Naval Postgraduate School, 2020.

[10] Ying Zhang and Qiang Liu. “On IoT intrusion detection based on data augmentation for enhancing learning on unbalanced samples”. In: *Future Generation Computer Systems* 133 (2022), pp. 213–227. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2022.03.007. URL: https://www.sciencedirect.com/science/article/pii/S0167739X22000826.

[11] Hakan Can Altunay and Zafer Albayrak. “A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks”. In: *Engineering Science and Technology, an International Journal* 38 (2023), p. 101322. ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2022.101322. URL: https://www.sciencedirect.com/science/article/pii/S2215098622002312.

[12] Joel Margolis et al. “An In-Depth Analysis of the Mirai Botnet”. In: *2017 International Conference on Software Security and Assurance (ICSSA)*. 2017, pp. 6–12. DOI: 10.1109/ICSSA.2017.12.

[13] David Kushner. *The Real Story of Stuxnet*. Published 26 Feb 2013. Accessed 24 May 2024. IEEE Spectrum. 2013. URL: https://spectrum.ieee.org/the-real-story-of-stuxnet.

[14] IBM. “What is the internet of things?” In: URL: https://www.ibm.com/topics/internet-of-things.

[15] Anna L. Buczak and Erhan Guven. “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection”. In: *Commun. Surveys Tuts.* 18.2 (Apr. 2016), pp. 1153–1176. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2494502. URL: https://doi.org/10.1109/COMST.2015.2494502.

[16] Khraisat A. et al. “Survey of intrusion detection systems: techniques, datasets and challenges”. In: *Cybersecurity* 2 (2019). ISSN: 2523-3246. DOI: 10.1186/s42400-019-0038-7.

[17] Khaled El Emam, Lucy Mosquera, and Richard Hoptroff. In: *Practical Synthetic Data Generation*. O’Reilly Media, Inc., 2020.

[18] Fida K. Dankar, Mahmoud K. Ibrahim, and Leila Ismail. “A Multi-Dimensional Evaluation of Synthetic Data Generators”. In: *IEEE Access* 10 (2022), pp. 11147–11158. DOI: 10.1109/ACCESS.2022.3144765.

[19] Jianyu Wang et al. “Def-IDS: An Ensemble Defense Mechanism Against Adversarial Attacks for Deep Learning-based Network Intrusion Detection”. In: *2021 International Conference on Computer Communications and Networks (ICCCN)*. 2021, pp. 1–9. DOI: 10.1109/ICCCN52240.2021.9522215.

[20] Danni Yuan et al. “Intrusion Detection for Smart Home Security Based on Data Augmentation with Edge Computing”. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148632.

[21] Hasan Abdulameer, Inam Musa, and Assis. Prof. Noora Salim. “Three level intrusion detection system based on conditional generative adversarial network”. In: *International Journal of Electrical and Computer Engineering* 13 (Dec. 2022). DOI: 10.11591/ijece.v13i2.pp2240-2258.

[22] Ayesha S. Dina, A.B. Siddique, and D. Manivannan. “A deep learning approach for intrusion detection in Internet of Things using focal loss function”. In: *Internet of Things* 22 (2023), p. 100699. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2023.100699. URL: https://www.sciencedirect.com/science/article/pii/S2542660523000227.

[23] Kim-Hung Le et al. “IMIDS: An Intelligent Intrusion Detection System against Cyber Threats in IoT”. In: *Electronics* 11.4 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11040524. URL: https://www.mdpi.com/2079-9292/11/4/524.

[24] Hafsa Benaddi et al. “Adversarial Attacks Against IoT Networks using Conditional GAN based Learning”. In: *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*. 2022, pp. 2788–2793. DOI: 10.1109/GLOBECOM48099.2022.10000726.

[25] Imtiaz Ullah and Qusay H. Mahmoud. “A Framework for Anomaly Detection in IoT Networks Using Conditional Generative Adversarial Networks”. In: *IEEE Access* 9 (2021), pp. 165907–165931. DOI: 10.1109/ACCESS.2021.3132127.

[26] Omar Habibi, Mohammed Chemmakha, and Mohamed Lazaar. “Imbalanced tabular data modelization using CTGAN and machine learning to improve IoT Botnet attacks detection”. In: *Engineering Applications of Artificial Intelligence* 118 (2023), p. 105669. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2022.105669. URL: https://www.sciencedirect.com/science/article/pii/S0952197622006595.

[27] Basim Ahmad Alabsi, Mohammed Anbar, and Shaza Dawood Ahmed Rihan. "Conditional Tabular Generative Adversarial Based Intrusion Detection System for Detecting Ddos and Dos Attacks on the Internet of Things Networks". In: *Sensors* 23.12 (2023), p. 5644. DOI: 10.3390/S23125644. URL: https://doi.org/10.3390/s23125644.

[28] Guosheng Zhao et al. "IoT intrusion detection model based on gated recurrent unit and residual network". In: *Peer-to-Peer Networking and Applications* 16 (2023). ISSN: 4.

[29] Paulo Freitas de Araujo-Filho et al. "Unsupervised GAN-Based Intrusion Detection System Using Temporal Convolutional Networks and Self-Attention". In: *IEEE Transactions on Network and Service Management* 20.4 (2023), pp. 4951–4963. DOI: 10.1109/TNSM.2023.3260039.

[30] Pradeepkumar Bhale et al. "OPTIMIST: Lightweight and Transparent IDS With Optimum Placement Strategy to Mitigate Mixed-Rate DDoS Attacks in IoT Networks". In: *IEEE Internet of Things Journal* 10.10 (2023), pp. 8357–8370. DOI: 10.1109/JIOT.2023.3234530.

[31] Yalong Song et al. "Intrusion Detection for Internet of Things Networks using Attention Mechanism and BiGRU". In: *2023 5th International Conference on Electronic Engineering and Informatics (EEI)*. 2023, pp. 227–230. DOI: 10.1109/EEI59236.2023.10212791.

