



REPUBLIC OF TÜRKİYE

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Electrical and Computer Engineering

**IMPLEMENTATION OF AI-BASED DETECTION
SYSTEMS FOR SECURING INTERNET OF
THINGS (IOT) CONTEXTS**

Mohamed Bahaulddin Husein AL-TAMEEMI

Master's Thesis

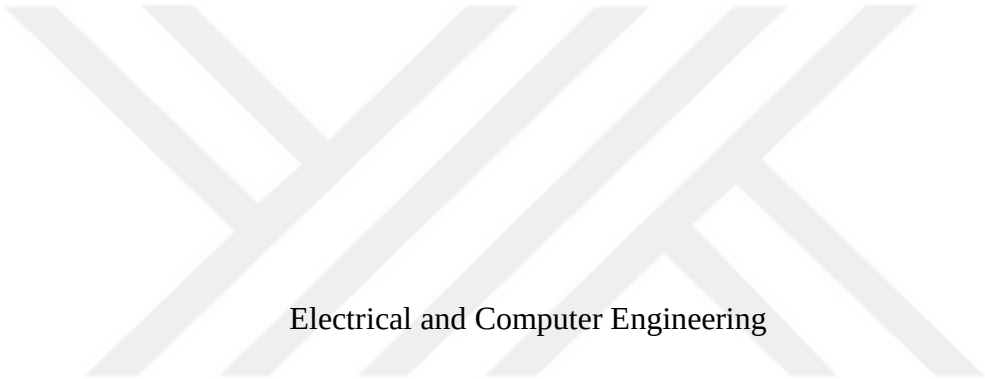
Supervisor

Asst. Prof. Dr. Ayça KURNAZ TÜRK BEN

İstanbul, 2024

IMPLEMENTATION OF AI-BASED DETECTION SYSTEMS FOR SECURING INTERNET OF THINGS (IOT) CONTEXTS

Mohamed Bahaulddin Husein AL-TAMEEMI



Electrical and Computer Engineering

Master's Thesis

ALTINBAŞ UNIVERSITY

2024

The thesis titled IMPLEMENTATION OF AI-BASED DETECTION SYSTEMS FOR SECURING INTERNET OF THINGS (IOT) CONTEXTS prepared by MOHAMED AL-TAMEEMI and submitted on 5/1/2024 has been **accepted unanimously** for the degree of Master of Science in Electrical and Computer Engineering.

Asst. Prof. Dr. Ayça Kurnaz TÜRKBEN
Supervisor

Thesis Defense Committee Members:

Asst. Prof. Dr. Ayça Kurnaz TÜRKBEN

Department of Software
Engineering,
Altınbaş University

Asst. Prof. Dr. Oğuz KARAN

Department of Software
Engineering,
Altınbaş University

Asst. Prof. Dr. Serdar KARGIN

Department of Biomedical
Engineering,
Istanbul Arel University

I hereby declare that this thesis meets all format and submission requirements of a Master's thesis.

I hereby declare that all information/data presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

Mohamed Bahaulddin Husein AL-TAMEEMI

Signature



DEDICATION

I want to start by giving Allah, the Almighty, praise and thanks for giving me the knowledge, abilities, and opportunity to carry out and successfully complete my research. I genuinely thank my parents. Everything wonderful that has happened to me has been fueled by them, and their love is what makes everything possible. During this time, I'd like to express my gratitude to my sister and brother for their love and support.



PREFACE

I owe a lot of gratitude to my advisor, Asst. Prof. Dr. Asst. Prof. Dr. Ayça Kurnaz TÜRKBEN, for providing me with the direction, oversight, inspiration, and assistance I required to complete my studies. My understanding and confidence have been substantially improved by his tenacity, desire to work with me, constructive criticism, and helpful recommendations. He reviewed my work and pointed out my errors, and I appreciate his time spent doing so. Last but not least, I would want to thank everyone for their support, prayers, and/or helpful advice.



ABSTRACT

IMPLEMENTATION OF AI-BASED DETECTION SYSTEMS FOR SECURING INTERNET OF THINGS (IOT) CONTEXTS

AL-TAMEEMI, Mohamed Bahaulddin Husein

M.Sc., Electrical and Computer Engineering, Altınbas University,

Supervisor: Asst. Prof. Dr. Ayça KURNAZ TÜRK BEN

Date: January/2024

Pages: 63

This research utilizes six algorithms, with four originating from the field of machine learning (ElasticNet, Extra Trees, GBC and RF) and the other two from learning (LSTM and GRU). According to the study's findings ElasticNet emerges as the performer in terms of both accuracy and computational efficiency outshining GRU and LSTM models in terms of time efficiency. However, when it comes to precision, recall and F1 score evaluation the LSTM model surpasses the GRU model. Particularly noteworthy is LSTMs superior training efficiency compared to GRU. The choice between LSTM and GRU depends on factors such as the nature of the problem at hand characteristics of the dataset used and available computing resources. In terms of performance both LSTM and GRU models demonstrate levels of accuracy, recall, precision and F1 score with an approximate value of 97%. Amongst all the models examined in this research study stands out Random Forest algorithm – an machine learning technique that combines multiple decision trees for making predictions. This model exhibits performance across accuracy metrics well as recall, precision and F1 score measurements. Additionally the Random Forest model showcases efficiency in both training time and prediction time which makes it an appealing choice for achieving a balance, between performance level achieved and computational efficiency. The Extra Trees algorithm also performs well in terms of accuracy and other evaluation measures although it requires longer training times compared to Random Forest. However, its efficiency remains significant. Although the Gradient Boosting Classifier demonstrates classification accuracy it does require an amount of training time compared to other classifiers. When considering the trade off, between cost and performance improvement the importance of performance becomes crucial. A comparative analysis with research

highlights the performance of the algorithm set proposed in this study. These models show promise, for generalization and testing using data sources potentially enhancing intrusion detection procedures.

Keywords: Long Short-Term Memory, Gated Recurrent Unit, Extra Trees, Internet Security, Intrusion Detection Systems.



TABLE OF CONTENTS

	<u>Pages</u>
ABSTRACT.....	vii
LIST OF TABLES	xi
LIST OF FIGURES	xii
ABBREVIATIONS	xiv
1. INTRODUCTION.....	1
1.1 INTRODUCTION.....	1
1.2 HYPOTHESIS STATEMENT	3
1.3 RESEARCH QUESTIONS	4
1.4 OBJECTIVES	4
1.5 IOT CHALLENGES	5
2. LITERATURE REVIEW	7
2.1 THE UNSW NB15 DATASET; AN EXCEPTIONAL RESOURCE	7
2.2 DETECTING THREATS IN PREVIOUS STUDIES	8
3. METHODOLOGY.....	13
3.1 THE ARCHITECTURE OF IOT.....	13
3.2 IDS ANATOMY	15
3.3 DETECTION METHODS IOT	17
3.4 METHOD MACHINE LEARNING.....	20
3.4.1 Random Forest.....	20
3.4.2 Extra Trees	22
3.4.3 Gradient Boosting Classifier	23
3.4.4 Elasticnet	25
3.5 METHOD DEEP LEARNING	27

3.5.1 Gated Recurrent Unit (GRU)	27
3.5.2 Long Short-Term Memory	28
3.6 EVALUATING THE PERFORMANCE	30
4. RESULTS AND DISCUSSION	33
4.1 RESULTS OF MACHINE LEARNING ALGORITHMS	34
4.1.1 Extra Trees Classifier.....	34
4.1.2 Random Forest Classifier.....	36
4.1.3 Gradient Boosting Classifier	39
4.1.4 Compare The Machine Learning Algorithms Results	41
4.1.5 Linear Model ElasticNet	41
4.2 RESULTS OF DEEP LEARNING ALGORITHMS	44
4.2.1 Gated Recurrent Unit (GRU)	44
4.2.2 Long Short-Term Memory	45
4.2.3 Compare The Deep Learning Algorithms Results	46
4.3 COMPARISON WITH PREVIOUS STUDIES	47
5. CONCLUSION	48
REFERENCES	49

LIST OF TABLES

	<u>Pages</u>
Table 2.1: A Table Containing Some Previous Studies	12
Table 4.1: Comparison of Previous Studies	47



LIST OF FIGURES

	<u>Pages</u>
Figure 2.1: Internet of Things Attacks	10
Figure 2.2: Attacks Classification.....	11
Figure 3.1: Figure the Architecture of IoT [46].....	15
Figure 3.2: Random Forest Works [59]	21
Figure 3.3: Random Forest Extra Trees [61].....	22
Figure 3.4: ElasticNet Regression [65]	26
Figure 3.5: Long Short-Term Memory vs Gated Recurrent Unit (GRU) [72].....	30
Figure 4.1: Extra Trees Classifier Code.....	35
Figure 4.2: Extra Trees Classifier Results.....	35
Figure 4.3: Extra Trees Classifier Confusion Matrix Display.....	36
Figure 4.4: Random Forest Classifier Code	37
Figure 4.5: Random Forest Classifier Results.....	38
Figure 4.6: Random Forest Classifier Confusion Matrix Display	38
Figure 4.7: Gradient Boosting Classifier Code	39
Figure 4.8: Gradient Boosting Classifier Results	40
Figure 4.9: Gradient Boosting Classifier Confusion Matrix Display	41
Figure 4.10: Compare the Results.....	41
Figure 4.11: ElasticNet Code	42
Figure 4.12: ElasticNet Results	42
Figure 4.13: The Regression Metrics.....	43
Figure 4.14: Plotting the True Values vs. Predicted Values	43
Figure 4.15: Gated Recurrent Unit Results	45
Figure 4.16: Plot Training Accuracy and Plot Training Loss to GRU	45

