

Intrusion detection System Using Machine Learning



**A graduation project document submitted
to the department of cyber security as
partial fulfillment for the requirement of
the B.Sc. degree in cyber security**

Prepared by

**Alaa Mohammed Aldebis (20210384)
Ahmad Ibrahim Joudeh (20214179)
Mahmoud Mohammad Al Essawi (20221143)**

Supervised by

Dr. Abdullah Qammaz

Committee Report

January,2025

We certify that we have read this graduation project report as examining committee, examined the student in its content and that in our opinion it is adequate as a project document for B.Sc. in Cyber Security.

Chairman:

Name:

Signature:

Date: / /

Supervisor:

Name:

Signature:

Date: / /

Examiner:

Name:

Signature:

Date: / /

ABSTRACT

Cybercrime has grown to become more than anyone ever considered it could be. In this day and age, people are very much relying on digital as well as communication technologies to do things in their lives that it can get a feeling of being in a scenario where the known methods to detect cyber threats, which are signature-based, have been working against traditional attacks, but are never sure of how to address advanced ones such as unknown or zero-day attacks.

This study has delved into anomaly detection as a result to this foregone conclusion. By harnessing machine learning algorithms like One-Class SVM and its counter-part, Isolation Forest, it has further enhanced intrusion detection to enrich in its coverage of an area called network traffic-analysis for abnormal behavior. Such algorithms demonstrated remarkable improvement here using an optimized NSL-KDD dataset with respect to detection accuracy and adaptation against ever-changing threats.

ACKNOWLEDGMENTS

Praise be to ALLAH, whose blessings are only fulfilled, and pray for those who have no prophet after him.

We would like to extend our sincere thanks and gratitude to our honorable supervisor, Dr. Abdullah Qammaz , for the outstanding support, guidance and encouragement you provided to us throughout the project's work.

We also would like to express our gratitude and great appreciation to the virtuous discussion committee.

We also thank the Faculty of Information Technology represented by all faculty members without exception for the academic and moral support and guidance they provided us, which had a prominent impact in this effort.

Likewise, do not forget about our dear university, which we cherish our belonging to during this decisive and important period of our lives, which we spent in its entirety.

Finally, the sincere thanks to our people, especially the source of tenderness, care and guidance, the mother and our guide in our lives, the virtuous father for their encouragement, patience, and help over the years. We owe them to them forever and I ask God Almighty to extend our life so that we can fulfill them.

Table of Contents

Committee Report

Abstract

Acknowledgments

1. Introduction

1.1 Background and Motivation

1.2 Problem Statement

1.3 Objectives of the Study

1.4 Organization of the Document

2. Literature Review

2.1 Intrusion Detection Systems (IDS)

2.1.1 Definition

2.1.2 Types of IDSs

2.2 Machine Learning in Cybersecurity

2.2.1 Overview of Machine Learning Techniques

2.2.2 Datasets Used in Cybersecurity

2.3 Anomaly Detection Techniques

3. Methodology

3.1 Dataset Descriptio.....

3.2 Data Preprocessing

3.3 Model Development

3.3.1 Support Vector Machine (SVM)

3.3.2 Isolation Forest

3.4 Model Evaluation

4. IMPLEMENTATION.....

5.1 Load Dataset

5.2 Run Model

5.3 Visualization

5. Conclusion and Future Work

6.1 Summary of Findings

6.2 Future Directions

6. Appendix

7. References

1. Introduction

1.1 Background and Motivation

It is significant for the digital world now, cyber security in safeguarding vital information, personal data, and essential sites. Because the digital space is so interconnected, increasingly complex systems for cyberspace and improved sophistication of attacks, the greatest challenge organizations have is securing their networks from an evil source. Emerging

threats such as ransomware, phishing, denial-of-service, and advanced persistent threats necessitate practical and innovative models of cybersecurity. A major tool in the development and implementation of such models is the Intrusion Detection System.

Through time, there has been a remarkable evolution in IDS technology. The earlier models were essentially of the form static rule-based systems, i.e., they were operating on fixed set of instructions. Such kinds of systems were well good in detecting known threats but did not have the capacity to detect novel attack patterns, thus putting the networks of these organizations at risk of advanced and new kinds of threats. Such systems had the problem of a greater proportion of false positives; as a result, these systems flooded security teams with unnecessary alerts and decreased the efficiency of the security infrastructure.

To solve these challenges, the field of cybersecurity has to offer machine learning (ML) as a powerful, adaptable tool. Machine learning has possible revolutionized IDS because it allows systems to learn from the data dynamically and adapt to new threats. Unlike traditional rule-based systems, ML-powered IDS can uncover hidden patterns within datasets, detect complex relationships among them, and classify activities inside various normal or malicious categories, which are based on learned activities. This transition from static rules to adaptive models paved the way for previously unseen threats such as zero-day attacks or polymorphic malware.

This project aims to develop a modern intrusion detection system based on machine learning technologies which will specifically put stress on the unsupervised-learning methodologies. Thus, using a combination of for instance Isolation Forest and One-Class SVM algorithms, the project aims to be able to provide very strong and scalable solutions with flexibility to present-day challenges in the field of cybersecurity. The project has a lot to do with real-life applications, but also holds a valuable academic part by offering insights and results for better understanding of the role machine learning plays in the field of cybersecurity.

1.1.1 Why Machine Learning?

The present project employed machine learning because of its revolutionary solutions to contemporary cybersecurity challenges. Conventional intrusion detection systems (IDS) have been limited to rule-based or signature-based methods and devoid of the capacity to timely detect zero-day and new threats. Dynamic and scalable, adaptive solution machine learning can evolve with the emerging attacks. Anomaly detection systems also have much

more potential with machine learning to understand the extent of anomaly or the relationship within the data, thereby improving the most intelligent and proactive defense mechanisms. This is considering very aggressive sophistication of cyber threats and thus is an important component in the development of any intrusion detection system.

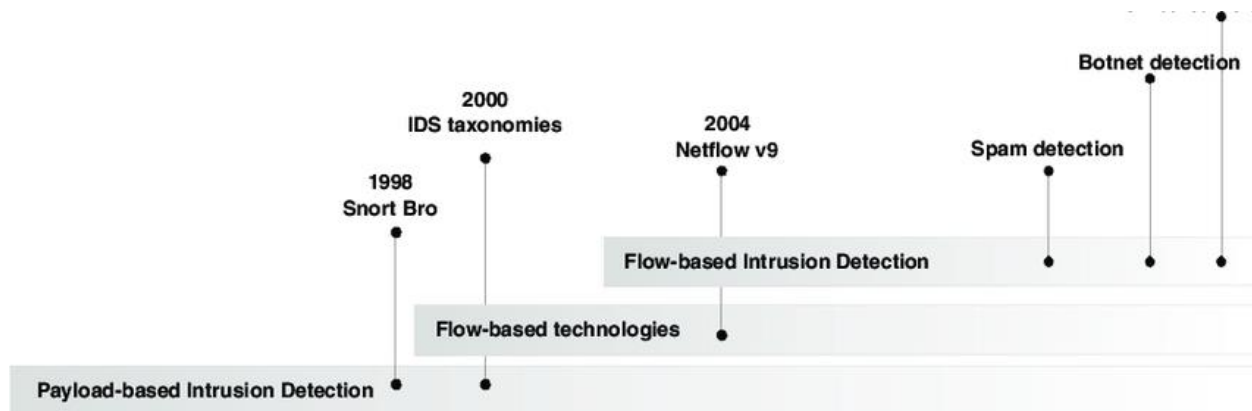


Figure 1: Evolution of Intrusion Detection Systems

these are in fact some wonderful advantages that machine learning offers when processed into an intrusion detection system:

Scalability and Efficiency: Machine learning can process any amount of data at its speed these algorithms have significant competitive advantages over others for large-scale network and application usage on the cloud.

Anomaly Detection: Algorithms like Isolation Forest and One-Class SVM are used to identify very slight deviations from normal behavior of systems where possible threats behave obscurely, thus escaping detection by traditional methods.

Adaptability: unlike the static systems, machine learning models keep evolving from training on new data and hence adapt to the rapidly changing threat landscapes.

Proactive Defense: shifting one's interest from reactive defense toward proactive threat mitigation by employing ML-powered IDS helps organizations in conceiving as well as detecting threats before really harmful incidences occur.

This transformation is particularly relevant in an era where cyberattacks are not only more frequent but also more complex. Threat actors now employ techniques such as polymorphic malware, which changes its code to evade traditional detection methods, and advanced social engineering tactics that exploit human vulnerabilities. Machine learning provides the

necessary tools to counter these challenges by identifying anomalies that deviate from established norms and predicting potential attack vectors before they materialize.

In this project, we aim to harness the power of machine learning to enhance intrusion detection capabilities. By implementing algorithms such as Isolation Forest Classifier, and One-Class SVM, we strive to develop a system that is robust, scalable, and efficient. These models allow for diverse approaches to detecting intrusions, from identifying anomalies in unlabeled data to classifying network traffic into distinct categories.

Furthermore, machine learning's ability to handle the dynamic nature of cyber threats is critical for modern IDS solutions. Unlike static rule-based systems, ML models continuously learn and improve their accuracy by analyzing incoming data. This makes them particularly well-suited for environments with rapidly evolving threats, such as those posed by zero-day exploits or advanced persistent threats.

By building upon these advancements, this project not only seeks to contribute to the growing body of knowledge in cybersecurity but also aims to develop practical, real-world solutions that address the shortcomings of traditional IDS. The integration of machine learning into IDS is not just a technological innovation—it represents a paradigm shift in the way we approach cybersecurity, enabling us to stay one step ahead in the ongoing battle against cybercrime.

This endeavor is driven by the urgent need for more intelligent and adaptive security mechanisms that can protect our digital ecosystems in an era of unprecedented connectivity and complexity. Through this work, we hope to illustrate the transformative potential of machine learning in safeguarding critical assets and ensuring the resilience of modern networks against the ever-evolving threat landscape.

1.2 Problem Statement

Intrusion Detection Systems (IDS) are integral to modern cybersecurity frameworks, acting as the first line of defense against unauthorized access and malicious activities within a network. However, traditional IDS face several critical challenges that limit their effectiveness, particularly in the face of rapidly evolving cyber threats and increasingly complex network environments. These challenges highlight the need for innovative solutions that can overcome the inherent limitations of conventional approaches.

1. Static Rule-Based Systems

Traditional IDS often rely on static, rule-based frameworks that depend on predefined signatures and rules to detect intrusions. While effective against known attack patterns, these systems falter when confronted with novel or zero-day threats. Attackers frequently modify their tactics, leveraging techniques such as polymorphic malware, which changes its code to evade detection. As a result, rule-based systems struggle to adapt, leaving networks vulnerable to sophisticated exploits. Furthermore, manually updating rules to address new threats is both time-consuming and resource-intensive, making these systems increasingly impractical in dynamic environments.

2. High False Positive Rates

Another significant drawback of traditional IDS is their tendency to produce high false positive rates. These systems often flag legitimate user activities as potential threats, leading to unnecessary alerts that burden security teams. The excessive noise generated by false positives not only diverts attention from genuine threats but also contributes to alert fatigue among cybersecurity professionals. Over time, this can undermine the efficiency and reliability of security operations, allowing critical threats to slip through unnoticed.