[32] Cheolhee Park et al. "An Enhanced AI-Based Network Intrusion Detection System Using Generative Adversarial Networks". In: *IEEE Internet of Things Journal* 10.3 (2023), pp. 2330–2345. DOI: 10.1109/JIOT.2022.3211346.

[33] Shuang Zhao et al. "attackGAN: Adversarial Attack against Black-box IDS using Generative Adversarial Networks". In: *Procedia Computer Science* 187 (2021). 2020 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI2020, pp. 128–133. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.04.118. URL: https://www.sciencedirect.com/science/article/pii/S1877050921009303.

[34] Mustafizur R. Shahid et al. "Generative Deep Learning for Internet of Things Network Traffic Generation". In: *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. 2020, pp. 70–79. DOI: 10.1109/PRDC50213.2020.00018.

[35] Shiming Li et al. “EIFDAA: Evaluation of an IDS with function-discarding adversarial attacks in the IIoT”. In: *Heliyon* 9.2 (2023), e13520. ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2023.e13520. URL: https://www.sciencedirect.com/science/article/pii/S2405844023007272.

[36] Arpita Srivastava, Ditipriya Sinha, and Vikash Kumar. “WCGAN-GP based synthetic attack data generation with GA based feature selection for IDS”. In: *Computers Security* 134 (2023), p. 103432. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103432. URL: https://www.sciencedirect.com/science/article/pii/S0167404823003425.

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

[38] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[39] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[40] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[41] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[42] Michael L. Waskom. “seaborn: statistical data visualization”. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[43] Lei Xu et al. “Modeling Tabular data using Conditional GAN”. In: *Advances in Neural Information Processing Systems*. 2019.

[44] Andrzej Maćkiewicz and Waldemar Ratajczak. “Principal components analysis (PCA)”. In: *Computers Geosciences* 19.3 (1993), pp. 303–342. ISSN: 0098-3004. DOI: https://doi.org/10.1016/0098-3004(93)90090-R. URL: https://www.sciencedirect.com/science/article/pii/009830049390090R.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Gerlin Vainomäe

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Balancing Classes with Synthetic Data for IoT Intrusion Detection Systems", supervised by Zhe Deng and Ants Torim
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

06.01.2025

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Confusion Matrices of Categories

Figure 8. Worms class, confusion matrix with 10000 added synthetic samples, Boruta V2

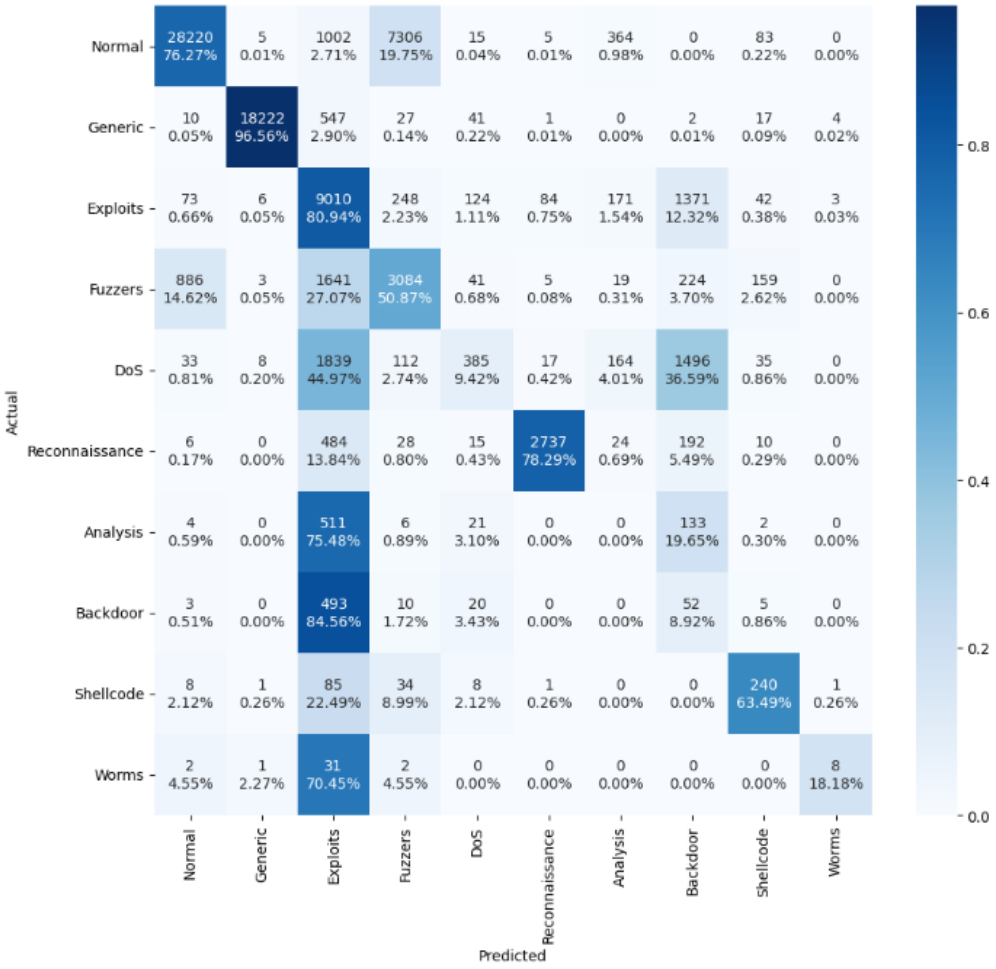

Figure 9. Worms class, confusion matrix with 28000 synthetic samples, Boruta V2

Figure 10. Shellcode class, confusion matrix with 10000 added synthetic samples, Boruta V2