Figure 4.17: Long Short-Term Memory Results	46
Figure 4.18: Long Short-Term Memory Results	46
Figure 4.19: The Deep Learning Algorithms Compare	47



ABBREVIATIONS

CPS	:	Cyber Physical Systems
DDoS	:	Distributed Denial of Service
DL	:	Deep Learning
IDSS	:	Intrusion Detection Systems
IoT	:	Internet of Things
ML	:	Machine Learning

1. INTRODUCTION

1.1 INTRODUCTION

The Internet of Things (IoT) has completely transformed how humans interact with technology by enabling objects to connect and communicate with each other. However, the increasing adoption of IoT has also brought security risks as many IoT devices lack the safeguards to protect against cyberattacks [1]. This makes them vulnerable, to threats. As a result, ensuring the safety of environments has become a concern for both consumers and businesses. Recently AI powered detection systems have emerged as a solution to address the security challenges posed by the Internet of Things (IoT). These systems utilize intelligence algorithms and machine learning techniques to identify risks and anomalies in IoT settings enabling proactive safety measures [2]. Internet of Things (IoT) devices are becoming increasingly prevalent in various industries [3], including healthcare, manufacturing, transportation and smart homes. However, this widespread adoption has also led to a rise in cyberattacks targeting these devices. Due to security measures on IoT devices they are vulnerable to various cyber threats such as DDoS attacks, ransomware incidents and data breaches. Not do these attacks compromise the security of devices but they also pose a risk to the entire network they are connected to address these security vulnerabilities several solutions have been proposed. One approach is implementing security measures, like firewalls, encryption techniques and access control mechanisms [4]. However, there are limitations associated with these strategies when it comes to the Internet of Things (IoT). It is possible that they may not be scalable, adaptable or effective enough to address the range of devices and their unique security requirements. This thesis aims to investigate the use of AI based detection systems, for safeguarding environments. Specifically, it will explore the types of security risks that affect devices various strategies proposed for mitigating these threats and assess how AI based detection systems can enhance IoT security. The thesis will also present the results of developing security systems for the Internet of Things using AI. It will describe the techniques used to build these systems. In the conclusion it will discuss the implications of these findings. Suggest areas, for research. AI powered detection systems present an alternative, to security measures in IoT. These systems leverage machine learning algorithms and real time data analysis to identify potential threats and abnormalities in IoT environments. This makes them a promising substitute for security mechanisms. By

analyzing patterns and behaviors of devices and users these systems can detect malicious activities and take proactive measures to safeguard against them [5]. Integration of AI based detection systems for securing environments involves overcoming operational challenges such as data collection, algorithm selection, model training and deployment. However, given the benefits they offer in terms of enhancing the security and resilience of settings they have emerged as an intriguing area, for research and development. To summarize one of the areas of focus and development, in the field of Internet of Things (IoT) security is the implementation of detection systems that utilize intelligence. These systems aim to protect environments by identifying security vulnerabilities and devising strategies to mitigate associated risks. The objective of this thesis is to contribute to this field by investigating the effectiveness and practicality of AI based detection systems in identifying security threats within settings. Additionally, it aims to develop measures that can enhance security and minimize the impact of cyberattacks on devices and networks. The application of data mining, machine learning and learning techniques, in Intrusion Detection Systems (IDSs) has shown promise in improving the accuracy of detecting activities [6]. These approaches have the ability to analyze amounts of data and identify patterns that indicate threats. While achieving detection accuracy is impressive it's crucial to take into account other factors such as the speed of detection and practical challenges during deployment.

Detection Speed; Achieving high accuracy often requires complex and computationally intensive algorithms. Consequently, some IDSs based on machine learning and deep learning may have detection times, which can pose issues in time or high throughput environments. In applications where quick response is crucial the delay introduced by these systems may not be acceptable [7].

Resource Requirements; Machine learning and deep learning algorithms typically demand resources including memory and processing power. Deploying IDSs that are resource intensive on power or resource constrained devices for instance may be impractical. It's essential to strike a balance, between detection accuracy and resource limitations.

Minimizing False Positives; While achieving accuracy is desirable minimizing positives is equally important. High rates of positives can result in fatigue, where security personnel are bombarded with notifications making it difficult to discern genuine threats. Finding the balance, between accuracy and false positives is crucial [8].

Adaptability; Intrusion Detection Systems (IDSs) should possess the capability to adapt to

emerging and evolving threats. Machine Learning (ML) and Deep Learning (DL) models may require updates and retraining in order to maintain effectiveness. Maintaining an IDSs adaptability without compromising detection speed or accuracy can present challenges [9]. Real world Variability; ML and DL models are typically trained on data, which can lead to performance degradation when confronted with previously unseen attack patterns. Ensuring that IDSs are resilient and capable of handling real world variability is of importance.

Complexity; The implementation and maintenance of ML and DL based IDSs can be both expensive and complex. Organizations must take into account the costs associated with hardware, software, training data as the expertise required for system development and operation.

Explain ability; ML and DL models are often referred to as " boxes " making it difficult to explain their decision-making process. In security contexts understanding why a specific alert or detection was triggered is essential in order to take action [10].

To tackle these challenges organizations and researchers should strive for a balance, between detection accuracy and speed while considering resource limitations, adaptability, as real-world variability.

Furthermore, integrating machine learning (ML) and deep learning (DL) techniques, with rule-based methods has the potential to decrease the occurrence of identifications and enhance comprehensibility. Ultimately when selecting an Intrusion Detection System (IDS) technology it is crucial to ensure that it aligns with the security needs, infrastructure and operational limitations of the organization.

1.2 HYPOTHESIS STATEMENT

After conducting a review of literature and considering the challenges inherent in securing IoT contexts it is hypothesized that implementing AI based detection systems can enhance the security and resilience of Internet of Things environments. These systems have the potential to identify threats and abnormalities in time enabling the creation of proactive security measures that effectively mitigate risks associated with cyberattacks on IoT devices and networks. The purpose behind this thesis is to contribute to existing research, on AI based detection systems and internet of things security by validating this hypothesis through implementation using AI based detection mechanisms.

1.3 RESEARCH QUESTIONS

To address the hypothesis statement this thesis will be guided by the following research questions.

- a. What are the primary security threats that IoT devices are vulnerable, to and how have researchers addressed them in the existing literature?
- b. What are the benefits and limitations of using AI based detection systems to secure contexts. How do they compare to traditional security mechanisms?
- c. What is the effective methodology for implementing AI based detection systems in securing contexts and what factors should be considered during the implementation process?
- d. How accurate are AI based detection systems in identifying threats and anomalies in environments? Additionally, how successful are they in mitigating security risks and minimizing the impact of cyber-attacks on devices and networks?
- e. What are the implications of these findings for the field of security and what areas of research do these results suggest?

1.4 OBJECTIVES

The main goal of this thesis is to explore how AI based detection algorithms can be utilized to enhance security measures in environments. Specifically, this study aims to. Identify security risks faced by devices and evaluate the effectiveness of current defense strategies against these risks. Assess the suitability and effectiveness of employing AI based detection methods, for safeguarding IoT environments while comparing them with traditional security approaches. Develop a methodology, for implementing detection systems based on intelligence that takes into account the characteristics and requirements of Internet of Things (IoT) environments to ensure the security of IoT contexts. Evaluate the accuracy and effectiveness of AI based detection systems in identifying threats and anomalies in environments well as their ability to mitigate security risks and minimize the impact of cyberattacks on IoT networks and devices. Examine the findings of this study. Discuss their implications for the security industry. Identify any limitations in the study. Propose areas for future research. By achieving these objectives this thesis aims to contribute to the existing body of knowledge regarding AI based detection systems and their role in enhancing security. It also seeks to provide insights, into how such systems can bolster the security and

resilience of IoT environments.

1.5 IOT CHALLENGES

The challenges discussed in the passage shed light on some of the issues related to IoT (Internet of Things) networks. Let's take a look, at these challenges;

- i. Scalability; IoT networks often involve a range of devices from sensors and actuators to gadgets used by consumers. Scaling these networks to accommodate an increasing number of devices while ensuring performance and reliability can pose difficulties [11].
- ii. Reliability; Many IoT applications encompass functions like healthcare monitoring, industrial automation and transportation systems. Ensuring the dependability of devices and networks is vital to prevent failures that could cause disruptions or even endanger lives.
- iii. Privacy; IoT devices. Transmit amounts of data, much of which can be sensitive and personal in nature. Safeguarding the privacy of individuals and organizations is a priority in situations where mishandling or unauthorized access to data could occur.
- iv. Security; In order to keep costs down many IoT devices are designed with security features, which can make them targets for malicious individuals. Security breaches can have consequences such as data theft, unauthorized control over devices and even physical harm [12].
- v. Isolation; Some IoT devices operate in environments, like industrial sites or smart homes.

In environments where devices are isolated monitoring and managing them can be quite challenging especially when it comes to ensuring security updates and patches. One of the challenges arises from the approach, to developing and deploying IoT devices. Since these devices are created by manufacturers and placed in locations there is often a lack of standardized security measures and practices. This makes it easier for attackers to exploit vulnerabilities. Another obstacle stems from the resources that many IoT devices have such as processing power, memory and energy. These resource constraints make it difficult to implement security mechanisms that can adequately protect the devices. The heterogeneity of networks adds another layer of complexity. With devices from manufacturers each having its own hardware, software and communication protocols implementing standardized security solutions becomes a challenging task. The absence of a IoT architecture also

contributes to the difficulty in establishing consistent security practices across different IoT ecosystems. This lack of standardization leads to fragmentation. Creates difficulties in ensuring security across the board. Lastly the potential impact of attacks on networks is significant due to their integration into infrastructure and various aspects of daily life. Breaches or disruptions, in these networks can result in disasters, substantial financial losses and even loss of lives. To tackle these difficulties, it is crucial for groups involved, such as manufacturers, policymakers and cybersecurity experts to work. Implementing security measures safeguarding privacy and following protocols are vital, in reducing the potential dangers linked to the rapid expansion of IoT networks and guaranteeing their secure and dependable functioning.