3. Scalability Issues

As networks continue to grow in size and complexity, traditional IDS struggle to process and analyze the vast volumes of data generated. Modern organizations operate in environments that span on-premises infrastructure, cloud platforms, and IoT devices, all of which produce large and diverse data streams. The inability of static systems to scale effectively in such environments results in slower response times and diminished detection accuracy. This scalability challenge is exacerbated by the increasing sophistication of cyberattacks, which demand real-time analysis and response capabilities that traditional IDS often lack.

Addressing these challenges requires a fundamental paradigm shift in the design and implementation of IDS. The integration of machine learning techniques offers a promising solution by introducing adaptability, efficiency, and accuracy to intrusion detection. Unlike static rule-based systems, machine learning-powered IDS can dynamically learn from data, identify emerging attack patterns, and continuously improve their performance.

Our project seeks to address these limitations by leveraging advanced machine learning algorithms to enhance the capabilities of IDS. Specifically, we incorporate techniques such as Isolation Forest Classifier, and One-Class SVM to build a system that is robust, scalable, and capable of detecting both known and unknown threats. These algorithms enable the system to identify anomalies, classify network traffic, and adapt to changing threat landscapes with greater precision. By shifting from a reactive to a proactive approach, our project aims to create an IDS that can keep pace with the demands of modern cybersecurity.

1.3 Objectives of the Study

1. Design and build a Machine Learning based IDS: The study introduces the design and building of an intrusion detection system that uses machine learning algorithms for anomaly detection. This includes the development of the whole system using Python with its libraries, Scikit-learn, NumPy, that processes and identifies anomalies from network data as legitimate or maliciously associated activities.

2. Improve Detection Accuracy and Reduce False Positives: The effort is primarily focused on achieving less false positives without compromising the detection rates. The efforts are undertaken on different algorithms such as Isolation Forest, and One-Class SVM to measure the success of the project that defines the pros and cons of varying methods.

3. Performance Evaluation with Real-World Datasets the Evaluation system shall be evaluated on various activities such as NSL-KDD dataset to analyze different network behavior and attacks. Evaluate parameters of accuracy, precision, recall, and F1 to analyze the performance and credibility of the entire system.

4. Develop a Scalable and Adaptable Framework The scalable framework will handle processing thousands of records over the whole system. It will adapt swiftly and efficiently and will perform well when the demand increases. It will use feature engineering and

dimensionality reduction techniques to ensure performance, even for very complex data sets. The system is also capable of processing data in real-time so that threats can be detected quickly.

5. Future Recommendations and Real-time deployment. Future improvements incorporated into this would be as follows: real-time monitoring integration into the system, deep learning approaches implementation, and the latest datasets such as CIC-IDS2017 to keep the system updated with current issues in cybersecurity.

1.4 Organization of the Document

This document is designed to offer a detailed and structured account of the entire research and development process undertaken to build and evaluate a machine learning-powered Intrusion Detection System (IDS). Each section of the document provides valuable insights into the different phases of the project, from the initial background research to the final conclusions and suggestions for future advancements. The organization of the document is as follows:

Introduction

The Introduction serves as the starting point for understanding the motivation and objectives of this project. It provides a background on the significance of Intrusion Detection Systems (IDS) in modern cybersecurity, outlining the challenges faced by traditional IDS and the potential for machine learning techniques to address these limitations. The section also highlights the growing importance of cybersecurity in today's increasingly interconnected world and sets the stage for the specific research questions and goals of the study. This section concludes by presenting the overarching objectives and expected outcomes of the project, ensuring that readers have a clear understanding of what the research aims to accomplish.

Literature Review

The literature review examines previous intrusions detection systems and brings in machine learning integrated research within cybersecurity. This literature survey is a state-of-the-art survey of techniques in intrusion detection that surveys the history of IDS, which has evolved from rule-based systems to nowadays machine learning techniques. The advantages over and weaknesses of conventional IDS methods are listed in this section along with the justification

for the proposed inclusion of machine learning models. This review also covers machine learning algorithms as applied to anomaly detection and classifying such as Isolation Forest and One-Class SVM. Therefore, this section creates an important pillar in the establishment of methodology and approaches for this project by synthesizing previous studies.

Methodology

The steps that we observed in creating the IDS are elucidated here in a stepwise manner, in the Methodology section. This further explains to the readers the collection, preprocessing, and preparation procedures for data that were devoted to training and testing machine learning algorithms. The methodology includes selections of algorithms; namely Isolation Forest and One-Class SVM, along with proper reasoning for selecting these two models, keeping in mind their successful applicability in the field of anomaly detection in cybersecurity applications. Further, the process of splitting dataset samples into training, testing, and validation sets is described to ensure empirical evaluations of models to be fairly and accurately interpreted. Evaluation methodology adopted for each model will also be discussed in terms of the key metrics such as precision, recall, accuracy, and F1 score. The above account will give a comprehensive factorization of our research methodology, ensuring that the reader properly comprehends the whole process of development, along with the reasons for each aspect concerning contemporary cybersecurity needs.

Conclusion and Future Work

This section on Conclusion and Future Work summarizes all the main findings of the research along with all its success and failures in developing an ideal IDS. It enunciates One-Class SVM and Isolation Forest encompassing ways to enhance the performance of an IDS. This is done in terms of anomaly detection and the ability to adapt to new attack patterns. Considering this aspect, the conclusion presents future extensions of such an implementation in cyberspace and shows improvements in resilience and scalability by incorporating such machine learning techniques within intrusion detection systems. Besides this, the future developments are also mentioned to include other areas which need future development, such as hyperparameter optimization for One-Class SVM and Isolation Forest, exploring hybrid approaches to combine their strengths, and testing the system on more diverse and modern datasets like CIC-IDS2017. All of these aspects are presented by emphasizing future possibilities with this section to facilitate the continued exploration of the anomaly detection area for cybersecurity.

Appendices

Finally, the Appendices section includes supplementary materials that support the content of the document. These materials are not critical to the main body of the text but offer additional details for those who wish to delve deeper into the technical aspects of the project. For example, this section contains code snippets used to implement the machine learning models, dataset descriptions, and other technical documents that provide insight into the programming and data-handling procedures employed in the project. By including these appendices, the document ensures that readers have access to the resources needed to replicate or extend the work presented in the study.

References

The References section provides a comprehensive list of all the academic papers, books, articles, and technical resources that were consulted during the course of the project. These references support the claims made throughout the document and provide readers with the opportunity to explore the foundational research that guided the development of the machine learning models used in this study. Proper citation ensures that the work draws upon credible sources and adheres to academic standards.

2. Literature Review

2.1 Intrusion Detection Systems (IDS)

2.1.1 Definition

Intrusion Detection Systems (IDS) have become an indispensable part of modern cybersecurity strategies, playing a pivotal role in safeguarding digital infrastructures from unauthorized access, cyber-attacks, and other malicious activities. An IDS is essentially a security mechanism designed to monitor network traffic and system activities to detect potential security breaches, violations, or threats. Its main function is to identify anomalous

or unauthorized actions within a computer system, network, or application environment that could jeopardize the confidentiality, integrity, or availability of data. The importance of these systems continues to grow as cyber threats become more sophisticated and complex, posing significant risks to the security of critical infrastructure [1]¹.

At a fundamental level, an IDS operates by analyzing network or system activities, looking for patterns that may indicate malicious or abnormal behavior. These systems rely on predefined sets of rules or signatures that identify known attack patterns or behaviors that pose a risk to the system. Whenever the IDS identifies an activity that deviates from normal behavior or matches a predefined threat signature, it generates an alert, which is typically reviewed by a system administrator or security analyst to determine whether it represents an actual security breach. In some cases, the IDS may be configured to take automatic action in response to specific threats, such as isolating the affected system or blocking malicious traffic [2]².

Early intrusion detection systems were primarily rule-based, operating on a set of defined rules or signatures that allowed them to detect only known types of attacks. These rule-based systems were effective at identifying attacks that had been previously observed and documented, such as certain types of malware or unauthorized access attempts. However, they faced significant limitations when it came to detecting novel or unknown attacks. New types of malware, zero-day vulnerabilities, and evolving attack strategies could easily bypass these systems, making them less effective in dynamic and evolving threat landscapes [3]³. This inherent weakness led to the development of more sophisticated IDS systems that could learn from data, adapt to new and emerging threats, and offer more accurate and robust detection capabilities. Machine learning (ML) and artificial intelligence (AI) have

¹ [1] Bace, R. G., & Mell, P. (2001). Intrusion detection systems.

² [2] Singh, A. P., & Singh, M. D. (2014). Analysis of host-based and network-based intrusion detection system. *International Journal of Computer Network and Information Security*, 6(8), 41–47.

³ [3] Thakur, K., & Pathan, A. S. K. (2020). *Cybersecurity fundamentals: A real-world perspective*. CRC Press.

significantly enhanced the effectiveness of modern IDS solutions by enabling them to continuously improve their detection capabilities through experience.

In its current form, modern IDS technology has evolved beyond simple signature-based systems. Newer approaches, including those that leverage machine learning (ML), have allowed for the detection of both known and unknown threats. Machine learning techniques enable IDS to continuously learn from new data, adapt to emerging attack vectors, and improve their ability to identify sophisticated and previously unseen threats. These advanced systems are increasingly capable of identifying suspicious behavior and activities that may not match traditional attack patterns but still indicate malicious intent [4]⁴. This shift toward more adaptive, dynamic, and proactive detection systems represents a significant leap forward in the field of intrusion detection and is crucial in combating the growing sophistication of cyber threats.

2.1.2 Types of IDS

Intrusion Detection Systems (IDS) are typically classified into two major categories: Host-Based Intrusion Detection Systems (HIDS) and Network-Based Intrusion Detection Systems (NIDS). Each type serves different purposes and provides distinct advantages and disadvantages, depending on the security needs and the environment in which they are deployed. In recent years, hybrid approaches that combine elements of both HIDS and NIDS have gained popularity, offering a more comprehensive security solution that can address the complexity of modern cyber threats.

Host-Based IDS (HIDS):

A Host-Based Intrusion Detection System (HIDS) is installed directly on an individual host, such as a server, desktop, or laptop, and monitors the activities occurring within that specific host. HIDS primarily focuses on detecting malicious activities that originate within the host or target the host directly, such as unauthorized login attempts, file modifications, system call anomalies, and changes to system configurations. Unlike NIDS, which analyzes traffic on the network, HIDS focuses on monitoring the host's internal activities and its interactions with other devices or systems [5]⁵.

⁴ [4] Paffenroth, R. C., & Zhou, C. (2019). Modern machine learning for cyber-defense and distributed denial-of-service attacks. *IEEE Engineering Management Review*, 47(4), 80–85.

⁵ [5] Karatas, G., Demir, O., & Sahingoz, O. K. (2018, December). Deep learning in intrusion detection systems. In *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)* (pp. 113–116). IEEE.