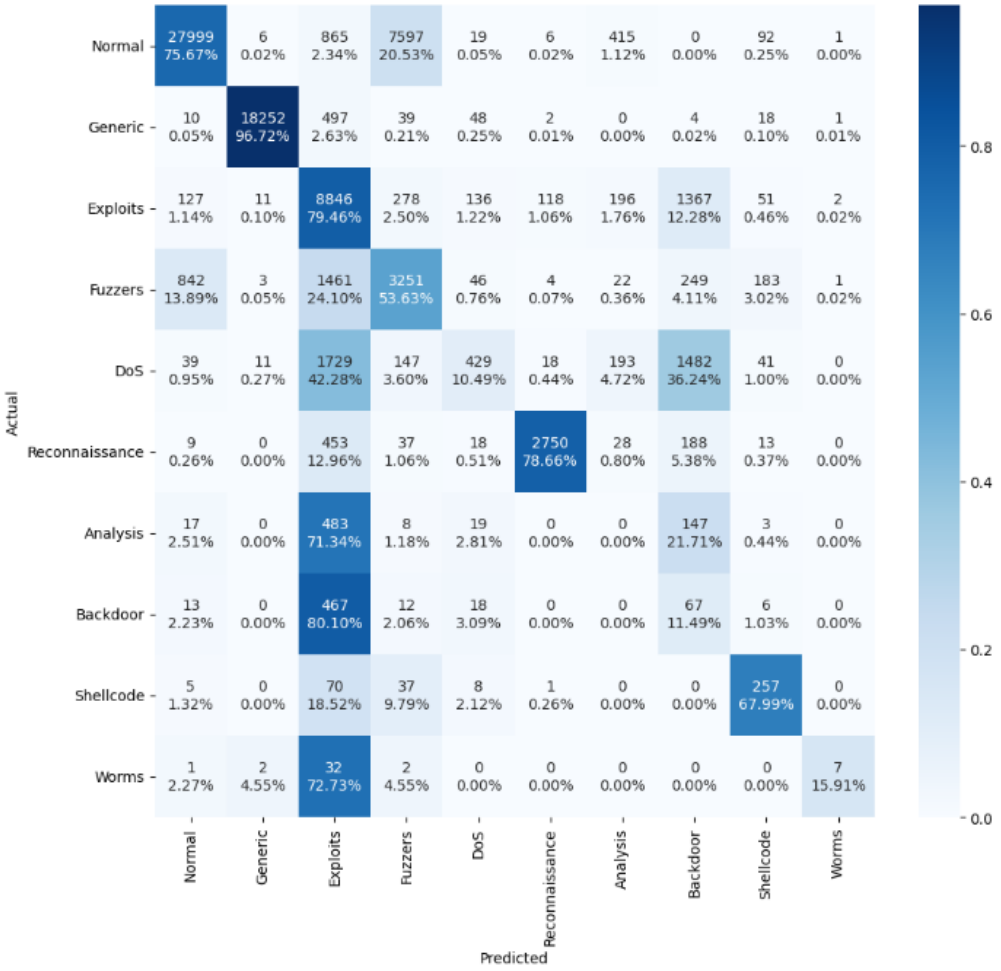

Figure 11. Shellcode class, confusion matrix with 28000 synthetic samples, Boruta V2

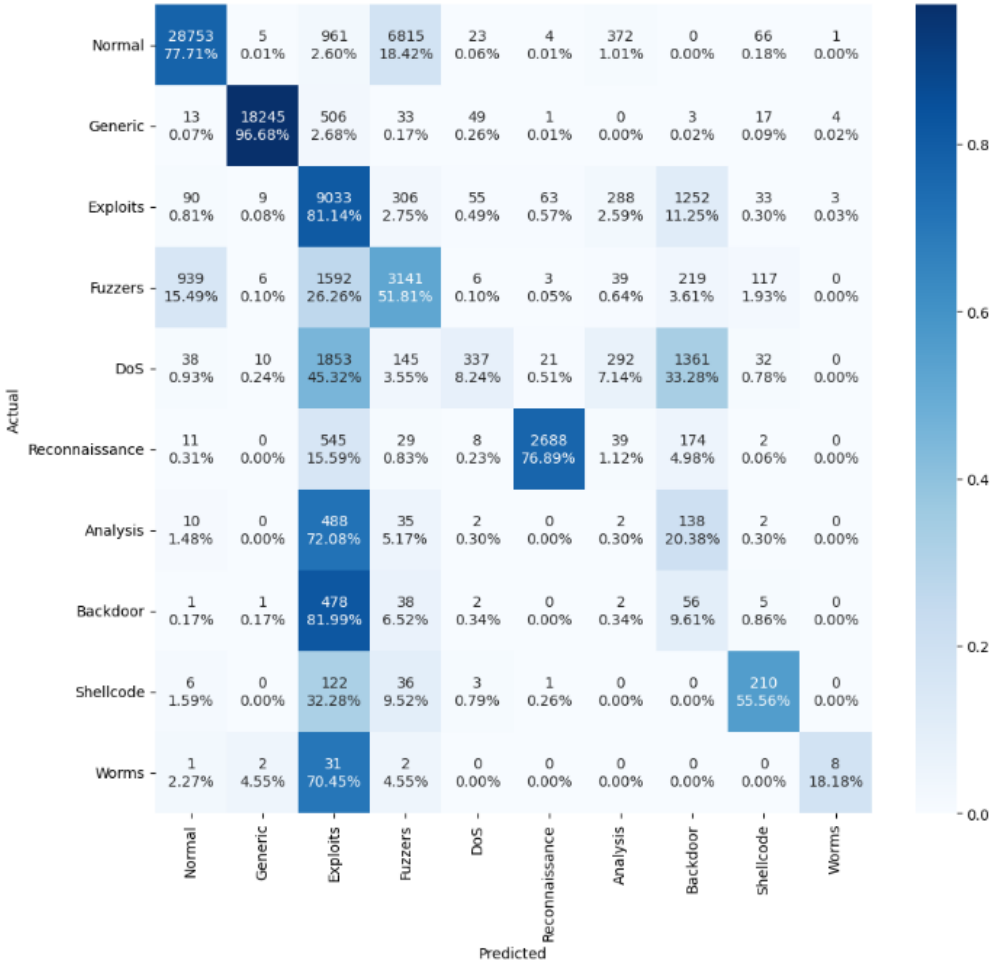

Figure 12. Backdoor class, confusion matrix with 10000 added synthetic samples, Boruta V2