2. LITERATURE REVIEW

In today's era the ongoing battle, against cyber threats requires the development of Intrusion Detection Systems (IDSs). Over time the UNSW NB15 dataset has become a resource for researchers aiming to improve the effectiveness of these security measures. This article offers a review of studies carried out using the UNSW NB15 dataset highlighting notable sources and their contributions to the field of intrusion detection [13].

2.1 THE UNSW NB15 DATASET; AN EXCEPTIONAL RESOURCE

The UNSW NB15 dataset has been seriously curated by a large group of researchers, confirming its importance in network security research. This dataset has characteristics that make it extremely valuable;

Diverse Attack Scenarios; The dataset covers a range of cyberattacks, including Denial of Service (DoS) Distributed Denial of Service (DDoS) Reconnaissance, Exploitation and more. This diversity accurately reflects the complexity of real-world cyber threats making it an ideal testing ground, for IDSs [14].

Realistic Network Traffic; The authenticity of network traffic is a distinguishing feature of the UNSW NB15 dataset.

Based on network traffic this dataset accurately reproduces the complexities of network environments. This ensures that research findings can be applied to real world scenarios effectively. To simulate the nature of network ecosystems the dataset includes both benign and malicious traffic. This balanced representation allows researchers to assess the ability of intrusion detection systems (IDSs), in distinguishing between potentially harmful network activities [15]. The dataset offers a range of features that encompass attributes at both flow levels as well as statistical characteristics. These features empower researchers to extract information for IDS development and evaluation purposes. Previous studies have made contributions to the field of intrusion detection using the UNSW NB15 dataset. One notable contribution comes from a paper "A Deep Learning Approach, for Network Intrusion Detection System." Their work demonstrates how learning techniques can effectively detect intrusions with accuracy using this dataset [16]. A study conducted by Moustafa and Slay (2016) Evaluation of Machine Learning Algorithms, for Network Intrusion Detection performance evaluation metrics such as accuracy, precision, recall and F1 score for Intrusion Detection Systems (IDSs). They utilized the UNSW NB15 dataset to establish a benchmark

for comparing IDS approaches [17]. They focused on feature selection techniques specifically designed for network traffic data in their publication Feature Selection Technique for Dealing with Imbalanced Data in Network Intrusion Detection. Their findings emphasized the importance of feature selection to enhance the efficiency of IDSs [18]. In their survey titled " Survey of intrusion detection systems: techniques, datasets and challenges" Khraisat et al. (2019) examined IDSs, including those that utilized the UNSW NB15 dataset highlighting advancements in anomaly detection techniques [19]. In the study titled "An Ensemble Intrusion Detection Technique Based on Correlation Filter and Principal Component Analysis " researchers explored approaches to intrusion detection systems (IDSs) using the UNSW NB15 dataset. Their research showed that combining models can improve the accuracy of detecting intrusions (Ismail et al., 2020) [20]. The resilience of IDSs, against attacks was examined in the research paper " A comprehensive survey of data mining. International Journal of Information Technology " by Gupta et al. (2020) which used the dataset. The researchers emphasized the importance of having intrusion detection systems that can withstand evolving threats [21]. Alazab et al. (2019) conducted dataset evaluations in their research paper titled "A Deep Learning Approach to Network Intrusion Detection." They assessed the adaptability and generalizability of intrusion detection models trained on the UNSW NB15 dataset highlighting how these models can be applied effectively in network environments [22]. The impact of studies conducted using the UNSW NB15 dataset has influenced practices, within the industry. Organizations have incorporated the lessons they've learned into their cybersecurity strategies strengthening their network security defenses. The UNSW NB15 dataset serves as a resource, for the development and evaluation of Intrusion Detection Systems. Previous research based on this dataset has resulted in algorithms, standardized performance metrics improved methods for selecting features and increased resilience against cyber threats. As the cybersecurity landscape continues to change the insights gained from the UNSW NB15 dataset will play a role, in shaping intrusion detection systems and bolstering our defenses against emerging threats.

2.2 DETECTING THREATS IN PREVIOUS STUDIES

The development of intrusion detection systems that prioritize human centric approaches was researched by Fernandez Carames and Fraga Lamas (2019). They demonstrated how incorporating user behavior and interaction patterns can enhance threat detection making

IDSs more context aware and improving cybersecurity, in environments where humans interact with machines [23] In a world where privacy is highly valued intrusion detection systems must also prioritize data protection. Laqtib et al. (2019) conducted a study Privacy Preserving Intrusion Detection; An Overview, which explored techniques for developing IDSs that effectively detect intrusions while maintaining data privacy. This is particularly crucial for organizations dealing with information [24]. The scalability and resource efficiency of intrusion detection systems have been the focus of studies such as Scalability and Performance Evaluation of Intrusion Detection Systems, by García Teodoro et al. (2009). These studies emphasize the importance of developing cost solutions of handling large scale network traffic [25]. Hybrid IDSs, which combine the strengths of intrusion detection techniques have garnered attention in recent years. Research conducted by Gaspar et al. (2023) in their study The present study aims to provide a comprehensive survey on the topic of smart substation communications and cybersecurity [26]. When it comes to machine learning based IDS ensuring interpretability is crucial. Explainable Artificial Intelligence for Secure Intrusion Detection Systems explored techniques aimed at making complex ML models more transparent and explainable which's highly valuable for cybersecurity professionals. To enhance the capabilities of IDSs there has been a research focus on utilizing real time threat intelligence feeds by Yan, J. et al. (2016) have demonstrated how timely threat information can be integrated into IDSs to enhance their accuracy and responsiveness [27]. Intrusion detection systems need to adapt not to protect networks but also cyber physical systems (CPS). Siddiqui et al. (2018) in their review explored the challenges and proposed solutions specifically related to intrusion detection, in CPS environments [28]. Cloud computing and the Internet of Things (IoT) present security challenges, for intrusion detection systems (IDSs). discussed these challenges in their study titled "A Survey on Intrusion Detection for Cloud Computing " while Buczak and Guven (2016) examined the security and privacy concerns in IoT based IDSs [29]. Their research emphasizes the importance of IDS development in relation to emerging technologies. Efforts have been made to integrate IDSs across domains aiming to create a network security approach in complex interconnected environments. Explored this integration concept in An Integrated Intrusion Detection System for Cloud and IoT Security. Furthermore, researchers have investigated the potential of machine learning models, in detecting zero-day attacks. conducted Zero Day Attack Detection; [30] Challenges and Opportunities to explore how

ML based IDSs can identify previously threats. Deep Learning Progress; Deep learning has gained attention in the field of Intrusion Detection Systems (IDSs). A comprehensive overview of learning techniques and their applications, in intrusion detection can be found in of Deep Learning for Network Intrusion Detection. Continuous Learning and Real Time Updates; In dynamic network environments the ability to adapt to changes is crucial. Various studies, including Online Learning Algorithms for Intrusion Detection have explored online learning approaches that enable IDSs to adapt and update their knowledge. IDSs Designed for IoT; With the increasing prevalence of devices specialized intrusion detection systems tailored specifically for environments have become necessary. IoT Intrusion Detection Systems provides insights into the challenges and solutions in IoT specific IDS research [31]. Integration of Blockchain and IDS; The combination of technology with intrusion detection has gained popularity recently. A Comprehensive Survey on Blockchain Based Intrusion Detection and Response Systems for IoT by Samani, P. (2023) delves into how blockchain can enhance security and trust, in IDSs designed for environments [32]. The following figure shows the types of Internets of Things attacks.

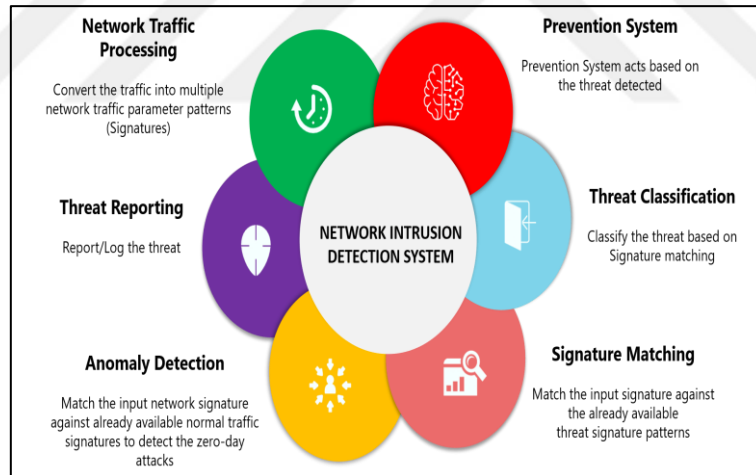


Figure 2.1: Internet of Things Attacks.