HIDS typically analyzes system logs, file integrity, and process behavior to detect anomalies or signs of malicious activity. For example, an IDS could detect unusual login patterns, attempts to escalate privileges, or modifications to critical system files that could indicate an attempted breach. OSSEC is a widely used example of a Host-Based IDS that provides real-time log analysis, file integrity checking, rootkit detection, and other advanced monitoring features. HIDS is particularly useful for detecting attacks that originate from within the host or those that attempt to exploit vulnerabilities in the host system.

However, HIDS also has limitations. Because it is installed on individual hosts, it may not be effective at detecting attacks that occur on the network level, such as Distributed Denial of Service (DDoS) attacks or network scans. Additionally, a compromised host may also lead to the IDS being tampered with or bypassed. Despite these drawbacks, HIDS remains an important tool in cybersecurity, particularly for environments where hosts are critical assets that must be closely monitored.

Network-Based IDS (NIDS):

In contrast to HIDS, a Network-Based Intrusion Detection System (NIDS) operates by monitoring network traffic to detect malicious activities or policy violations. NIDS is typically deployed at strategic points in the network, such as on the network perimeter or in the core network infrastructure, to monitor all inbound and outbound traffic. By analyzing the data packets that travel across the network, NIDS aims to identify patterns that match known attack signatures or suspicious behavior indicative of a potential threat [6]⁶.

NIDS is effective in detecting network-level attacks, such as DDoS attacks, port scans, and attempts to exploit network vulnerabilities. Popular NIDS tools, such as Snort, use signature-based detection to identify known attack patterns and generate alerts when such patterns are observed in the network traffic. Snort is widely used due to its flexibility, ease of use, and ability to detect a broad range of network attacks in real time. However, NIDS can be less effective at identifying attacks within encrypted traffic or attacks that attempt to blend in with legitimate network traffic, making it more challenging to detect advanced, stealthy attacks.

⁶ [6] Sharafaldin, I., Gharib, A., Lashkari, A. H., & Ghorbani, A. A. (2018). Towards a reliable intrusion detection benchmark dataset. *Software Networking*, 2018(1), 177–200.

The primary advantage of NIDS is its ability to monitor network-wide traffic, making it suitable for detecting attacks that span multiple devices or systems. However, its reliance on signature-based detection means that it may struggle to detect novel or zero-day attacks that have not been previously identified.

2.2 Machine Learning in Cybersecurity

2.2.1 Overview of Machine Learning Techniques

In the rapidly evolving landscape of cybersecurity, traditional security measures, such as rule-based Intrusion Detection Systems (IDS), have struggled to keep pace with the increasing complexity and diversity of cyber-attacks. As cyber threats continue to grow in sophistication, attackers develop new strategies and techniques that bypass signature-based detection systems. This is where machine learning (ML) has emerged as a transformative force in the cybersecurity domain, offering the ability to learn from vast datasets, detect previously unknown threats, and adapt to new and evolving attack patterns. Machine learning has proven to be a game-changer by empowering IDS systems to identify subtle anomalies and attack behaviors that would have otherwise gone undetected by traditional methods [8]⁷.

Machine learning techniques, when applied to intrusion detection, can analyze patterns within large volumes of data to identify suspicious activities, making them more effective at detecting both known and unknown threats. This capability is particularly crucial as cyber-attacks become increasingly complex, targeting not only vulnerabilities in software or hardware but also leveraging social engineering, zero-day exploits, and other sophisticated tactics. By leveraging the power of machine learning, modern IDS can detect anomalous behavior by identifying patterns that deviate from established norms, and can continuously evolve to recognize new types of threats without the need for manual intervention.

⁷ [8] Shyaa, M. A., Ibrahim, N. F., Zainol, Z., Abdullah, R., Anbar, M., & Alzubaidi, L. (2024). Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion detection systems. *Engineering Applications of Artificial Intelligence*, 137, 109143.

Machine learning can be broadly categorized into two primary types based on how it learns from data: supervised learning, unsupervised learning. Each of these methods has its own strengths, limitations, and applications within the context of intrusion detection.

Supervised Learning:

This type of machine learning is perhaps the most widely used technique during the entire life cycle of an intrusion detection system. The data fed to an algorithm in supervised learning comes from labeled datasets, where attack data is defined together with normal (benign) data, thus enabling the model to then learn classifying incoming data into either one of those categories on the basis of patterns developed during training phase. Examples of supervised learning algorithms used for intrusion detection include Support Vector Machines (SVM). Such algorithms are very efficient systems that are present in environments where the labeled data is plentiful and structured, becoming extremely good at catching known types of attacks [9]⁸.

Of the different methods used in machine learning, supervised learning has been found to be very effective in detecting not unknown but known signatures and behaviors of attacks, making it an ideal candidate for identifying previously encountered threats. When battle techniques such as distributed denial of service (DDOS) attacks have been seen previously, it is labeled as seen and the system is trained to know these attack techniques the next time they recur. Supervised learning, on the other hand, tends to have a serious weakness it heavily relies on the availability of labeled datasets, which can be a cumbersome exercise to get them. Again, supervised modeling fails to work for new attacks never seen by the system unless the system is retrained on its updated data. In this way, it turns out that this kind becomes an ineffective downer if new or evolving patterns of attack come through, missing detection or becoming a false negative.

Unsupervised Learning:

Unsupervised learning is an unmonitored type of learning, as opposed to the supervised paradigm of allowing labels. In this case, it is all about letting the system learn the patterns and anomalies in the data without a priori knowing what it considered as an attack. The objective of the algorithms for unsupervised learning is to catch the abnormal behavior-and

⁸ [9] Paul, J. (2024). Comparative analysis of supervised vs. unsupervised learning in API threat detection.

material-an outcome of values-of a certain type that does not correspond to the acceptable subsequently.

One of the most significant unsupervised detection algorithms for intrusion detection is Isolation Forest. The method isolates anomalies in the data recursively by partitioning them. It is effective in detecting outlier condemnation and is the best within an indoor setting where the data high dimensionality is concerned. Another popular unsupervised algorithm is the One-Class SVM, which builds a classifier around normal data points and considers all points outside the boundary as anomalous. Thus, it is very useful in cases where the data is skewed or there is no available malicious data for training purposes.

Unsupervised learning technique is expected to be beneficial in discovering an attack vector or completely new techniques that cannot be recognized by signature-based detection. Because it does not specifically distinguish, legitimate from illegitimate activity, this technique generally great increase found in the false positive rates. This may produce a high volume of alerts, that overwhelms these security teams or mislabels benign behavior as malicious.

Discrimination was a formal process between an activity of illegitimate nature and among that which is legal, however, in this case, an increase in false-positive rates has been observed because of application of unsupervised learning methods. This can create a potentially very large number of alerts that could both overwhelm security teams and mislabel benign behavior as malicious.

By combining both supervised and unsupervised learning techniques, IDS can achieve a balanced performance; it is ensured that visible threats are detected with the same precision and that newly developed threats are detected through anomaly detection. This project aims to develop intrusion detection systems with enhanced capabilities using unsupervised learning techniques, i.e., One-Class SVM and Isolation Forest, which are robust defenses against modern cybersecurity threats. [10]⁹.

⁹ [10] Zakariah, M., AlQahtani, S. A., Alawwad, A. M., & Alotaibi, A. A. (2023). Intrusion detection system with customized machine learning techniques for NSL-KDD dataset. *Computers, Materials & Continua*, 77(3).

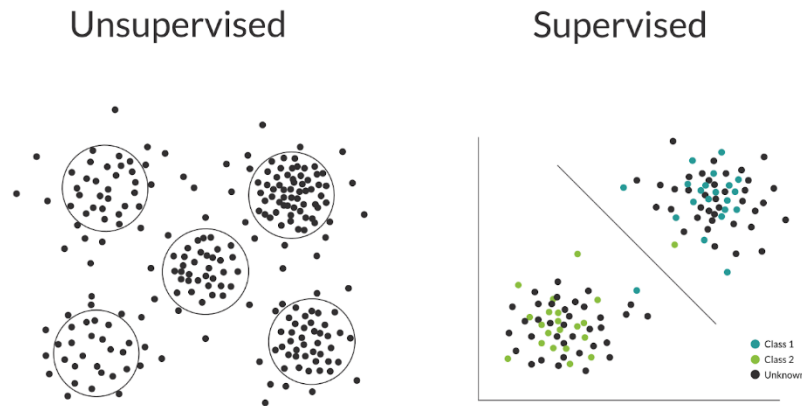


Figure 2: Supervised vs. Unsupervised Learning in IDS

2.2.2 Datasets Used in Cybersecurity

The development and evaluation of machine learning models for intrusion detection depend heavily on the availability of high-quality datasets. A reliable dataset serves as the foundation for training and testing machine learning algorithms, enabling the evaluation of their performance in realistic, real-world environments. Various benchmark datasets have been created over the years to facilitate the development of IDS systems and to compare the performance of different machine learning models. These datasets typically consist of network traffic data, where both benign and malicious activities are labeled, allowing algorithms to learn the characteristics of normal behavior and detect deviations that could indicate a potential security threat.

One of the most widely recognized and frequently used datasets in intrusion detection research is the KDD Cup 1999 dataset. This dataset, originally created for a data mining competition, contains network traffic data labeled with attack types such as Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and probing attacks. Despite its widespread use, the KDD Cup 1999 dataset has received criticism for its limitations, including the presence of redundant records and the overrepresentation of certain attack types, which may bias machine learning models. These issues have led to the creation of more refined datasets designed to address these shortcomings [11]¹⁰.

¹⁰ [11] Steingartner, W., Galinec, D., & Kozina, A. (2021). Threat defense: Cyber deception approach and education for resilience in hybrid threats model. *Symmetry*, 13(4), 597.

The NSL-KDD dataset was introduced as an improvement over the KDD Cup 1999 dataset, eliminating redundant records and ensuring a more balanced distribution of attack types. This dataset has become a widely accepted standard for evaluating machine learning-based IDS, particularly in academic and research settings. In a comprehensive evaluation conducted by Sharafaldin et al. (2018), the NSL-KDD dataset was shown to be an effective benchmark for training and testing IDS models, providing a more accurate representation of real-world network traffic and attack scenarios.

Another valuable resource in IDS research is the CICIDS dataset, developed by the Canadian Institute for Cybersecurity. The CICIDS dataset includes network traffic data from modern attack scenarios, making it highly relevant for the detection of contemporary threats, such as Advanced Persistent Threats (APTs) and zero-day attacks. The CICIDS dataset offers a diverse range of attack types, including DDoS, malware infections, and botnet activities, providing a comprehensive dataset for evaluating the performance of machine learning models against a broad spectrum of modern cyber threats.

The availability of high-quality datasets like KDD Cup, NSL-KDD, and CICIDS plays a crucial role in advancing the development of machine learning models for intrusion detection. These datasets not only provide a means to train and validate algorithms but also serve as benchmarks for comparing the performance of different models and techniques. As the cybersecurity landscape continues to evolve, the continued development of new and diverse datasets will be essential for ensuring that IDS systems can effectively detect emerging threats and safeguard critical infrastructure.