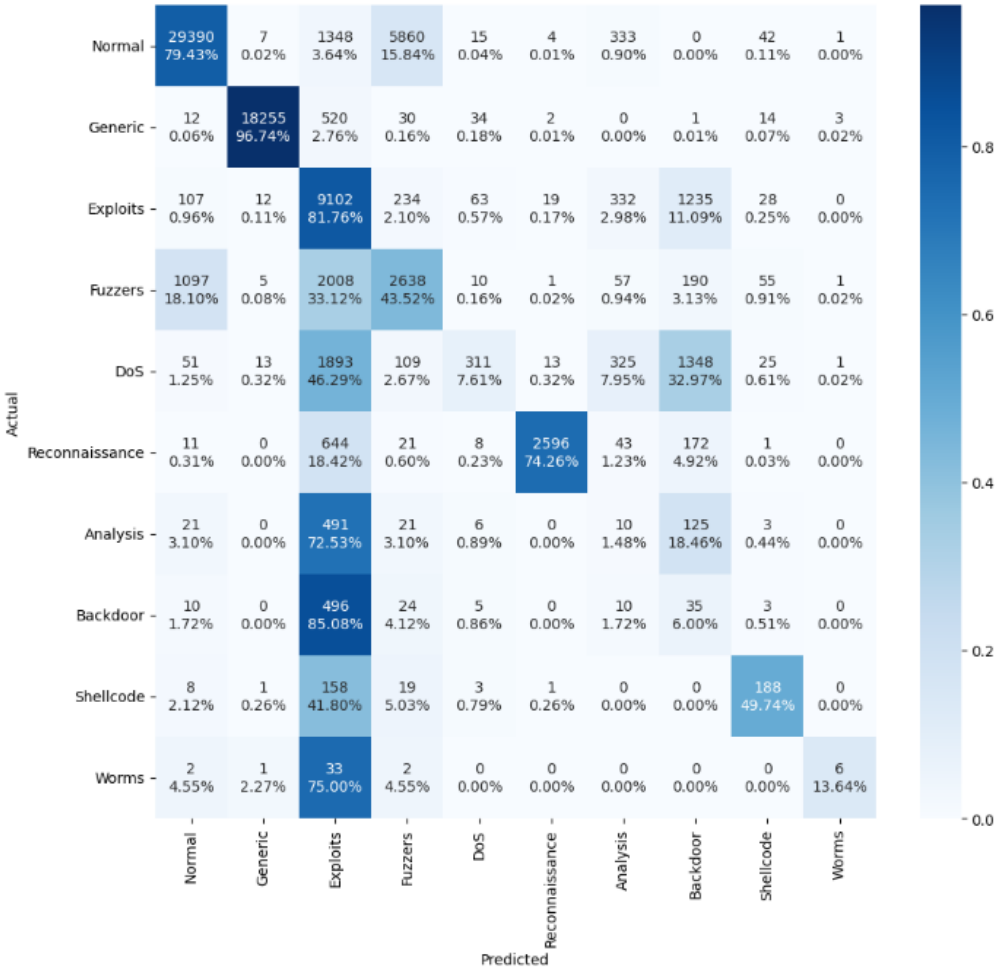

Figure 13. Backdoor class, confusion matrix with 28000 synthetic samples, Boruta V2

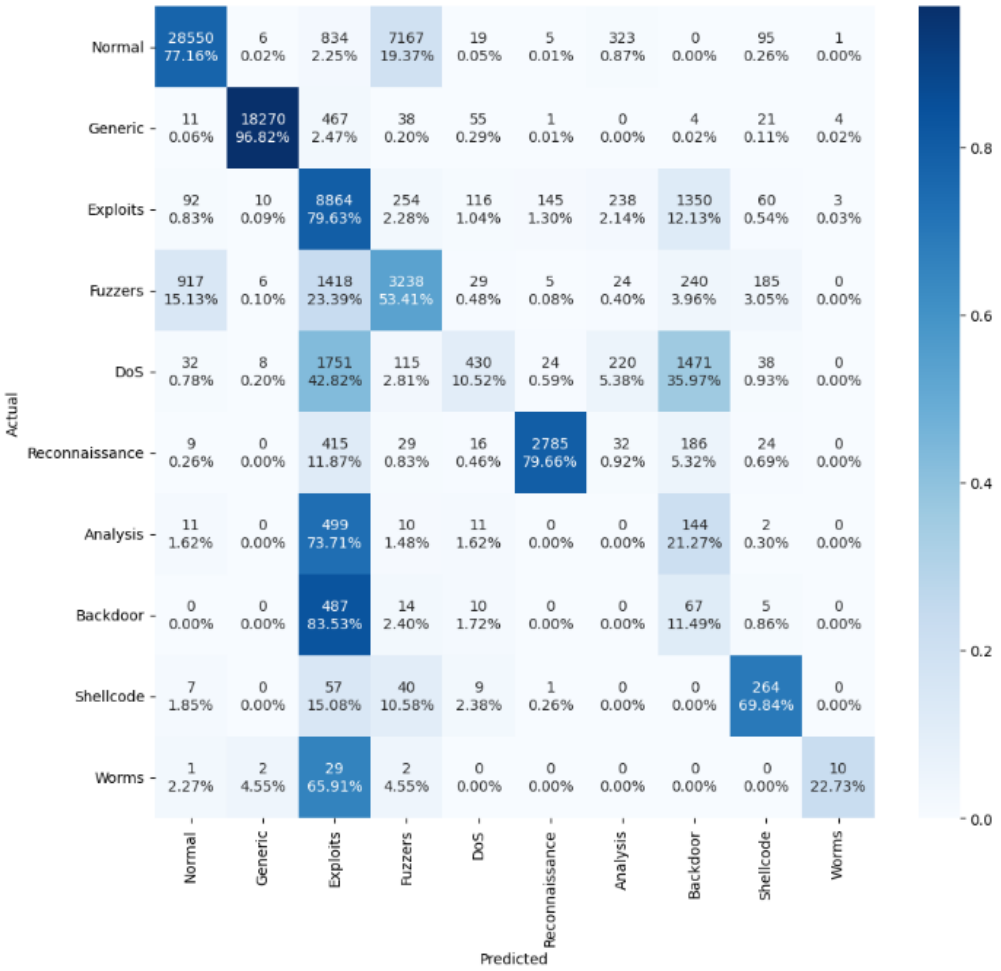

Figure 14. Analysis class, confusion matrix with 10000 added synthetic samples, Boruta V2