NLP techniques have been utilized in the analysis of log data, for intrusion detection. A study conducted Using Natural Language Processing for Intrusion Detection through HTTP Network Traffic Analysis explores the application of NLP in extracting insights from network logs [33]. Various studies have looked into integrating vulnerability assessment tools with IDSs to proactively identify and mitigate vulnerabilities. A comprehensive survey conducted Incorporating Vulnerability Analysis in Network Intrusion Detection Systems provides an overview of this approach. Zero Trust has emerged as a concept in network

security Zero Trust Network Security; examining how Zero Trust principles can be incorporated into IDSs to enhance network defense.

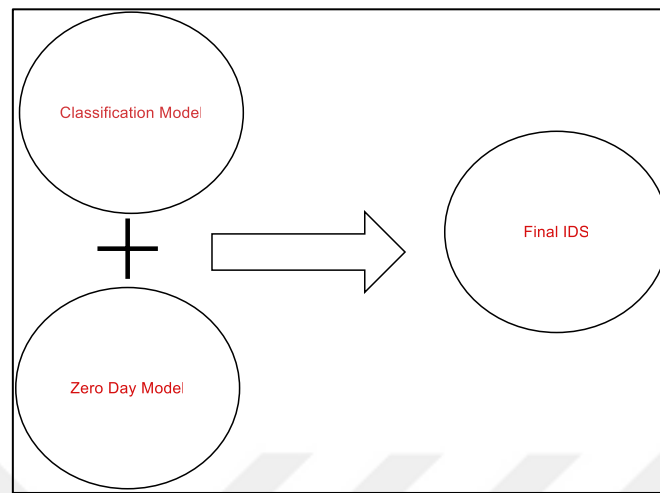


Figure 2.2: Attacks Classification.

The issue of positives in IDSs has led to research on developing resilience mechanisms. A Survey of False Positives Reduction, in Intrusion Detection Systems" discusses strategies aimed at minimizing alarms while maintaining high detection rates. The emergence of quantum computing brings forth both possibilities and challenges, for intrusion detection systems (IDSs). Opportunities and Challenges of Quantum Computing in Intrusion Detection Systems investigates the impact that quantum computing could have on network security. When it comes to intrusion detection systems (IDSs) ethical considerations come into play due to the collection and analysis of network data. as Examining the Ethical Implications of Network Intrusion Detection Systems delves into the dimensions surrounding the deployment and operation of IDSs [34]. Collaboration, between Intrusion Detection Systems (IDSs) and organizations has become crucial in addressing threats. A comprehensive Collaborative Intrusion Detection Systems; Exploring Benefits and Challenges extensively discusses the advantages and obstacles associated with intrusion detection. The development of Intrusion Detection Systems (IDSs) that focus on modeling and detecting behavior patterns is an aspect of modern threat detection. Behavior Based Intrusion Detection Systems; An In-depth Examination. Research has focused on integrating IDSs with incident response systems to enhance their effectiveness. Intrusion Detection and Response; provides insights into how IDSs can contribute to incident response strategies [35]. The advent of data has significantly influenced research in intrusion detection systems. Big Data Analytics for Intrusion Detection; explores how big data technologies can be

utilized to analyze volumes of network data Federated learning, an approach that enables collaboration among organizations without sharing sensitive data holds great potential for intrusion detection systems. Federated Learning for Intrusion Detection; delves into the applicability of federated learning in this domain. Deep Packet Inspection for Network Intrusion Detection; Opportunities and Challenges have explored the utilization of inspection (DPI) techniques, [36] in IDSs. DPI provides an examination of network packets to identify threats. When it comes to Intrusion Detection Systems (IDSs) it is crucial to ensure transparency and interpretability of AI based IDSs, for Intrusion Detection explore various Explainable AI techniques that aim to make complex models more understandable. The following table shows some previous studies with the accuracy measure used for each study.

Table 2.1: A Table Containing Some Previous Studies.

IDS	Model(s)	Dataset	Accuracy (%)
AL-Hawawreh [37]	autoencoder	UNSW-NB15	92.4
Zhou [38]	gradient-boosted trees	UNSW-NB15	91.22
	Knn		91.9
	DT		92.29
	logistic regression		90.35
	gaussianNB		92.52
	SVM		92.32
TAMA [39]	Rotation	UNSW-NB15	72.52
	Forest+bagging		

3. METHODOLOGY

3.1 THE ARCHITECTURE OF IOT

The architecture of IoT (Internet of Things) refers to the structure or framework that allows for the integration of components, devices and technologies, within an ecosystem. It encompasses the design principles, protocols and layers necessary to connect and manage devices gather data, process information and provide services. The specific IoT architecture can vary depending on the application at hand. Typically includes components and layers as outlined below;

i. Devices or Things Layer

This layer comprises devices or "things" such as sensors, actuators, smart appliances, wearables and industrial machines. These devices are equipped with sensors to collect data and actuators to perform actions based on commands or triggers. Communication protocols like Wi Fi, Bluetooth, Zigbee, LoRa cellular networks are used for device connectivity [40].

ii. Connectivity Layer

The connectivity layer facilitates communication, between devices and the network. There are technologies and protocols involved such, as; Wireless Communication; Wi Fi, Bluetooth, cellular (3G/4G/5G) Zigbee, NFC (Near Field Communication) and more. Wired Communication; Ethernet, Power over Ethernet (PoE) RS 232, RS 485 and others. Low Power Wide Area Networks (LPWANs); Technologies like LoRa and Sigfox are suitable for range low power IoT devices [41]. At this layer edge computing can be utilized to preprocess data to the devices. This helps reduce latency and bandwidth requirements.

iii. The Data Acquisition and Processing Layer takes care of collecting, storing and processing data generated by devices. It includes components like data gateways, edge devices and IoT platforms. Preprocessing of data occurs here through filtering and aggregation to decrease the amount of transmitted data while extracting insights.

iv. The IoT Platform Layer provides the infrastructure, for managing, monitoring and controlling both devices and their associated data. IoT platforms typically offer a range of features, including managing devices analyzing data ensuring security, provisioning devices and integrating with systems. Cloud based IoT platforms, like AWS IoT, Azure IoT and Google Cloud IoT are widely used in this layer [42]

v. Application Layer

The application layer is where the data collected from devices is used to create value for end users, businesses or industries. Applications can vary from consumer-focused ones (like home apps and fitness trackers) to industrial solutions (such as predictive maintenance and supply chain optimization). The data gathered from devices is processed and analyzed to generate insights that trigger automated actions [43]

vi. Security and Privacy Layer

Security plays a role in the architecture of IoT to safeguard devices, data and communication channels. security measures such as encryption, authentication protocols, access control mechanisms and regular security updates are implemented across different layers. Privacy concerns related to data ownership and compliance with regulations, like GDPR are also given attention at this layer [44].

vii. User Interface Layer

The user interface layer provides users with a way to interact with and monitor their devices and applications effectively. This can involve creating web and mobile applications, dashboards and user interfaces to configure and control devices.

viii. Analytics and Decision Layer

In this layer advanced analytics, machine learning and AI algorithms are often used to extract insights, from data. Based on the analysis performed at this layer data driven decisions can be. Automated actions can be taken.

ix. Integration Layer

The integration layer connects the ecosystem with systems, databases and enterprise applications. It allows for interoperability and data sharing with existing IT infrastructure.

x. Management and Orchestration Layer;

This layer is responsible for managing resources handling device provisioning and deprovisioning processes well as orchestrating various operations. Its purpose is to ensure the functioning of the IoT system. IoT architecture can be highly customized to meet application or industry requirements. The selection of protocols, components and technologies depends on factors such, as scalability, latency, security measures and cost limitations. Moreover, edge computing, fog computing and decentralized architectures may also be integrated into systems to address performance needs while minimizing latency. The

following figure shows the architecture. [45]

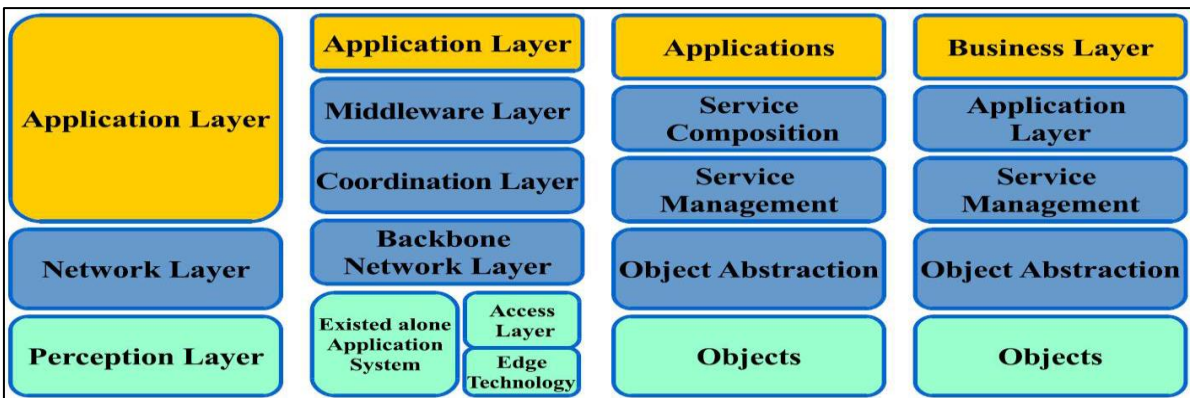


Figure 3.1: Figure the Architecture of IoT [46].

3.2 IDS ANATOMY

An Intrusion Detection System (IDS) is comprised of components that work together to monitor and identify unauthorized activity, within a network or on a specific host. Let’s take a look at the elements of an IDS;

i. Sensors

The sensors play a role as the line of defense in an IDS. Their primary function is to gather data related to network traffic or host behavior.