In the following section, we will delve deeper into how the integration of machine learning techniques into IDS can enhance detection capabilities and improve overall system performance. We will also examine the challenges associated with training machine learning models for intrusion detection, including issues related to dataset quality, feature selection, and model interpretability.

2.3 Anomaly Detection Techniques

In the region of Intrusion Detection Systems (IDS), anomaly detection is an essential tool that alerts a possible detection of any security breaches, an attack, or suspicious activity within a network or system. Anomaly detection highlights deviations between actual behavior or traffic from what has been algorithmically defined as 'normal' or baseline behavior or traffic. If these deviations go unnoticed, it is most likely that an attack is already in progress or that malicious actors are present. With the modernity of cyber threats growing increasingly sophisticated or evasive, the detection of abnormal behavior becomes critical, particularly with regard to unseen or zero-day attacks.

This generally covers the monitoring and evaluation of network traffic in IDS as well as system logs or any other data sources in anomaly detection for attack pattern identification. Anomaly detection differs from signature-based detection since the latter compares features based on predefined attack patterns. However, the former bases its assessment on marked deviations from established baselines and thus is capable of detecting novel polymorphic attacks, which have no signatures in databases. Thus, anomaly detection proves to be one of the arguments for dealing with present-day cybersecurity challenges.

Anomaly detection methods can be classified into two broad categorizations: statistical and machine learning-based approaches. Statistical methods are simple yet easy to implement, while machine learning approaches extend more flexibility and adaptation to evolving adynamic cyber threats.

2.3.1 Statistical Methods in Anomaly Detection

Statistical methods are widely considered one of the first approaches employed in IDS and are set on an assumption that most data in a system follows predictable patterns or distributions. These methods will create a baseline depicting "normal" behavior through historic data and any data point that deviates from such baseline from then on will be flagged as anomalous. For example, they tend to employ probability distributions in characterizing normal behavior using distributions such as Gaussian or Poisson while flagging data points that fall outside the expected range. Another approach is labeled outlier detection, in which any data point lying far from the mean or within predetermined thresholds is considered suspicious. The major drawback of using statistical methods is that it is quite straightforward and works efficiently to predict behavior, but it comes with serious limitations.

One would be the difficulty in setting appropriate thresholds in large or dynamic systems, in addition, modern networks contain high-dimensional datasets that prove to be a headache for legal statistical methodologies. With such simplicity, however, statistical methods fail to detect novel attacks that do not conform to typical statistical distributions. Nevertheless, these methods are extremely popular in applications where behaviors are comparatively simple or predictable as they are easy to implement.

2.3.2 Machine Learning-Based Anomaly Detection

Machine learning-based anomaly detection has gained considerable attention in recent years, as it provides a much more flexible and adaptive approach compared to traditional statistical methods. Machine learning models are capable of learning complex patterns and relationships within large datasets, enabling them to detect anomalies more accurately and with a greater degree of adaptability. This is particularly important in cybersecurity, where threats evolve continuously, and new attack vectors are constantly being developed.

In machine learning-based anomaly detection, there are a variety of algorithms that can be applied, each designed to uncover different types of irregularities in data. These algorithms can be broadly classified into supervised and unsupervised learning methods, depending on whether the algorithm requires labeled data (where normal and attack behaviors are explicitly marked) or if it works with unlabeled data.

2.3.2.1 Unsupervised Learning Algorithms

Unsupervised learning approaches play a crucial role in anomaly detection, particularly in scenarios where labeled data is either unavailable or insufficient. These methods enable models to autonomously identify patterns and deviations in data, without predefined knowledge of what constitutes an attack. This makes unsupervised learning particularly effective in detecting novel and previously unseen threats, which are prevalent in the ever-evolving field of cybersecurity.

In this project, **Isolation Forest** and **One-Class SVM** are the primary unsupervised learning algorithms selected to enhance anomaly detection in network traffic. These algorithms provide robust solutions for identifying anomalies in unlabeled data while addressing modern cybersecurity challenges.

Isolation Forest: This algorithm isolates anomalies by recursively partitioning data into smaller subsets. Anomalies, being rare and distinct, are separated more easily compared to normal data points. The algorithm demonstrates strong reliability in handling high-

dimensional datasets and large-scale data, making it particularly effective in detecting anomalies within complex network environments. Isolation Forest is especially well-suited for scenarios requiring the identification of subtle deviations from normal behavior.

One-Class SVM: This algorithm models normal behavior by defining a boundary that encompasses the majority of normal data points. Any point falling outside this boundary is flagged as anomalous. One-Class SVM is particularly valuable in situations where malicious data is rare, as it reliably detects deviations from established norms. It has been widely applied in network security to identify new and emerging attack vectors that deviate from existing patterns.

These algorithms serve as the foundation of the intrusion detection system developed in this project. By leveraging their capabilities, the system aims to identify anomalies effectively in unlabeled data and address the sophisticated challenges posed by modern cybersecurity threats.

2.3.2.2 Supervised Learning Algorithms

Approaches in Supervised Learning (Not Included for the Purpose of This Project) Although supervised learning methods have found extensive application in many intrusion detection systems, the present work does not include them in its efforts since this work has emphasis on discovering anomalies in unlabeled data. Supervised models learn from training data as regards normal and abnormal input. It shows highly efficiency towards known attacks, but fails to cater to new attacks since this approach updating new labeled data.

Albeit effective in attacking known threats, the very popular supervised learning algorithms such as Decision Trees and Random Forest fail to yield well in terms of new and emerging threats, as they become more dynamic. This shows why supervised learning is unsuitable in this project's aim of finding previously unknown attack vectors.

2.3.2.3 Hybrid Approaches

In practice, hybrid approaches that combine both unsupervised and supervised learning techniques are often employed to take advantage of the strengths of each. For example, unsupervised learning techniques might first be used to identify potential anomalies, which are then analyzed further using supervised learning methods to classify whether the anomalies are indeed attacks. By combining these two methods, IDS systems can be more effective at detecting both known and unknown threats.

Why One-Class Algorithms?

The purpose of selecting One-Class algorithms like One-Class SVM and Isolation Forest is because they prove very effective in anomaly detection, mainly in cases where there are very few or no labeled malicious data present. These algorithms are designed in a manner that focuses on normality and deviation detection; hence, they will perfectly fit cases of imbalanced datasets such as those found in cybersecurity contexts. One-Class SVM produces impressive results particularly when the normal model is very specific and thus anomaly detection is achieved with few false alarms. On the other hand, Isolation Forest works excellently with very high dimensional datasets and employs recursive partitioning to isolate anomalies in a very fine manner. This gives the algorithms an added feature of detecting zero-day attacks and adapting to connected networks' continuously changing environments; hence makes them robust for intrusion detection.

2.3.3 Advantages of Machine Learning-Based Anomaly Detection

The main advantage of using machine learning-based anomaly detection techniques over traditional statistical methods lies in their adaptability and scalability. Machine learning models can automatically adjust to new data, allowing them to detect novel attack patterns that have not been seen before. This is crucial for modern IDS, as attackers continually evolve their tactics to evade detection. Furthermore, machine learning techniques are capable of handling large, complex datasets and can identify subtle patterns in data that may not be apparent using simple statistical methods.

Additionally, machine learning models can be trained to reduce false positive rates, an issue that is commonly associated with anomaly detection. By learning from a larger variety of data, these models can become more accurate over time, improving their ability to distinguish between legitimate anomalies and benign deviations.

2.3.4 Challenges and Future Directions

Despite the many advantages of machine learning-based anomaly detection, there are still several challenges that need to be addressed. One of the primary concerns is the quality and quantity of the data used for training the models. Anomalies are often rare events, making it difficult to acquire sufficient labeled data for supervised learning models. In unsupervised learning, the difficulty of detecting anomalies increases in high-dimensional data where noise can overwhelm the signal. Furthermore, the interpretation of machine learning models

remains a challenge, as many of these models, especially deep learning techniques, are often considered "black boxes" with limited transparency.

As the field of cybersecurity continues to evolve, it is expected that further research into hybrid models, real-time processing, and interpretable machine learning will improve the effectiveness of anomaly detection systems. Advances in these areas will help mitigate the challenges faced by current anomaly detection techniques and provide more robust protection against emerging threats.

Methodology

3. 1 Dataset Description

NSL-KDD dataset It comes totally recommended by other state-of-the-art resources for evaluating intrusion detection systems. This dataset was very clean and well-balanced, without duplicates or redundancies of its predecessor, the KDD Cup 1999 dataset. Along with NSL-KDD, custom datasets were additionally included for testing flexibility and resilience of the system. These custom datasets comprise diverse features as well as attack scenarios for enriching evaluation of the system.

As most ND traffic features are based on intrusion detection systems, the NSL-KDD dataset has 41 features, which represent different dimensions of the network's traffic behavior. Experiments were approved concerning these features: Basic Features, Content Features, and Traffic Features.

Basic Features refer to common characteristics associated with network connection, such as protocol type, duration, and service. Example: Protocol_Type represents whether TCP or UDP was used in establishing the connection, while Duration states the duration of the connection.

The Content Features describe session-specific events, such as how many times a user failed to log in (Num_Failed_Logins) or whether created files or executed scripts were present (Hot). These features are most likely indicative of behaviors that correspond with malicious involvement.

Traffic Features refer to summated measures representing the behavior as measured in the network during a specified interval. Count-based: Number of connections per source for a

specific time interval; Same_Srv_Rate-the ratio of connections made toward the same service. The most useful in denoting behaviors like DDoS attacks.

The types of traffic have two broad classes: normal or abnormal. Abnormal traffic has been further categorized into four types of attack: Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L). Each of these categories represents its unique style of intrusion

3.2 Data preprocessing

All the huge preprocessing steps taken were to make the datasets trainable and testable. It identified all the missing or invalid values and filled them in or discarded incomplete record entries completely. The clean process guarantees the validity of data.

Continuous features were normalized within the range of 0 and 1 with Min-Max scaling. This brings features on the same footing so that no single feature dominantly influences the models. This conversion of categorical attributes like Protocol_Type and Service to numerical values using One-Hot Encoding makes the machine learning model to allow them for processing.

3.3 Model Development

The above-mentioned machine learning models would then be employed on network traffic for anomaly detection. They are One-Class Support Vector Machine (SVM) and Isolation Forest. These two models mainly address anomaly in high-dimensional data sets.

It is built by training only on normal traffic to classify deviations from this boundary as anomalous. Such modeling is well suited for environments where false positives have to be minimized within a well-defined data domain. This model was built on Scikit-learn, Python, and hyper-parameters were defined such as $\gamma=0.1$ and $\nu=0.05$.

Isolating the anomaly using recursive-partitioning. This unsupervised algorithm works very well in detecting anomalies for large and high dimensional data and also on zero day attacks. Isolation Forest was trained on Scikit-learn using parameters defined as: $n_estimators = 100$,

contamination=0.1 for easy comparison of performances with other future models on dynamically changing data.

3.3.1 One class Support Vector Machine (SVM)

The Support Vector Machine (SVM) is a supervised learning algorithm that classifies data through the optimal hyperplane. In this project, One-Class SVM variant is used for anomaly detection. This variant sets the boundary of normal behavior, and everything else is flagged as an anomaly.