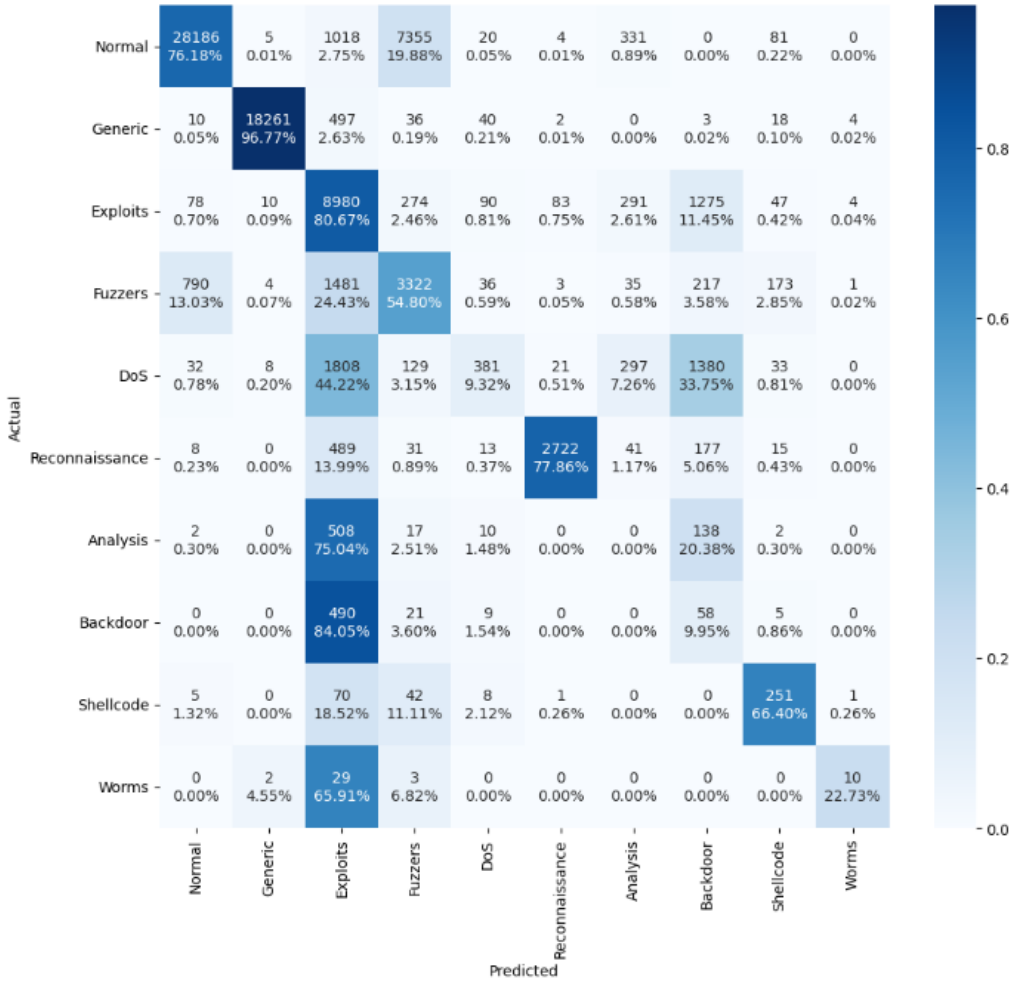

Figure 15. Analysis class, confusion matrix with 28000 synthetic samples, Boruta V2