Network based IDS (NIDS) sensors are strategically positioned at network chokepoints or within segments to monitor and analyze network traffic and packets.

Host based IDS (HIDS) sensors operate on hosts or servers keeping a close eye on activities happening within them such as file modifications, system log entries and process actions [46].

ii. Data Collection

Sensors constantly collect data by monitoring either network traffic or host logs. In the case of NIDS this involves capturing packets. Comprehensively examining their contents.

Data collection encompasses information like source and destination IP addresses ports used protocols employed, packet payload details, file system events recorded process related information and more.

iii. This breakdown provides an overview of how an IDS's structured and how its key components collaboratively ensure monitoring for potential threats, in both network wide and host specific contexts.3. Analysis Engine [47]. The analysis engine plays a role, in processing the gathered data to identify any security threats or abnormal occurrences.

It utilizes methods such as signature-based detection, anomaly-based detection and heuristics to analyze the data for known attack patterns or unusual behavior.

iv. Detection Methods

- a. Signature based Detection; This method involves comparing network traffic or host activities with a database of defined attack signatures or patterns. It effectively detects known threats that match those signatures [48].
- b. Anomaly based Detection; By establishing a baseline of behavior this method identifies deviations from it as anomalies. It proves useful in identifying unfamiliar threats that don't conform to patterns.
- c. Heuristic Detection; Than relying on attack patterns this approach employs rules and heuristics to spot suspicious behavior that may indicate an attack.

v. Rule. Signatures

In signature based and heuristic detection IDSs depend on rule sets or signatures that outline patterns of suspicious activities. Regular updates are made to these rule sets to ensure the IDS remains effective, against emerging threats.

vi. Alert Generation

Whenever the analysis engine detects any activity or potential security threats it promptly generates alerts. Alerts contain information regarding the nature of an event, its severity and the systems that might be affected [50].

vii. Alert Notification

Intrusion Detection Systems (IDSs) can inform administrators or security teams about identified threats through channels, like email, SMS, SNMP traps or integration with Security Information and Event Management (SIEM) systems [51]

viii. Logging and Reporting;

- a. IDSs maintain logs of all detected events and alerts for the purpose of auditing and forensic analysis.
- b. Reporting capabilities enable security teams to generate summaries and reports on security incidents as trends over time.

ix. Response Mechanisms

Certain IDSs have built in response mechanisms. Can integrate with security tools to

automate responses to specific threats, like blocking malicious IP addresses or isolating compromised hosts.

x. User Interface

- a. IDSs typically offer a user interface (UI) or dashboard that allows administrators to configure, monitor and manage the IDS effectively.
- b. The UI also provides real time data display of alerts and reports [52].

xi. Centralized Management

In large scale deployments multiple sensors and instances of IDS may be centrally managed using a management console or dedicated software.

- xii. Keeping the Intrusion Detection Systems (IDSs) up, to date is extremely important. This involves updating the rule sets, signatures and the IDS software itself to address any threats or vulnerabilities that may arise. To summarize an Intrusion Detection System (IDS) consists of components, including sensors, an analysis engine, detection methods, alert mechanisms and reporting capabilities. Its main purpose is to monitor and identify any unauthorized activities occurring within a network or, on a host. By doing it helps organizations promptly detect and respond to security threats.

3.3 DETECTION METHODS IOT

Detecting anomalies or security threats, in environments is extremely important to ensure the integrity and safety of devices and systems. There are methods and techniques that can be used to detect activities or deviations from normal behavior in IoT networks. Let's take a look at some detection methods utilized in the realm [53].

i. Signature based Detection;

Signature based detection relies on defined patterns or signatures of known attacks or malicious behaviors. By comparing devices and network traffic against a database of signatures it becomes possible to identify and flag known threats. For instance, antivirus software can be employed to detect malware on devices while intrusion detection systems (IDS) with signature databases can analyze network traffic.

ii. Anomaly based Detection;

Anomaly detection aims to identify deviations from established baselines of behavior. It creates a profile of behavior for devices or networks raising alerts when anomalies are detected. Machine learning algorithms, statistical methods and heuristics are often employed

for anomaly detection in the context of IoT. Anomaly based detection proves particularly effective in identifying unknown or zero day attacks [54].

iii. These approaches play a role, in maintaining security within the landscape of IoT environments.

3. Behavioral Analysis;

- a. Monitoring the behavior of devices over time enables the detection of changes or deviations.
- b. This involves keeping track of parameters such, as device activity patterns, network communication behavior and power usage.
- c. An alteration in behavior could indicate a compromise or malfunction within a device.

iv. Flow based Analysis;

- a. Flow based analysis focuses on studying the patterns and characteristics of data flows between devices.
- b. It examines the flow of data, including sizes, frequencies and destinations to identify any communication patterns.
- c. Through flow-based analysis we can detect distributed denial of service (DDoS) attacks, data exfiltration attempts and network reconnaissance activities [55].

v. Machine Learning and AI;

- a. Utilizing machine learning and AI techniques allows us to create models that learn the behavior of IoT devices while detecting anomalies.
- b. Algorithms, like decision trees, forests, support vector machines and deep learning networks are employed for classification and anomaly detection purposes.
- c. These models possess the ability to adapt to changing environments and uncover previously threats.

vi. Analyzing the firmware and software that run on devices can help uncover vulnerabilities and potential security issues.

Techniques, like dynamic analysis can be utilized to identify vulnerabilities, malware or unauthorized modifications in the device software [56]. Enhancing device security can be achieved by scanning for known vulnerabilities and applying security patches.

vii. Monitoring network traffic in environments is crucial for detecting any activities.

Deep packet inspection (DPI) and network intrusion detection systems (NIDS) are tools for analyzing traffic patterns associated with attacks or malicious behavior. Network traffic

analysis aids in identifying access, data leaks and unusual communication patterns.

viii. Encryption methods and cryptographic techniques play a role in protecting data and communication channels. Detection methods can include monitoring weakly encrypted traffic identifying anomalies in protocols and recognizing unauthorized cryptographic keys.

ix. Hardware based security measures provide a layer of protection, against tampering of IoT devices. Secure boot processes, hardware-based root of trust and hardware security modules (HSMs) are some techniques used to detect and prevent access, to components of IoT devices.

x. User and Entity Behavior Analytics (UEBA)

UEBA solutions analyze the behavior of users and entities to identify any activities occurring across devices and network users.

These solutions combine machine learning with analytics to detect instances of access and potential insider threats.

xi. Threat Intelligence Feeds

Organizations can enhance their security systems by integrating threat intelligence feeds and databases allowing them to stay updated on known threats and indicators of compromise (IoCs). The inclusion of threat intelligence can improve signature-based detection methods. Help organizations proactively protect their infrastructure [57].

xii. Edge based Detection;

Some IoT deployments utilize edge computing capabilities, which involve analyzing data to the devices themselves. This enables edge-based detection mechanisms that can quickly identify anomalies, triggering alerts or necessary actions at the edge level. This approach reduces latency issues and minimizes bandwidth requirements. To effectively secure systems it is crucial to employ a combination of these detection methods as no single method alone is completely foolproof. Implementing a layered security approach that integrates diverse detection techniques along with regular updates and adherence to best practices is essential, for effectively mitigating risks associated with IoT environments.

3.4 METHOD MACHINE LEARNING

3.4.1 Random Forest

Random Forest, a used learning technique, in machine learning is popular for both classification and regression tasks. It is a powerful algorithm that leverages the predictions of multiple decision trees to generate a robust and accurate final prediction. Here's an explanation of how Random Forest works [58];

i. Decision Trees

Before diving into Random Forest, it's important to grasp the concept of decision trees. A decision tree is a model resembling a tree structure, where each internal node represents a feature each branch signifies the decision rule based on that feature and each leaf node represents either a class label (for classification) or a numerical value (for regression).

ii. Ensemble Learning

Random Forest falls under the category of learning methods. Ensemble methods combine models to enhance overall performance and generalization.

In the case of Random Forest these individual models are decision trees.

iii. Introduction of Randomness

The term "random" in Random Forest refers to the incorporation of randomness, during both training and prediction phases [59].

During the training phase Random Forest incorporates randomness in two ways

- a. Bootstrap Aggregating (Bagging); It randomly selects subsets of the training data (with replacement) to train each decision tree. This process generates subsets of the data for each tree.
- b. Feature Randomization: At each split in the decision tree, it randomly chooses a subset of features. This ensures that each tree focuses on sets of features reducing the risk of overfitting.
- c. After training decision trees using data subsets and features Random Forest combines their predictions through a voting mechanism for classification or averaging for regression tasks. In classification tasks the class that receives the number of votes from trees becomes the final prediction. In regression tasks the final prediction is obtained by averaging predictions made by trees.

- d. Random Forest offers advantages.
 - i. It achieves accuracy and is less prone to overfitting compared to using individual decision trees.
 - ii. It can handle both numerical features without preprocessing.
 - iii. It provides insights, into feature importance aiding in feature selection and understanding the significance of variables when making predictions.
 - iv. It can handle data and outliers effectively.