Advantages:

It works well with datasets having a large number of dimensions.

Does not require large labeled datasets suitable for limited data scenario.

Implementation:

Built using the Python Scikit-learn library. normal (non-malicious) traffic. The algorithm learns the boundary of normal behavior through training on a dataset made up predominantly of normal (non-malicious) traffic. While testing, it considers points outside of it as anomalies.

3.3.2 Isolation Forest

An unsupervised learning algorithm known as Isolation Forest isolates anomalies for anomaly detection. It randomly splits different data points and isolates those which are way different from the rest.

Advantages: Able to process large quantities of data fast.

Can detect outliers in high dimensional datasets.

Unsupervised nature: Requires no labelled data, which makes it useful in finding new attack patterns.

Implementation:

Created using the Isolation Forest module of Scikit-learn. The algorithm keeps dividing a dataset to get rid of anomalies. Anomalies easy to isolate (i.e. needing fewer splits) are identified as anomalies.

3.4 Model Evaluation

The model assessment is aggravated in this project for assuring the models' viability of effectiveness and generalization for new data. Evaluation processes use many key collections of metrics and validation techniques to assess the ability of the model.

Evaluation metrics:

The performance of One-Class SVM and Isolation Forest models was evaluated against the metrics which illuminate the various aspects of anomaly detection.

Accuracy, measures the proportion of correctly classified normal and anomalous instances to the number of instances in the dataset. In totality, an overall picture is derived about the performance of the models, but it may mislead itself when it comes to imbalanced datasets where anomalies appear very infrequently.

Precision calculates how many of the predicted anomalies have been identified to be true anomalies. It is an important measure in a situation where false positives can initiate unwanted alerts or wasted resources.

Recall (Sensitivity) evaluates the actual true positive cases vis-a-vis the total actual cases of anomalies present in the dataset. It implies that high recall ensures most of the anomalies are detected, albeit more false positives might be generated.

F1-Score is the average of precision and recall in a harmonic mean. This is a very interesting metric to consider for anomaly detection, where one has to make his or her best

to minimize the number of false positives along with false negatives.

Validation Process:

Divided into three subsets for strong validation and unbiased evaluation completion.

Training Set: The data that is focused on training the models for their training so the models can learn normal behavior network traffic patterns. The data used in training was majorly normal traffic to create a clear baseline for that behavior.

Validation Set: For One-Class SVM, model parameters such as gamma and nu were critically estimated for Isolation Forest as `n_estimators` and `contamination`. This step was put into consideration to avoid underfitting and overfitting of the models.

Testing Set: Final evaluation and independent data set composed of the normal traffic and anomalous measurements. The models were tested with truly unseen data and thus giving an unbiased evaluation of their generalization abilities.

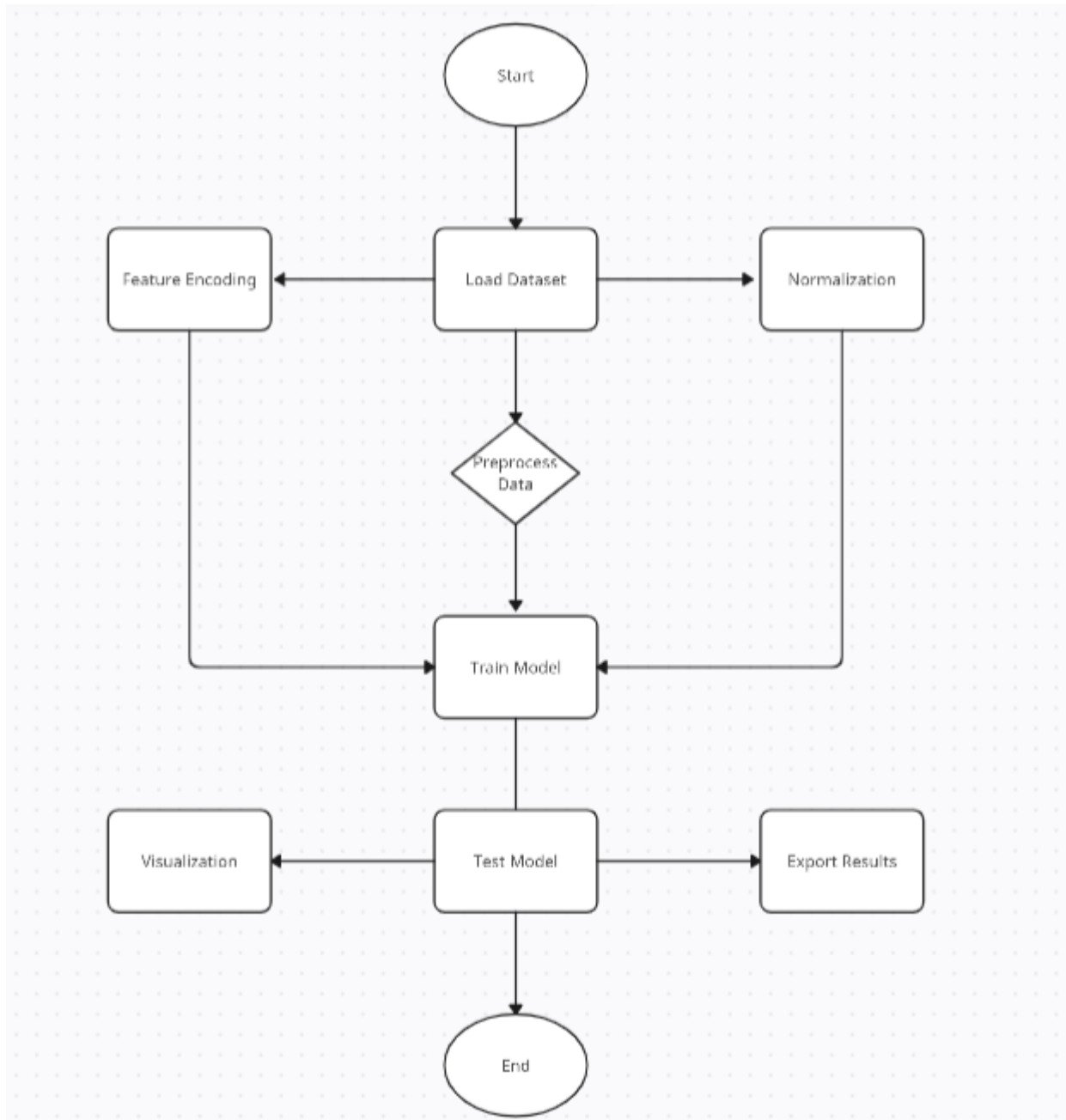


Figure 3 : Methodology

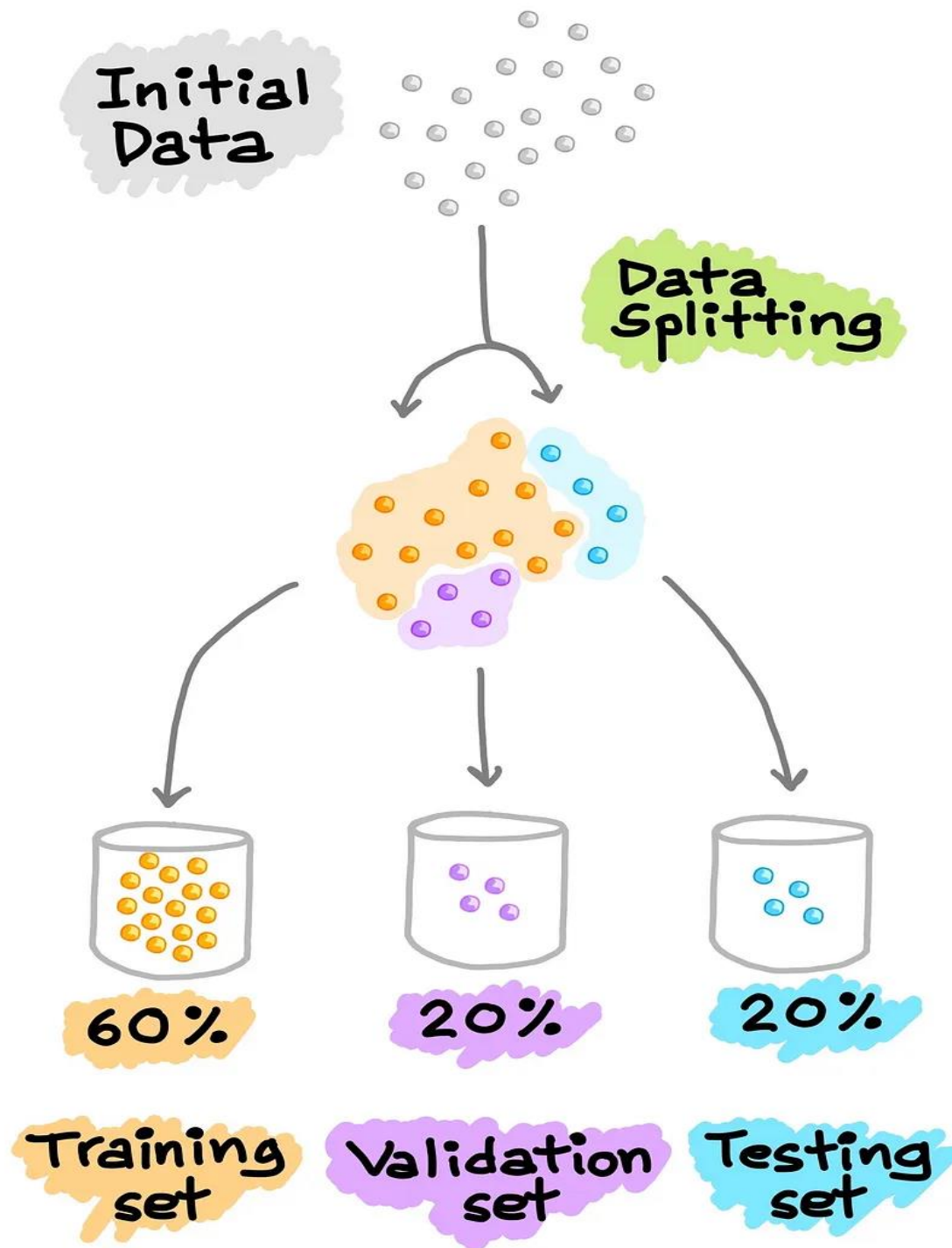


Figure 4 : Model Evaluation

Implementation

5.1 Tools

The implementation of this project was carried out using Python, supported by several essential libraries to handle machine learning, data processing, and visualization tasks:

- **Scikit-learn:** Used to implement machine learning algorithms such as One-Class SVM and Isolation Forest.
- **NumPy:** Facilitates efficient numerical computations and array manipulation.
- **Matplotlib:** Generates both static and interactive visualizations for analyzing results.
- **Tkinter:** Provides a framework for building the graphical user interface (GUI)

5.2 GUI Overview

The GUI was developed using Python's Tkinter library to offer a user-friendly interface that simplifies dataset management, model training, and results visualization. It includes features designed to make the system accessible for users with minimal technical expertise.