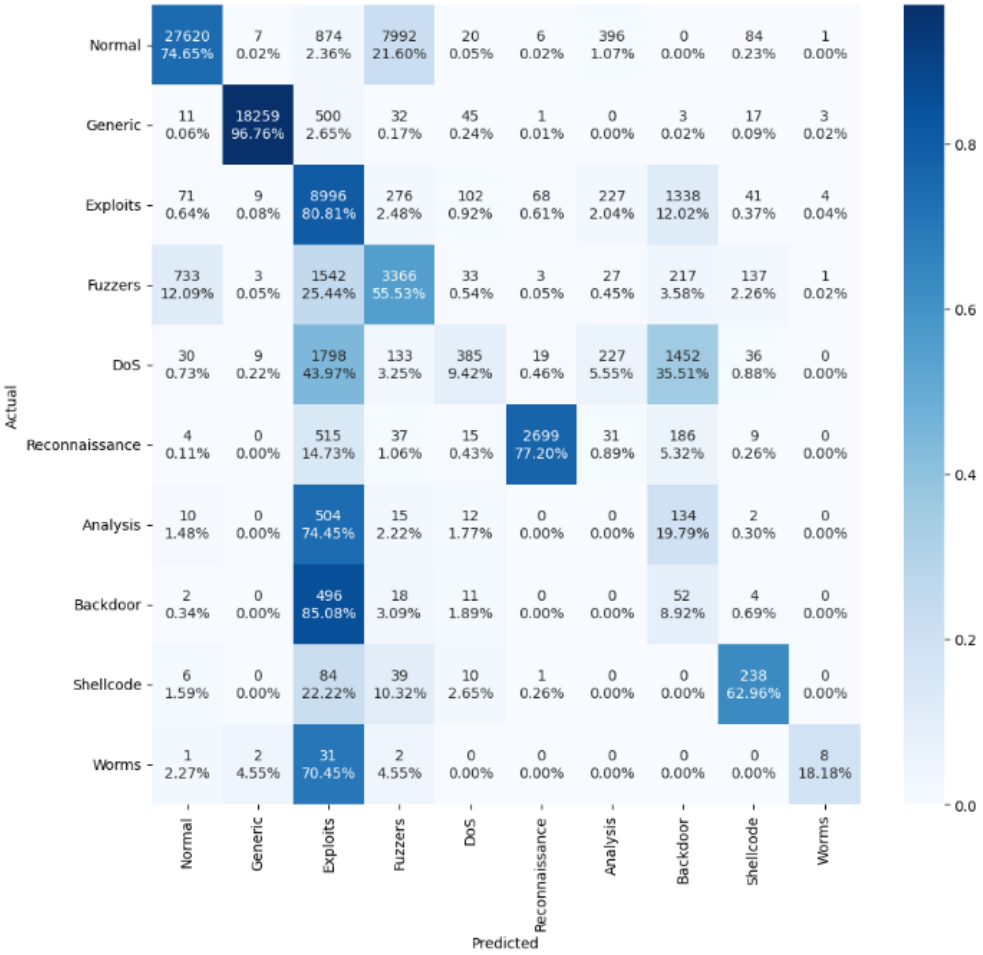

Figure 16. Reconnaissance class, confusion matrix with 10000 added synthetic samples, Boruta V2

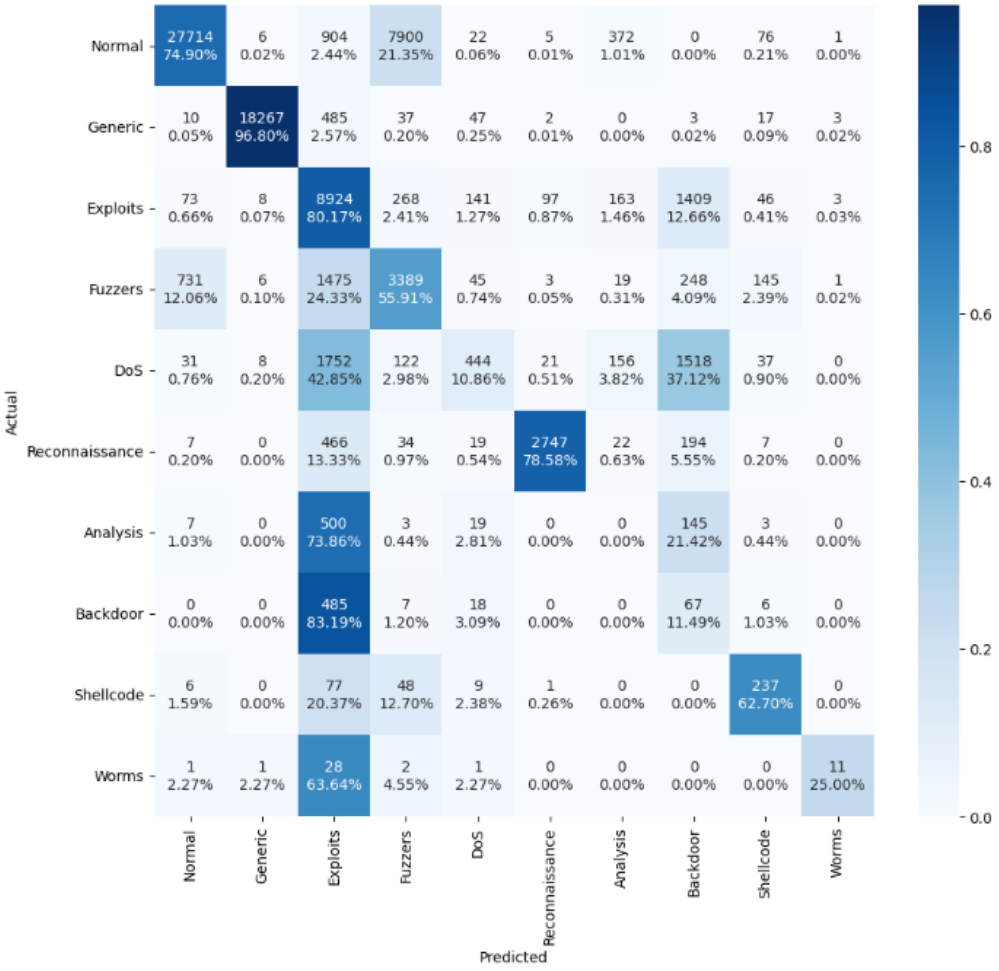

Figure 17. Reconnaissance class, confusion matrix with 28000 synthetic samples, Boruta V2

Figure 18. DoS class, confusion matrix with 10000 added synthetic samples, Boruta V2