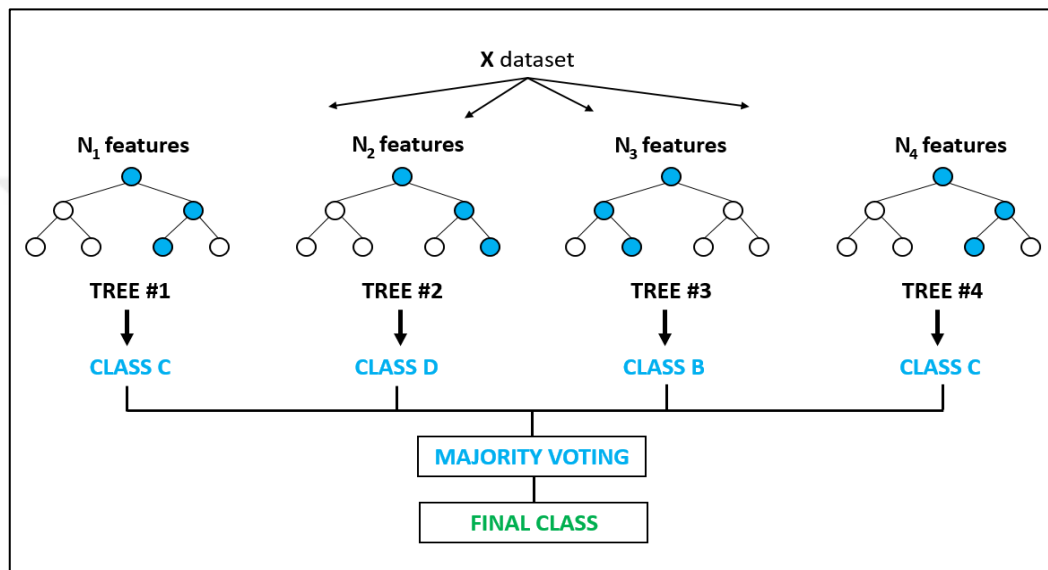


Figure 3.2: Random Forest Works [59].

- e. Drawbacks
 - i. Random Forest models can be computationally intensive when dealing with a number of trees and features.
 - ii. Interpretability can be challenging when working with a Random Forest model [59]
 - iii. In cases Random Forest may not perform well as more advanced algorithms, for specific tasks.
- f. Applications

Random Forest finds usage in applications such as image classification, medical diagnosis, finance (e.g., credit scoring) and natural language processing. In summary Random Forest is a technique for learning that combines multiple decision trees using data sampling and feature selection randomness to generate accurate and robust predictions for both classification and regression tasks. Due to its effectiveness and user friendliness, it is one of the algorithms, in machine learning.

3.4.2 Extra Trees

Extra Trees, which is short, for Trees is a method of ensemble learning that shares similarities with Random Forest. It can be used for both classification and regression tasks. However, there are some differences that make Extra Trees unique [60];

a. Randomization

Like Random Forest Extra Trees also relies on a collection of decision trees as its foundation. The main distinction lies in how the decision trees constructed. In Extra Trees they are grown using randomness.

While Random Forest randomly selects a subset of features at each split Extra Trees takes it a step further. It chooses a threshold for each feature at each split of searching for the optimal threshold as done in Random Forest. This added dose of randomness results in diversity among the trees in Extra Trees.

b. Bagging

Similar, to Random Forest Extra Trees also incorporates bagging techniques by sampling subsets of training data with replacement to train each decision tree [61].

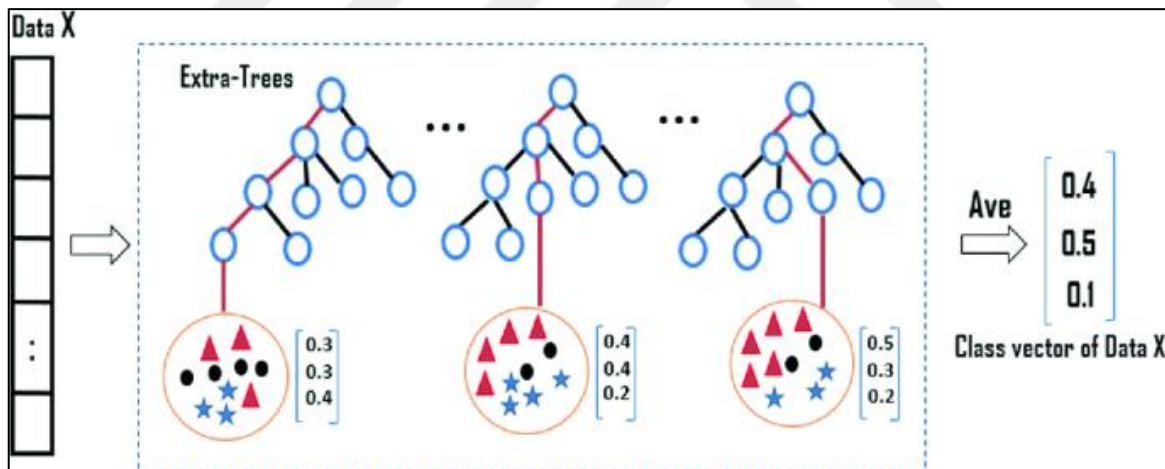


Figure 3.3: Random Forest Extra Trees [61].

c. Feature Importance

Like Random Forest Extra Trees provides a measure of feature importance. This score assists in understanding which features have the impact when making predictions.

d. Benefits

- i. Extra Trees can be computationally efficient compared to Random Forest because it constructs decision trees at a pace due, to its simplified feature selection process.
- ii. The additional randomness in Extra Trees sometimes results in generalization on datasets

that're noisy or less structured.

e. Drawbacks

- i. The increased randomness in Extra Trees might make the model less interpretable than Random Forest as understanding the reasoning behind trees becomes more challenging.
- ii. Extra Trees may not always outperform Random Forest on every dataset; its performance relative to the problem and data at hand varies.

f. Applications

Similar, to Random Forest Extra Trees finds use in applications, including classification, regression and feature selection tasks. In summary Extra Trees is a learning method that builds upon decision trees while introducing more randomness during tree construction compared to Random Forest. This added randomness makes Extra Trees a valuable alternative, especially when computational resources are limited or when you want to explore the advantages of increased diversity within the ensemble. However, deciding between Extra Trees and Random Forest usually relies on the problem. Dataset, at hand. It might be necessary to conduct some experiments to ascertain which one yields results.

3.4.3 Gradient Boosting Classifier

A Gradient Boosting Classifier is a type of algorithm commonly used in machine learning for classification purposes. It falls under the category of methods, which involve combining the predictions, from weak learners usually decision trees to create a robust and accurate predictive model. Gradient Boosting is particularly effective when it comes to improving model accuracy in datasets. Let me explain how Gradient Boosting Classifiers work [62];

i. Weak Learners (Decision Trees)

To start with Gradient Boosting begins by using a learner often a shallow decision tree with limited depth. Although this initial tree may not be very accurate on its own it acts as a foundation for the ensemble.

ii. Boosting Iterations

The process of Gradient Boosting proceeds, by building a series of decision trees. In each iteration an additional decision tree is added to the ensemble to rectify any errors made by the existing ensemble.

iii. Error Correction

During every iteration the Gradient Boosting algorithm identifies those data points (samples)

that were misclassified or had prediction errors within the ensemble. The new decision tree is specifically trained to focus on these misclassified data points. It aims to capture the patterns that were challenging for the trees to identify.

iv. Gradient Descent

- a. The term "gradient", in Gradient Boosting is derived from the utilization of a gradient descent optimization algorithm.
- b. During each boosting iteration a new tree is constructed to minimize the loss function by addressing the errors made by the ensemble.
- c. The gradient descent algorithm adjusts the parameters of the tree in a direction that reduces the loss (i.e., minimizing errors).

v. Weighted Voting

- a. Once a new tree is added to the ensemble it receives a weight based on its performance, in reducing error. More accurate trees are assigned weights.
- b. When making predictions each trees output is combined through a voting process to generate the classification.

vi. Regularization

Gradient Boosting incorporates regularization techniques to prevent overfitting. Common methods include controlling tree depth setting a number of samples required for node splitting and managing learning rate (which scales each trees contribution).

vii. Advantages

Gradient Boosting Classifiers are known for their accuracy and ability to achieve results in a wide range of classification tasks [63]. They handle data relationships effectively. Are resilient, against outliers. Feature importance can be determined from Gradient Boosting models, which's helpful in selecting features.

viii. Disadvantages

Gradient Boosting models can be computationally demanding and may require tuning of hyperparameters. If not properly regularized they can be prone to overfitting. Interpreting Gradient Boosting models can pose challenges due to the ensemble of decision trees.

ix. Applications

Gradient Boosting Classifiers find application in domains such as spam detection, credit scoring, image classification and disease diagnosis. In summary Gradient Boosting

Classifiers are a learning technique that sequentially constructs decision trees to correct errors made by previous trees. This iterative process yields a robust classification model. However, it is essential to tune hyperparameters due, to demands.

3.4.4 Elasticnet

ElasticNet is a technique used in machine learning and statistics for modeling and feature selection. It combines two regularization methods, L1 regularization (Lasso) and L2 regularization (Ridge). ElasticNet aims to overcome the limitations of these two methods by striking a balance, between them [64]

Here's an explanation of ElasticNet in terms;

a. Linear Regression: Linear regression is a learning technique that predicts a target variable based on one or more input features. The goal is to establish a relationship, between the inputs and the target variable.

b. Regularization:

L1 Regularization (Lasso); Lasso adds a penalty term to the linear regression equation encouraging the model to select the important features while setting less important one's coefficients to zero. This helps with feature selection and prevents overfitting.