Step-by-Step Functionality

1. **Loading Datasets:** Users can upload datasets through the "Load Dataset" button. The interface also provides options to specify the percentages for splitting the data into training, testing, and validation sets.
2. **Selecting a Model:** A dropdown menu allows users to select between One-Class SVM and Isolation Forest models. Hyperparameters can be configured directly within the interface for customized training.
3. **Training and Testing:** By clicking the "Train Model" button, users can train the selected model on the uploaded dataset. Once training is complete, the GUI automatically tests the model and displays the results.
4. **Visualizing Results:**
 - An anomaly detection plot displays the distribution of normal and anomalous data points.
 - Evaluation metrics such as accuracy, precision, recall, and F1-Score are presented for in-depth performance analysis.

5. **Exporting Results:** Users can export the evaluation metrics and results into external files for further review and reporting purposes.

Anomaly Detection GUI

Load DatasetRun Model

Load Dataset

Load Dataset

Validate Dataset

Training %:
70

Testing %:
10

Validation %:
20

Go to Next

Anomaly Detection GUI

Load DatasetRun Model

Select and Run Model

One-Class SVM

Training Model

Training Results

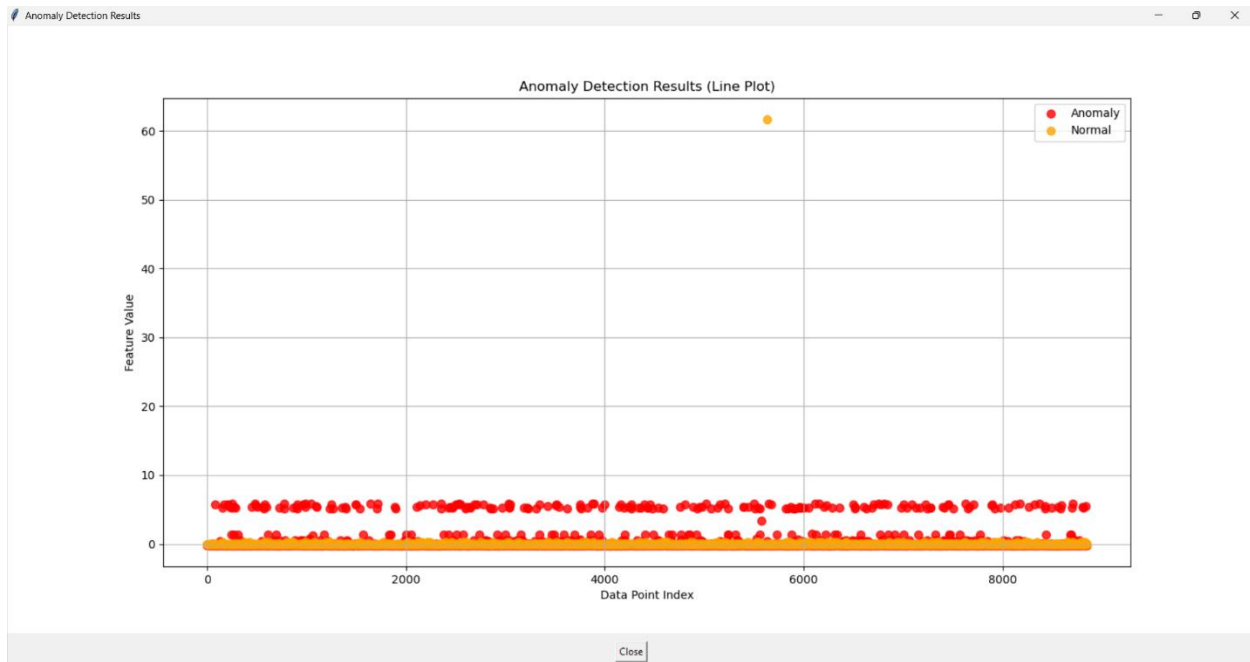
Load Test Dataset

Test Results

Evaluate Model

Export Results

Back to Page 1



Conclusion and Future Work

6.1 Summary of Results

This study can sort out the transformative potential that offers to machine learning techniques in the ways of anomalous detection for the entire network traffic scenarios. Through thoroughly evaluating both Isolation Forest and One-Class SVM, the study presents strengths and weaknesses garnering their differences across several scenarios.

Isolation Forest has been excellent in how it isolates outliers. This model has great applicability in large-scale data environments where the anomalies are subtle. Fast processing of a huge amount of data makes it compatible with very dynamic networks.

In contrast, One-Class SVM is powerful where normal behavior is defined. More than that, a very highly balanced or not too unbalanced dataset would give it good accuracy, which indicates suitability for use in conditions with very well-modeled normal traffic.

Those aspects refer to the inflexibility of modeling choice to the specific use case. Isolation forest can work with resilience and scalability in large, complex environments while One-

Class SVM can be considered for precision in datasets representing a perfect distinction of normal behavior.

The results reflect what should be considered adaptable and flexible in machine learning methods, which are high-performance tools for various applications in cybersecurity. They create the impetus for robust, scalable IDS that can proactively address modern attacks through their detection measures.

6.2 Future Directions

6.2 Proposed Future Directions

Future research in this area can broaden the practical implementation of the system on the following avenues:

Real-Time Deployment-Incorporating the system into real-time intrusion detection infrastructures in order to assess the effectiveness of the system within the live dynamic networking environment; Identifying its capability of detecting as well as stopping ongoing attacks in real time scenarios.

Validation on Modern Datasets:

Further validation tests must involve advanced and current datasets, such as CIC-IDS2017, in order to reveal adaptability, robustness, as well as performance of the system under different conditions.

Model Parameter Optimization:

Improving model performance through parameter tuning for an optimally accurate and efficient detection. Investigate advanced optimization techniques for further refinement of the models.

Exploration of Advanced Techniques:

Research and add advanced unsupervised learning techniques, like deep learning models (i.e., autoencoders, generative adversarial networks), to enhance the detection capability of the system.

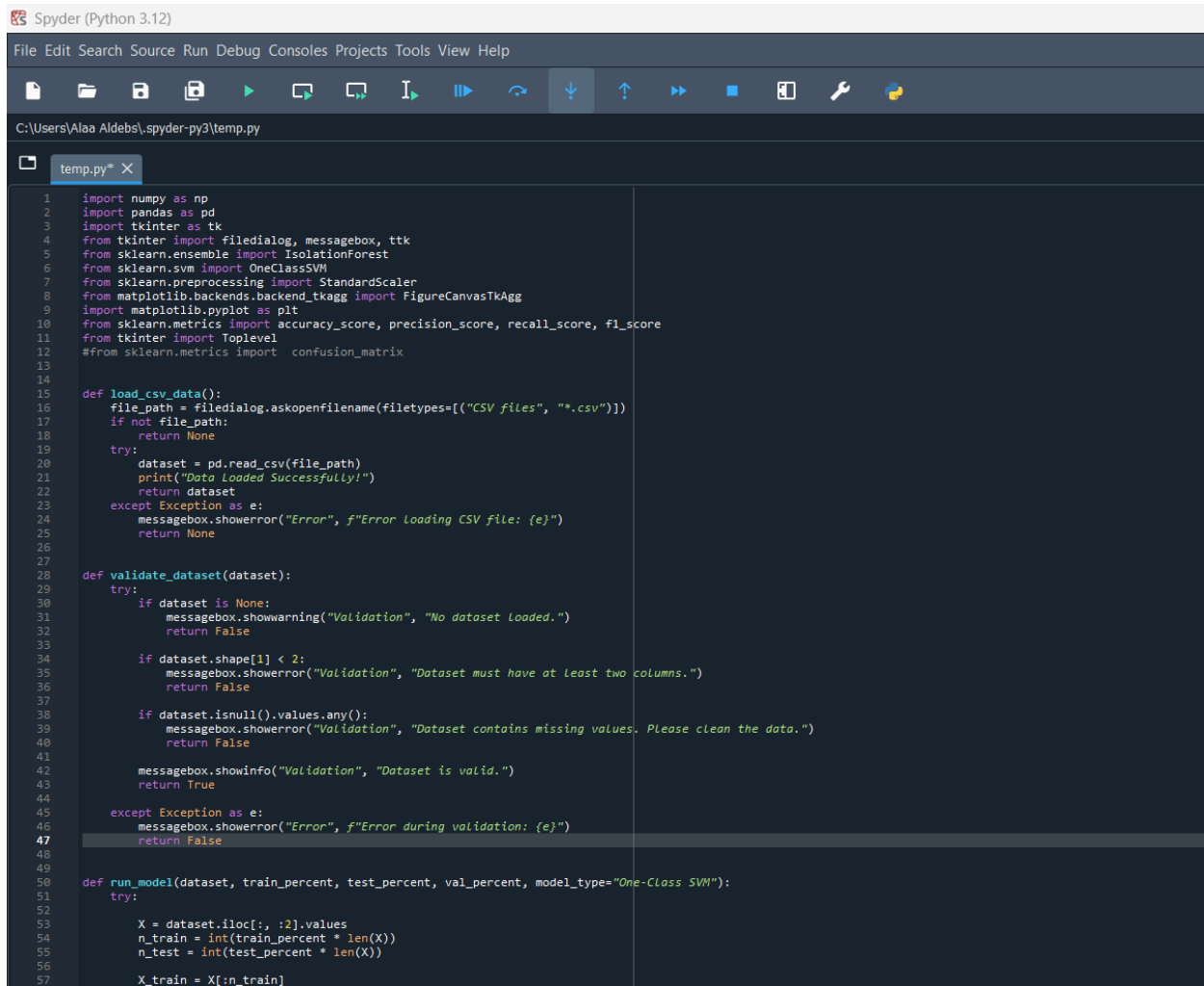
Knowledge Sharing:

Publish the results of this experiment in research papers for further scientific collaboration and forsyth in the field of machine learning and cybersecurity.

Facilitate open-source framework developments from the findings and thus pave the way for faster innovations in ICD technologies.

Appendix

7.2 Code Snippets and Screenshots



The screenshot displays the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. Below the menu is a toolbar with icons for file operations and execution. The status bar at the bottom shows the file path: C:\Users\Alaa Aldebs\spyder-py3\temp.py.