Figure 19. DoS class, confusion matrix with 28000 synthetic samples, Boruta V2

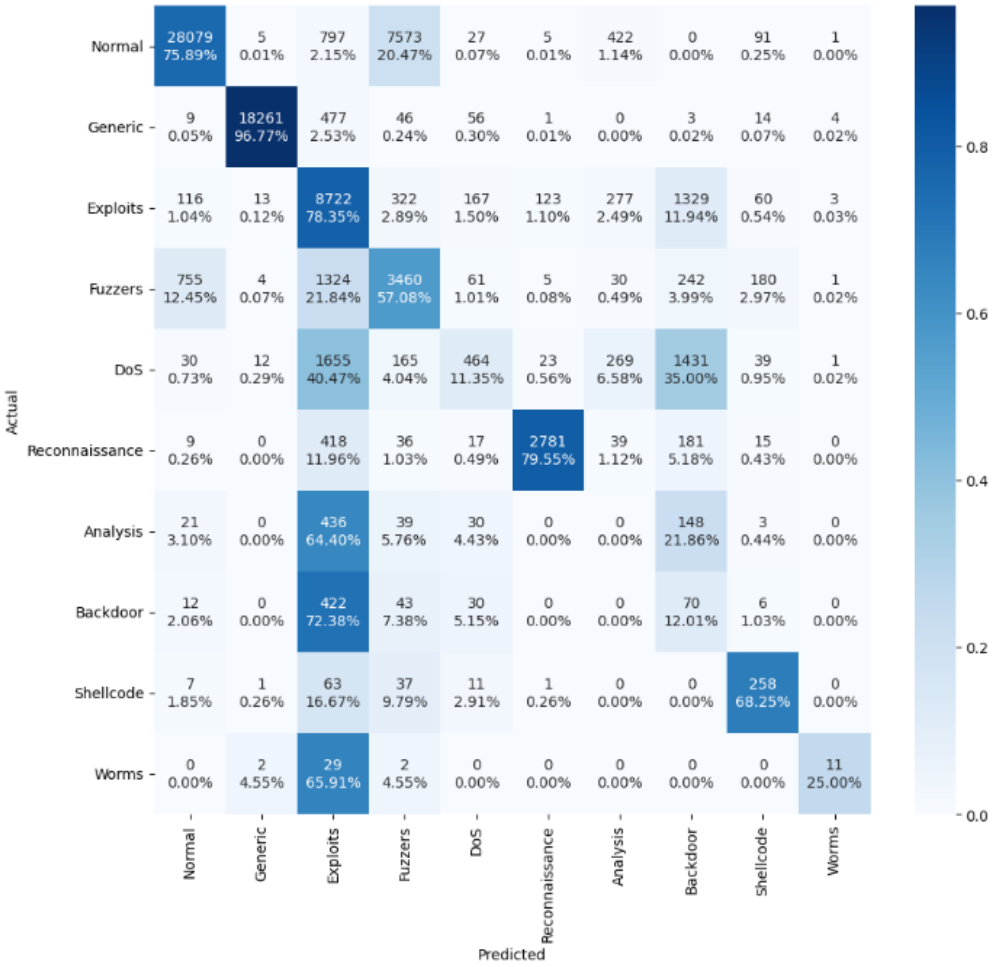

Figure 20. Fuzzers class, confusion matrix with 10000 added synthetic samples, Boruta V2

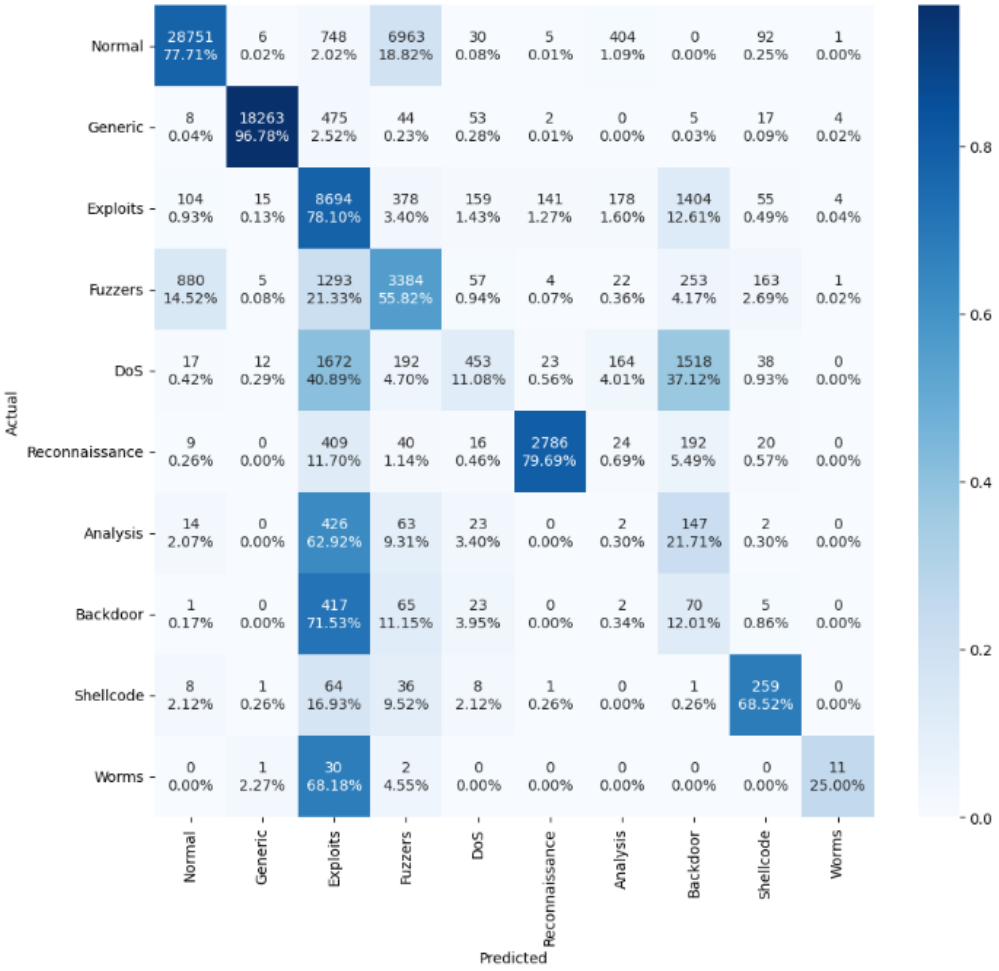

Figure 21. Fuzzers class, confusion matrix with 28000 synthetic samples, Boruta V2

Figure 22. Exploits class, confusion matrix with 10000 added synthetic samples, Boruta V2