L2 Regularization (Ridge); Ridge also adds a penalty term but uses squared values. This tends to keep all feature coefficients without setting them to zero.

c. ElasticNet; ElasticNet combines both L1 and L2 regularization techniques bringing together their strengths while addressing their limitations simultaneously. ElasticNet incorporates both the values of the coefficients (L1) and the squared values of the coefficients (L2), as penalty terms in regression. This unique combination allows ElasticNet to perform the tasks;

Select a subset of features, similar to Lasso regularization. Retain some less important features with small coefficients, similar to Ridge regularization. Achieve a balance between feature selection and coefficient shrinkage. There is a hyperparameter in ElasticNet called "alpha" (α) which controls the balance between L1 and L2 regularization. When $\alpha = 0$ ElasticNet behaves like Ridge regression while $\alpha = 1$ makes it behave like Lasso regression. Any value, between 0 and 1 offers a blend of both regularization techniques allowing you to tune how the model behaves. The following figure shows the description.

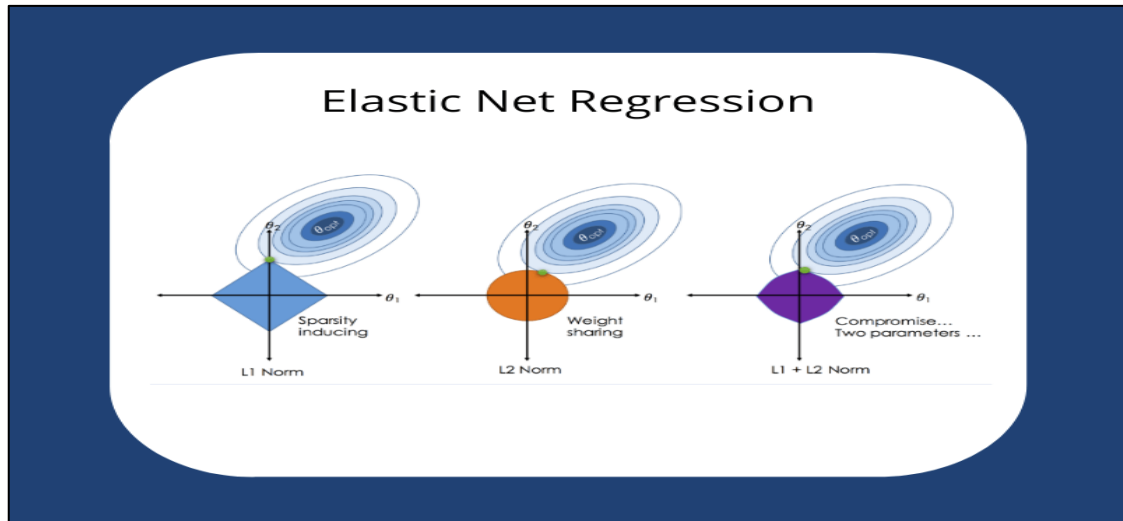


Figure 3.4: ElasticNet Regression [65].

d. ElasticNet finds usefulness in scenarios

When dealing with datasets that have features, some of which may be irrelevant or redundant ElasticNet automatically selects the most important features. It helps prevent overfitting by managing model complexity. It is commonly employed when limitations exist individually for both Lasso and Ridge regularization techniques.

b. Advantages

- i. ElasticNet has the flexibility to handle a range of datasets.
- ii. It is effective, for feature selection and regularization.
- iii. Unlike Lasso ElasticNet can handle multicollinearity (correlation among features)

c. Disadvantages

- i. The choice of the alpha parameter in ElasticNet requires tuning, which can be a bit tedious.
- ii. In cases where Ridge or Lasso methods clearly outperform ElasticNet may not perform well.
- iii. In summary ElasticNet serves as a regularization technique that combines the strengths of both Lasso and Ridge methods. It proves to be a tool for linear regression problems involving dimensional datasets. By striking a balance, between feature selection coefficient. Model complexity control.

3.5 METHOD DEEP LEARNING

3.5.1 Gated Recurrent Unit (GRU)

A Gated Recurrent Unit (GRU) is a type of architecture used in learning for tasks involving sequences, such, as natural language processing, speech recognition and time series prediction. Its purpose is to overcome limitations of recurrent neural networks (RNNs) like the vanishing gradient problem. Let's delve into the components and features of a GRU [67].

i. Recurrent Neural Networks (RNNs)

- a. RNNs are a category of networks designed to handle data by maintaining a hidden state that stores information from previous time steps.
- b. However conventional RNNs often struggle with capturing long range dependencies in sequences due, to the vanishing gradient problem.

ii. Gating Mechanism

- a. The GRU architecture introduces a gating mechanism that allows it to selectively update and reset the state at each time step.
- b. This gating mechanism improves the ability of GRUs to capture long term dependencies by controlling how information flows within the network [68].

iii. Key Components

- a. Hidden State (h_t); Similar, to RNNs GRUs maintain a state vector (h_t) that stores information from previous time steps.
- b. Update Gate (z_t); The update gate determines the balance, between retaining the state and incorporating new information from the current input. It considers both the current input and the previous hidden state.
- c. Reset Gate (r_t); The reset gate controls how much of the state should be disregarded or reset based on the current input.
- d. Candidate Hidden State ($\tilde{h}_t$); This intermediate representation combines the state and the current input, adjusted by the reset gate. It is utilized to calculate the state.
- e. Final Hidden State (h_t); The final hidden state is obtained by blending the candidate hidden state and the previous hidden state using the update gate. It represents the output of GRU at each time step.

iv. Mathematical Equations

- a. The update gate (z_t) and reset gate (r_t) are calculated using sigmoid activation

functions ensuring their values stay between 0 and 1.

- b. The candidate hidden state ($h_{\sim t}$) is computed using the tangent ($\tanh$) activation function.
 - c. The final hidden state (h_t) is determined by blending the state ($h_{(t-1)}$) and the candidate hidden state ($h_{\sim t}$) considering the values of the update gate (z_t).
- v. Advantages
- a. GRUs effectively capture long range dependencies, in data.
 - b. They are less prone to facing the vanishing gradient problem compared to RNNs.
 - c. GRUs are computationally efficient. Often require parameters than other RNN variants like LSTM (Long Short-Term Memory).
- vi. Applications
- a. GRUs find use in natural language processing tasks, such, as text generation, machine translation and sentiment analysis.
 - b. They are also utilized in speech recognition, time series forecasting and other sequence modeling tasks. To sum up a Gated Recurrent Unit (GRU) is a kind of architecture, for neural networks. Its purpose is to address the limitations of RNNs by incorporating gating mechanisms that regulate the flow of information, within the network. This feature makes GRUs highly efficient when it comes to analyzing data sequences that have long range interdependencies.

3.5.2 Long Short-Term Memory

Long Short-Term Memory is a network (RNN) architecture specifically designed to process and model sequences of data. LSTM networks excel in tasks that involve long range dependencies and find applications in natural language processing, speech recognition, time series analysis and other sequential data tasks. Let me explain the features and elements of an LSTM [69];

i. Recurrent Neural Networks (RNNs)

RNNs belong to a class of networks that handle data by maintaining a hidden state that captures information from previous time steps.

However traditional RNNs face challenges when it comes to learning and retaining information over sequences due to the vanishing gradient problem.

ii. Memory Cells

- a. To address the vanishing gradient problem LSTM networks, introduce memory cells of storing and retrieving information across sequences.
- b. These memory cells serve as building blocks within LSTM units [70].

iii. Key Components

- a. Hidden State (h_t); Similar, to RNNs LSTMs maintain a state vector (h_t) that encodes information from previous time steps.
- b. Cell State (c_t); LSTMs introduce a state vector called the cell state (c_t) which enables the network to retain and update information over time.

iv. Gates

- a. LSTMs incorporate three types of gates that regulate how information flows into and, out of the memory cell;
- b. Forget Gate (f_t); Determines what information from the time step should be forgotten or kept in the cell state.
- c. Input Gate (i_t); Controls how much new information from the input should be added to the cell state.
- d. Output Gate (o_t); Manages what information, from the cell state should be used to generate the state.

v. Mathematical Formulas

- a. LSTMs use sigmoid activation functions to calculate the values of the gates (f_t i_t . O_t) ensuring their values range between 0 and 1.
- b. The candidate cell state ($\tilde{c}_t$) is computed using the tangent ($\tanh$) activation function.
- c. The cell state (c_t) gets updated by combining the cell state, candidate cell state forgets gate and input gate.
- d. The hidden state (h_t) is calculated by applying the output gate to the cell state.

vi. Advantages

- a. LSTMs are highly effective, in capturing long term relationships in data making them valuable for tasks that involve temporal patterns.
- b. They can handle sequences of lengths. Adjust to various time scales.
- c. LSTMs are less susceptible to the vanishing gradient problem compared to RNNs.

vii. Applications

- a. LSTMs find application in natural language processing tasks like generating text translating languages and analyzing sentiment.
- b. They are also utilized in speech recognition forecasting time series data, recognizing handwriting and various other sequence modeling tasks.

In Long Short-Term Memory (LSTM) networks are a form of neural network architecture designed specifically to address the vanishing gradient problem and effectively model data sequences with long term dependencies. Their ability to capture patterns has made them a critical component, in several cutting-edge machine learning applications involving sequential data. The GRU cell is designed with two gates; the Update gate and the Reset gate. Similar, to LSTM gates these GRU gates are trained to filter out information while retaining what's useful. These gates are essentially vectors that contain values ranging from 0 to 1 which are multiplied with the input data and/or hidden state [71]. A gate vector value of 0 indicates that the corresponding data, in the input or hidden state is not significant and will be treated as zero. Conversely a gate vector value of 1 indicates that the corresponding data is crucial and will be utilized. As shown in the following figure.