The main editor window shows a Python script named temp.py with the following code:

```
1 import numpy as np
2 import pandas as pd
3 import tkinter as tk
4 from tkinter import filedialog, messagebox, ttk
5 from sklearn.ensemble import IsolationForest
6 from sklearn.svm import OneClassSVM
7 from sklearn.preprocessing import StandardScaler
8 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
9 import matplotlib.pyplot as plt
10 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
11 from tkinter import Toplevel
12 #from sklearn.metrics import confusion_matrix
13
14
15 def load_csv_data():
16     file_path = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
17     if not file_path:
18         return None
19     try:
20         dataset = pd.read_csv(file_path)
21         print("Data Loaded Successfully!")
22         return dataset
23     except Exception as e:
24         messagebox.showerror("Error", f"Error Loading CSV file: {e}")
25         return None
26
27
28 def validate_dataset(dataset):
29     try:
30         if dataset is None:
31             messagebox.showwarning("Validation", "No dataset loaded.")
32             return False
33
34         if dataset.shape[1] < 2:
35             messagebox.showerror("Validation", "Dataset must have at least two columns.")
36             return False
37
38         if dataset.isnull().values.any():
39             messagebox.showerror("Validation", "Dataset contains missing values. Please clean the data.")
40             return False
41
42         messagebox.showinfo("Validation", "Dataset is valid.")
43         return True
44
45     except Exception as e:
46         messagebox.showerror("Error", f"Error during validation: {e}")
47         return False
48
49
50 def run_model(dataset, train_percent, test_percent, val_percent, model_type="One-Class SVM"):
51     try:
52
53         X = dataset.iloc[:, :2].values
54         n_train = int(train_percent * len(X))
55         n_test = int(test_percent * len(X))
56
57         X_train = X[:n_train]
```



```

54     n_train = int(train_percent * len(X))
55     n_test = int(test_percent * len(X))
56
57     X_train = X[n_train]
58     X_test = X[n_train:n_train + n_test]
59     X_val = X[n_train + n_test:]
60
61
62     scaler = StandardScaler()
63     X_train = scaler.fit_transform(X_train)
64     X_test = scaler.transform(X_test)
65     X_val = scaler.transform(X_val)
66     X_outliers = np.random.uniform(low=-4, high=4, size=(3, 2))
67
68
69     if model_type == "One-Class SVM":
70         model = OneClassSVM(kernel='rbf', gamma=0.1, nu=0.05)
71     elif model_type == "Isolation Forest":
72         model = IsolationForest(n_estimators=100, contamination=0.1, random_state=42)
73     else:
74         raise ValueError("Invalid model type selected.")
75
76     model.fit(X_train)
77
78
79     y_pred_train = model.predict(X_train)
80     y_pred_test = model.predict(X_test)
81     y_pred_val = model.predict(X_val)
82
83
84     y_pred_train = [1 if x == 1 else 0 for x in y_pred_train]
85     y_pred_test = [1 if x == 1 else 0 for x in y_pred_test]
86     y_pred_val = [1 if x == 1 else 0 for x in y_pred_val]
87     y_pred_outliers = [0 if x == -1 else 1 for x in model.predict(X_outliers)]
88
89
90     y_true_train = [1] * len(y_pred_train)
91     y_true_test = [1] * len(y_pred_test)
92     y_true_val = [1] * len(y_pred_val)
93     y_true_outliers = [0] * len(y_pred_outliers)
94
95
96     y_true = y_true_train + y_true_test + y_true_val + y_true_outliers
97     y_pred = y_pred_train + y_pred_test + y_pred_val + y_pred_outliers
98
99
100     accuracy = accuracy_score(y_true, y_pred)
101     precision = precision_score(y_true, y_pred, average='binary')
102     recall = recall_score(y_true, y_pred, average='binary')
103     f1 = f1_score(y_true, y_pred, average='binary')
104     # conf_matrix = confusion_matrix(y_true, y_pred)
105
106
107     result_message = (
108         f"Model: {model_type}\n\n"
109         f"Accuracy: {accuracy:.2f}\n"
110         f"Precision: {precision:.2f}\n"
111         f"Recall: {recall:.2f}\n"
112         f"F1-Score: {f1:.2f}\n\n"
113         f"Confusion Matrix:\n{conf_matrix}"
114     )

```

```

115     messagebox.showinfo("Model Results", result_message)
116
117 except Exception as e:
118     messagebox.showerror("Error", f"Error in model: {e}")
119
120 #####
121
122
123 class MultiPageApp(tk.Tk):
124
125     def __init__(self):
126         super().__init__()
127         self.title("Anomaly Detection GUI ")
128         self.geometry("800x600")
129         self.configure(bg="#2C3E50")
130
131         self.train_percent = 0.7
132         self.test_percent = 0.1
133         self.val_percent = 0.2
134         self.report = None
135
136         #for page tabe
137         notebook = ttk.Notebook(self)
138         notebook.pack(fill='both', expand=True)
139
140
141         self.page1 = Page1(notebook, self)
142         self.page2 = Page2(notebook, self)
143
144         notebook.add(self.page1, text="Load Dataset")
145         notebook.add(self.page2, text="Run Model")
146
147         self.pages = {"Page1": self.page1, "Page2": self.page2}
148         self.notebook = notebook
149
150     def show_page(self, page_name):
151         page = self.pages.get(page_name)
152         if page:
153             self.notebook.select(self.pages[page_name])
154
155 class Page1(tk.Frame):
156     def __init__(self, parent, controller):
157         super().__init__(parent, bg="#34495E")
158         self.controller = controller
159         self.dataset = None
160
161
162         label = tk.Label(self, text="Load Dataset", font=("Arial", 18), fg="white", bg="#34495E")
163         label.pack(pady=20)

```

```

165 load_button = tk.Button(self, text="Load Dataset", command=self.load_dataset,
166                          bg="#16A085", fg="white", font=("Arial", 14), activebackground="#148F77", bd=0, relief="flat")
167 load_button.pack(pady=10, fill=tk.X)
168
169 validate_button = tk.Button(self, text="Validate Dataset", command=self.validate_dataset,
170                             bg="#16A085", fg="white", font=("Arial", 14), activebackground="#148F77", bd=0, relief="flat")
171 validate_button.pack(pady=10, fill=tk.X)
172
173 #####
174 self.train_percent = tk.StringVar(value="70")
175 self.test_percent = tk.StringVar(value="10")
176 self.val_percent = tk.StringVar(value="20")
177
178 tk.Label(self, text="Training %", font=("Arial", 12), fg="white", bg="#34495E").pack(pady=5)
179 tk.Entry(self, textvariable=self.train_percent, font=("Arial", 12)).pack(pady=5)
180
181 tk.Label(self, text="Testing %", font=("Arial", 12), fg="white", bg="#34495E").pack(pady=5)
182 tk.Entry(self, textvariable=self.test_percent, font=("Arial", 12)).pack(pady=5)
183
184 tk.Label(self, text="Validation %", font=("Arial", 12), fg="white", bg="#34495E").pack(pady=5)
185 tk.Entry(self, textvariable=self.val_percent, font=("Arial", 12)).pack(pady=5)
186
187 #####
188 next_button = tk.Button(self, text="Go to Next", command=self.go_to_next_page,
189                         bg="#16A085", fg="white", font=("Arial", 14), activebackground="#148F77", bd=0, relief="flat")
190 next_button.pack(pady=20, fill=tk.X)
191
192
193 def load_dataset(self):
194     self.dataset = load_csv_data()
195
196 def validate_dataset(self):
197     validate_dataset(self.dataset)
198
199 def go_to_next_page(self):
200     try:
201         train = float(self.train_percent.get())
202         test = float(self.test_percent.get())
203         val = float(self.val_percent.get())
204
205         if train + test + val != 100:
206             messagebox.showerror("Error", "Percentages must add up to 100!")
207             return
208
209         self.controller.train_percent = train / 100
210         self.controller.test_percent = test / 100
211         self.controller.val_percent = val / 100
212         self.controller.show_page("Page2")
213     except ValueError:
214         messagebox.showerror("Error", "Please enter valid percentages.")
215
216
217 class Page2(tk.Frame):
218     def __init__(self, parent, controller):
219         super().__init__(parent, bg="#34495E")
220         self.controller = controller
221         self.report = None
222

```

```

223 label = tk.Label(self, text="Select and Run Model", font=("Arial", 18), fg="white", bg="#34495E")
224 label.pack(pady=20)
225
226 self.model_var = tk.StringVar(value="One-Class SVM")
227 model_options = ["Isolation Forest", "One-Class SVM"]
228
229 model_dropdown = tk.OptionMenu(self, self.model_var, model_options)
230 model_dropdown.config(bg="#34495E", fg="white", font=("Arial", 12), relief="flat")
231 model_dropdown.pack(pady=10)
232
233 training_button = tk.Button(
234     self, text="Training Model", command=self.train_model,
235     bg="#2ECC71", fg="white", font=("Arial", 14), activebackground="#27AE60", bd=0, relief="flat")
236 training_button.pack(pady=10, fill=tk.X)
237
238
239
240
241
242
243 run_button = tk.Button(
244     self, text="Training Results", command=self.run_model_and_show_results,
245     bg="#1F618D", fg="white", font=("Arial", 14), activebackground="#1A5276", bd=0, relief="flat")
246 run_button.pack(pady=20, fill=tk.X)
247
248 #####
249
250 #Buttons
251 run_button = tk.Button(
252     self, text="Load Test Dataset", command=self.load_test_dataset,
253     bg="#1F618D", fg="white", font=("Arial", 14), activebackground="#1A5276", bd=0, relief="flat")
254 run_button.pack(pady=20, fill=tk.X)
255
256 #Buttons
257 run_button = tk.Button(
258     self, text="Test Results", command=self.run_test_model,
259     bg="#1F618D", fg="white", font=("Arial", 14), activebackground="#1A5276", bd=0, relief="flat")
260 run_button.pack(pady=20, fill=tk.X)
261
262 #Buttons
263 run_button = tk.Button(
264     self, text="Evaluate Model", command=self.evaluate_model,
265     bg="#1F618D", fg="white", font=("Arial", 14), activebackground="#1A5276", bd=0, relief="flat")
266 run_button.pack(pady=20, fill=tk.X)
267
268
269
270
271
272
273
274
275
276 export_button = tk.Button(
277     self, text="Export Results", command=self.export_results,
278     bg="#035400", fg="white", font=("Arial", 14)
279 )