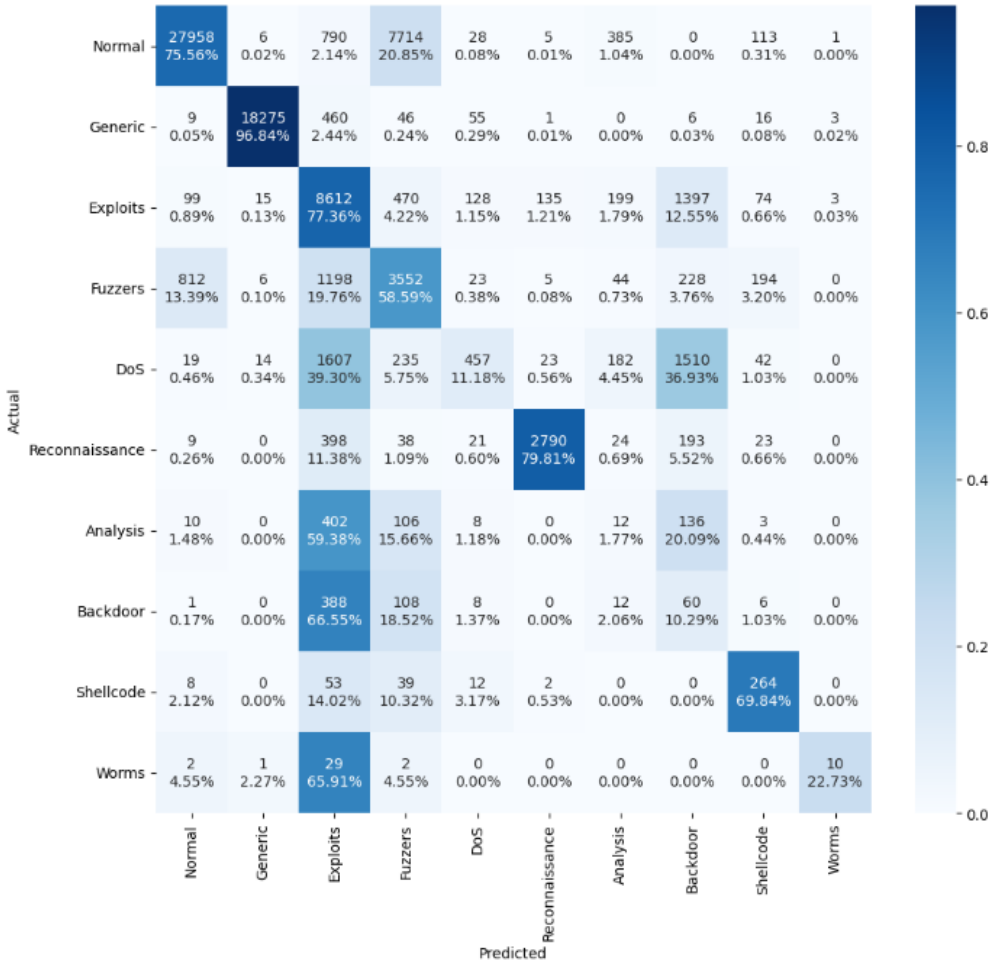

Figure 23. Generic class, confusion matrix with 10000 added synthetic samples, Boruta V2

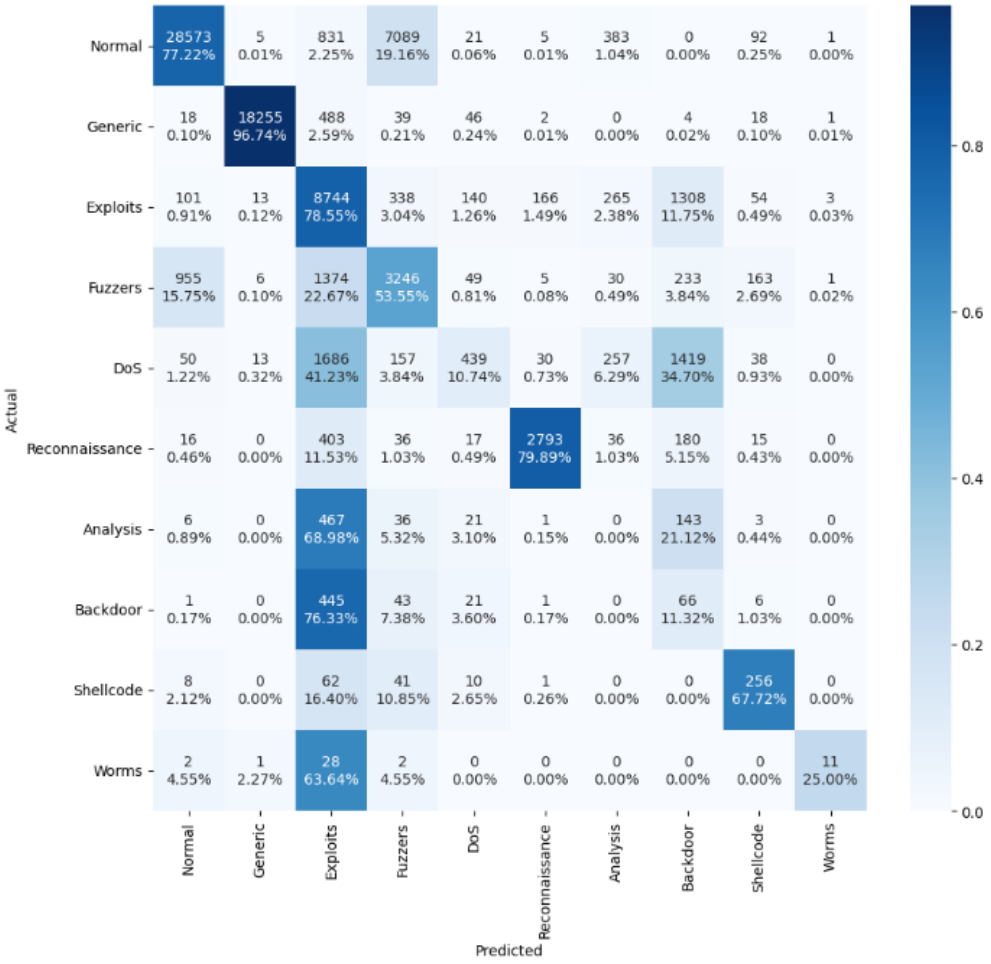

Figure 24. Normal class, confusion matrix with 10000 added synthetic samples, Boruta V2