```

```

280     export_button.pack(pady=10, fill=tk.X)
281
282     back_button = tk.Button(
283         self, text="Back to Page 1", command=lambda: controller.show_page("Page1"),
284         bg="#f6f6f6", fg="white", font=("Arial", 14), activebackground="#f5f5f5", bd=0, relief="flat"
285     )
286     back_button.pack(pady=20, fill=tk.X)
287
288     def run_selected_model(self):
289         try:
290             selected_model = self.model_var.get()
291             dataset = getattr(self.controller.page1, 'dataset', None)
292
293             if dataset is None:
294                 messagebox.showerror("Error", "Please Load a dataset first.")
295                 return
296
297             run_model(dataset, self.controller.train_percent, self.controller.test_percent, self.controller.val_percent, selected_model)
298
299         except Exception as e:
300             messagebox.showerror("Error", f"Error in model: {e}")
301
302     def run_model_and_show_results(self):
303         dataset = getattr(self.controller.page1, 'dataset', None)
304         if dataset is not None:
305             self.run_selected_model()
306         else:
307             messagebox.showerror("Error", "Please Load a dataset first.")
308
309
310
311
312
313     def train_model(self):
314         """Train the model with the selected dataset."""
315         dataset = getattr(self.controller.page1, 'dataset', None)
316         if dataset is None:
317             messagebox.showerror("Error", "Please Load a dataset first.")
318             return
319
320         if 'label' in dataset.columns:
321             normal_data = dataset[dataset['label'] == 1].iloc[:, :-1].values
322         else:
323             normal_data = dataset.values
324
325         # Select the model
326         selected_model = self.model_var.get()
327
328         if selected_model == "Isolation Forest":
329             model = IsolationForest()
330         elif selected_model == "One-Class SVM":
331             model = OneClassSVM()
332         else:
333             messagebox.showerror("Error", "Invalid model selected.")
334             return
335
336         try:
337             scaler = StandardScaler()
338             X_scaled = scaler.fit_transform(normal_data)
339             # Train the model
340             model.fit(X_scaled)
341             self.controller.trained_model = model
342             self.controller.scaler = scaler
343             messagebox.showinfo("Success", f"{selected_model} training completed successfully!")
344         except Exception as e:
345             messagebox.showerror("Error", f"Error during training: {e}")
346
347
348
349
350     def load_test_dataset(self):
351         """Load the test dataset from a file."""
352         try:
353             file_path = filedialog.askopenfilename(
354                 title="Select Test Dataset",
355                 filetypes=(("CSV Files", "*.csv"), ("Excel Files", "*.xlsx"), ("All Files", "*.*"))
356             )
357             if not file_path:
358                 return
359
360             if file_path.endswith(".csv"):
361                 test_dataset = pd.read_csv(file_path)
362             elif file_path.endswith(".xlsx"):
363                 test_dataset = pd.read_excel(file_path)
364             else:
365                 messagebox.showerror("Error", "Unsupported file format.")
366                 return
367
368             self.controller.test_dataset = test_dataset
369             messagebox.showinfo("Success", "Test dataset loaded successfully!")
370         except Exception as e:
371             messagebox.showerror("Error", f"Error Loading test dataset: {e}")
372
373     def test_model(self, test_dataset):
374         """Test the model with normal and anomalous data."""
375         if test_dataset is None:
376             messagebox.showerror("Error", "Please Load a test dataset first.")
377             return None
378
379         if 'label' not in test_dataset.columns:
380             messagebox.showerror("Error", "Test dataset must have a 'label' column (1 for normal, 0 for anomalies).")
381             return None
382
383         try:
384             normal_data = test_dataset[test_dataset['label'] == 1].iloc[:, :-1].values
385             anomaly_data = test_dataset[test_dataset['label'] == 0].iloc[:, :-1].values
386
387             X_test = np.vstack([normal_data, anomaly_data])
388             y_true = [1] * len(normal_data) + [0] * len(anomaly_data)
389
390
391

```

```

393         scaler = getattr(self.controller, 'scaler', None)
394         if scaler is None:
395             messagebox.showerror("Error", "Scaler not found. Train the model first.")
396             return None
397
398         X_test_scaled = scaler.transform(X_test)
399         self.controller.test_data = (X_test_scaled, y_true)
400         messagebox.showinfo("Success", "Test data prepared successfully!")
401
402         return normal_data
403
404     except Exception as e:
405         messagebox.showerror("Error", f"Error during testing: {e}")
406         return None
407
408     def run_test_model(self, model_type="One-Class SVM"):
409         """Run the test_model function with the loaded dataset."""
410         try:
411             test_dataset = getattr(self.controller, 'test_dataset', None)
412             if test_dataset is None:
413                 messagebox.showerror("Error", "Test dataset is empty or not loaded. Please load a valid dataset.")
414                 return
415
416             X_train = self.test_model(test_dataset)
417             if X_train is None:
418                 return
419
420             scaler = StandardScaler()
421
422             X_train = pd.DataFrame(X_train).fillna(X_train.mean()).values
423             if len(X_train.shape) == 1:
424                 X_train = X_train.reshape(-1, 1)
425
426             X_train_scaled = scaler.fit_transform(X_train)
427
428             test_dataset = pd.DataFrame(test_dataset).fillna(test_dataset.mean()).values
429             if len(test_dataset.shape) == 1:
430                 test_dataset = test_dataset.reshape(-1, 1)
431
432             X_test_scaled = scaler.transform(test_dataset)
433
434             if model_type == "One-Class SVM":
435                 model = OneClassSVM(kernel='rbf', gamma=0.1, nu=0.05)
436                 model.fit(X_train_scaled)
437
438             elif model_type == "Isolation Forest":
439                 model = IsolationForest(n_estimators=100, contamination=0.1, random_state=42)
440                 model.fit(X_train_scaled)
441
442             else:
443                 messagebox.showerror("Error", f"Invalid model type: {model_type}")
444                 return
445
446         except Exception as e:
447             messagebox.showerror("Error", f"Error running test_model: {e}")
448             return None
449
450

```

```

451         predictions = model.predict(X_test_scaled)
452         messagebox.showinfo("Success", f"Model trained and predictions made.\nPredictions: {predictions}")
453
454     except Exception as e:
455         messagebox.showerror("Error", f"Error running test_model: {e}")
456
457
458
459
460
461     def evaluate_model(self):
462         """Evaluate the model by training, scaling, and comparing training and testing results."""
463         try:
464             test_dataset = getattr(self.controller, 'test_dataset', None)
465             if test_dataset is None:
466                 messagebox.showerror("Error", "No test dataset found. Please load a test dataset first.")
467                 return
468
469             if 'label' not in test_dataset.columns:
470                 messagebox.showerror("Error", "Test dataset must contain a 'label' column.")
471                 return
472
473             X_test = test_dataset.iloc[:, :-1].values
474             y_true = test_dataset['label'].values
475
476             trained_model = getattr(self.controller, 'trained_model', None)
477             scaler = getattr(self.controller, 'scaler', None)
478
479             if trained_model is None or scaler is None:
480                 messagebox.showerror("Error", "Model or scaler not found. Please train the model first.")
481                 return
482
483             X_test_scaled = scaler.transform(X_test)
484
485             y_pred = trained_model.predict(X_test_scaled)
486             y_pred_labels = [0 if x == -1 else 1 for x in y_pred]
487
488             # حساب الأداء العام
489             accuracy = accuracy_score(y_true, y_pred_labels)
490             precision = precision_score(y_true, y_pred_labels, average='binary')
491             recall = recall_score(y_true, y_pred_labels, average='binary')
492             f1 = f1_score(y_true, y_pred_labels, average='binary')
493             # conf_matrix = confusion_matrix(y_true, y_pred_labels)
494
495             # عرض النتائج
496             result_message = (
497                 f"Accuracy: {accuracy:.2f}\n"
498                 f"Precision: {precision:.2f}\n"
499                 f"Recall: {recall:.2f}\n"
500                 f"F1-Score: {f1:.2f}\n"
501                 # f"Confusion Matrix:\n{conf_matrix}"
502             )
503             messagebox.showinfo("Evaluation Results", result_message)
504
505             if hasattr(self, 'plot_anomaly_detection_results'):
506                 self.plot_anomaly_detection_results(X_test_scaled, y_true, y_pred_labels, trained_model)
507

```

```

508     except Exception as e:
509         messagebox.showerror("Error", f"Error during evaluation: {e}")
510
511
512
513
514 def plot_anomaly_detection_results(self, X_test_scaled, y_true, y_pred_labels, model):
515     """Plot the anomaly detection results in a popup window."""
516     try:
517         popup = Toplevel()
518         popup.title("Anomaly Detection Results")
519         popup.geometry("800x600")
520
521         fig, ax = plt.subplots(figsize=(10, 6))
522
523         # Plot Feature 1
524         ax.plot(X_test_scaled[:, 0], label='Feature 1', color='blue', alpha=0.6)
525
526         # Plot Feature 2
527         ax.plot(X_test_scaled[:, 1], label='Feature 2', color='green', alpha=0.6)
528
529         anomaly_indices = [i for i, label in enumerate(y_true) if label == 0]
530         ax.scatter(
531             anomaly_indices, # X-axis: Index of the data point
532             X_test_scaled[anomaly_indices, 0], # Y-axis: Feature 1 values
533             c='red', s=50, label='Anomaly', alpha=0.8
534         )
535         ax.scatter(
536             anomaly_indices, # X-axis: Index of the data point
537             X_test_scaled[anomaly_indices, 1], # Y-axis: Feature 2 values
538             c='orange', s=50, label='Normal', alpha=0.8
539         )
540
541         ax.set_title("Anomaly Detection Results (Line Plot)")
542         ax.set_xlabel("Data Point Index")
543         ax.set_ylabel("Feature Value")
544         ax.legend()
545         ax.grid(True)
546
547         canvas = FigureCanvasTkAgg(fig, master=popup)
548         canvas.draw()
549         canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
550
551         close_button = tk.Button(popup, text="Close", command=popup.destroy)
552         close_button.pack(pady=10)
553
554     except Exception as e:
555         messagebox.showerror("Error", f"Error during plotting: {e}")
556
557
558
559
560

```

```

561
562
563 def export_results(self):
564     if self.report is None:
565         messagebox.showerror("Error", "No results to export. Please run the model first.")
566         return
567
568     file_path = filedialog.asksaveasfilename(
569         defaultextension=".txt",
570         filetypes=[("Text files", "*.txt")],
571         title="Save Results As"
572     )
573     if not file_path:
574         return
575
576     try:
577         with open(file_path, "w") as file:
578             file.write(self.report)
579         messagebox.showinfo("Success", "Results exported successfully!")
580     except Exception as e:
581         messagebox.showerror("Error", f"Error exporting results: {e}")
582
583
584
585 if __name__ == "__main__":
586     app = MultiPageApp()
587     app.mainloop()
588

```

References

- [1] Bace, R. G., & Mell, P. (2001). Intrusion detection systems.
- [2] Singh, A. P., & Singh, M. D. (2014). Analysis of host-based and network-based intrusion detection system. *International Journal of Computer Network and Information Security*, 6(8), 41–47.
- [3] Thakur, K., & Pathan, A. S. K. (2020). *Cybersecurity fundamentals: A real-world perspective*. CRC Press.
- [4] Paffenroth, R. C., & Zhou, C. (2019). Modern machine learning for cyber-defense and distributed denial-of-service attacks. *IEEE Engineering Management Review*, 47(4), 80–85.
- [5] Karatas, G., Demir, O., & Sahingoz, O. K. (2018, December). Deep learning in intrusion detection systems. In *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)* (pp. 113–116). IEEE.
- [6] Sharafaldin, I., Gharib, A., Lashkari, A. H., & Ghorbani, A. A. (2018). Towards a reliable intrusion detection benchmark dataset. *Software Networking*, 2018(1), 177–200.
- [7] Gaggero, G. B. (2022). Machine learning based anomaly detection for cybersecurity monitoring of critical infrastructures.
- [8] Shyaa, M. A., Ibrahim, N. F., Zainol, Z., Abdullah, R., Anbar, M., & Alzubaidi, L. (2024). Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion detection systems. *Engineering Applications of Artificial Intelligence*, 137, 109143.
- [9] Paul, J. (2024). Comparative analysis of supervised vs. unsupervised learning in API threat detection.
- [10] Zakariah, M., AlQahtani, S. A., Alawwad, A. M., & Alotaibi, A. A. (2023). Intrusion detection system with customized machine learning techniques for NSL-KDD dataset. *Computers, Materials & Continua*, 77(3).
- [11] Steingartner, W., Galinec, D., & Kozina, A. (2021). Threat defense: Cyber deception approach and education for resilience in hybrid threats model. *Symmetry*, 13(4), 597.