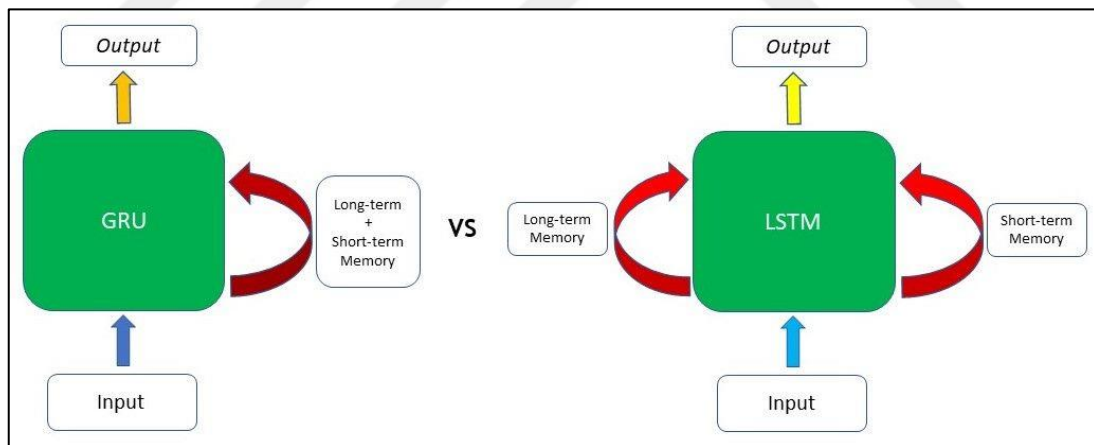


Figure 3.5: Long Short-Term Memory vs Gated Recurrent Unit (GRU) [72].

3.6 EVALUATING THE PERFORMANCE

When evaluating the performance of machine learning models there are metrics that are commonly used in classification tasks to determine how well a model is performing. Here I'll provide an explanation of each of these metrics;

a. F1 Score

The F1 score is a metric that takes into account both precision and recall to provide a value.

It becomes particularly useful when dealing with datasets that have imbalanced classes [73]. The F1 score can be calculated as the mean of precision. Recall using the following formula;

$$\text{F1 Score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

b. Precision Score

Precision measures the accuracy of predictions made by a model. It answers the question; "Out of all the predictions made by the model how many were actually correct?"

Precision can be calculated as follows [58]

$$\text{Precision} = \text{True Positives} / (\text{Positives} + \text{False Positives})$$

c. Recall Score

Recall, also known as sensitivity or true positive rate measures the ability of a model to identify all instances [74].

These metrics help you assess the performance of your classification model, in aspects. Depending on your problem and goals you may give priority to one metric over others. For instance, in a medical diagnosis scenario recall becomes crucial as missing a positive (failing to diagnose a disease) can lead to consequences. On the hand in a spam email filter precision is more important to avoid categorizing legitimate emails as spam. By utilizing these metrics along with the models' predictions and actual target values you can evaluate its performance. Make decisions regarding model tuning or selection.

Additionally, there are regression metrics commonly used to evaluate the performance of regression models, in machine learning. Here is a brief explanation of the metrics you are displaying;

i. Mean Absolute Error (MAE)

MAE calculates the difference, between the predicted values and the actual values [75]. It measures how much your predictions deviate from the true values considering each error equally important. Lower MAE values indicate better performance of the model.

The formula for MAE is usually; $\text{MAE} = (1/n) * \sum |\text{actual} - \text{Predicted}|$

Where "n" represents the number of data points.

ii. Mean Squared Error (MSE)

MSE computes the average of differences between predicted and actual values. By squaring errors MSE gives weight to errors making it sensitive to outliers. Lower MSE values indicate better model performance.

The formula for MSE is typically; $MSE = (1/n) * \sum (\text{predicted})^2$
where "n" represents the number of data points.

iii. R squared (R^2)

R squared is a measure that shows how much variation, in the variable (target) can be explained by independent variables (features) in your model. The range of R squared values is, from 0 to 1 with 1 indicating that the model explains all the variation in the data and 0 indicating that the model explains none.

R squared provides a measure of how your model fits the data.

Typically, R squared is calculated using the formula;

$$R^2 = 1 - (MSE(\text{model}) / MSE(\text{baseline}))$$

Here MSE (model) represents the squared error of your model and MSE (baseline) represents the squared error of a simple baseline model, such as predicting the mean value of the target, for all data points.

4. RESULTS AND DISCUSSION

After explaining the approach employed, we proceed to showcase the outcomes utilizing algorithms such, as ElasticNet, Extra Trees, GBC and RF. Subsequently we delve into an analysis and discussion of these findings. To ensure the credibility of our results we conducted tests on selected samples using a division of 30% for testing data and 70%, for training data. After that, the data was divided into 80% for training and 20% for testing algorithms deep learning LSTM and GRU. I have outlined a set of steps, for preparing your data and features before building and evaluating models in a project. These steps are essential for data preprocessing and feature selection. Let me provide some comments and clarification on each of these steps;

- i. Remove excessive features; It is a practice to drop features that do not provide meaningful information or are highly correlated with the target variable or other features. This simplifies your model making it easier to interpret and more efficient.
- ii. Use Clamping technique; Clamping helps in handling values in your dataset by reducing the impact of outliers on your models. You can set a threshold based on the value and median to ensure that extreme values do not overly influence your analysis.
- iii. Apply the function; Applying the function to skewed numeric features can make their distributions more symmetric, which is suitable for certain types of models. Using a threshold of 50 values as a criterion you can decide which features to transform, following an approach.
- iv. Reduce labels in features; This step involves decreasing the cardinality (number of labels), in features. High cardinality can cause dimensionality issues. Increased computational complexity. By combining labels into a single category, you simplify the encoding process.
- v. Examining the changes, in feature distributions before and after transformations is a technique to comprehend how each preprocessing step impacts the data. This aids in identifying issues and confirming that the transformations have the desired impact.
- vi. Converting features into a format that can be utilized by machine learning models is often accomplished through one hot encoding. It generates columns for each category making it suitable for ordinal categorical data.
- vii. Once preprocessing is complete the data is divided into training and test sets using sampling to ensure that both sets maintain the distribution of the target variable. With

data you can proceed with selecting and training machine learning models well as evaluating their performance.

- viii. These steps outline a thorough approach to data preprocessing and feature selection which're crucial, for constructing effective machine learning models. Make sure to document your process and consider utilizing validation techniques to more robustly assess model performance.

4.1 RESULTS OF MACHINE LEARNING ALGORITHMS

4.1.1 Extra Trees Classifier

We utilized the Extra Trees Classifier, from scikit learn to train a forest classifier based on your data. Let's break down what this code accomplishes;

`%%time`; This is a Jupyter Notebook or IPython command that measures the execution time of the code block.

`from import ExtraTreesClassifier`; This line imports the class from scikit learn, which is a classification method using ensemble learning.

`start = time.time()`; This line marks the starting point of the training process.

`model = ExtraTreesClassifier().fit(X_train, y_train)`; we create an instance of the ExtraTreesClassifier and fit it to your training data (X_train and y_train). This step trains the model using your provided training data.

`= time.time()`; This line records when the training process concludes.

`y_predictions = model.predict(X_test)`; After completing training we apply our trained model to make predictions on your test data (X_test). The predicted values are stored in y_predictions variable.

`end_predict = time.time()`; This line records when prediction process finishes.

By measuring the time taken for both training and prediction using `%%time` command as tracking start and end times you can evaluate how computationally efficient it is to train and predict with the Extra Trees Classifier, on your dataset.

To gain an understanding of the results it is important to assess the classifiers performance using suitable metrics and techniques. Simply measuring execution time is insufficient, for evaluating the quality of the model.

```
In [22]: %%time
from sklearn.ensemble import ExtraTreesClassifier
start = time.time()
model = ExtraTreesClassifier().fit(X_train,y_train)
end_train = time.time()
y_predictions = model.predict(X_test) # These are the predictions from the test data.
end_predict = time.time()

CPU times: total: 11.6 s
Wall time: 13.3 s
```

Figure 4.1: Extra Trees Classifier Code.

The CPU times you've given indicate the amount of CPU time utilized by the code block you executed;

CPU time; 11.6 seconds

Wall time; 13.3 seconds

The "total" CPU time includes the time used by all CPU cores in parallel while "wall time" takes into account any wait times for resources (I/O operations) and is typically slightly longer, than the CPU time.

These durations represent the combined length of training the model and making predictions, on the test data using that trained model. It's important to consider time when working with machine learning algorithms particularly if you intend to optimize your workflow or deal with datasets where execution time may be a crucial factor.

```
Accuracy: 94.85%
Recall: 94.85%
Precision: 94.86%
F1-Score: 94.86%
time to train: 12.31 s
time to predict: 0.99 s
total: 13.30 s
```

Figure 4.2: Extra Trees Classifier Results.

The ExtraTreesClassifier model has been analyzed and here is a breakdown of the findings;

Accuracy; 94.85%

Recall; 94.85%

Precision; 94.86%

F1 Score; 94.86%

Based on these metrics it appears that your model is performing well. With accuracy, recall,

precision and F1 score all hovering around 94.85% to 94.86% it indicates that the model is making predictions, on your test data.

Training time; 12.31 seconds

Prediction time; 0.99 seconds

These timing results indicate that training the model took 12.31 seconds while making predictions on the test data only required approximately 0.99 seconds.

Time taken; Around 13.30 seconds

The total time reflects the combined duration of both training and prediction phases summing up to 13.30 seconds.

These outcomes provide insights into both the performance of your model and its computational efficiency aspects. It's crucial to strike a balance, between achieving high model performance and reasonable computational time in alignment with your application requirements – in this case it seems like you have successfully achieved equilibrium.

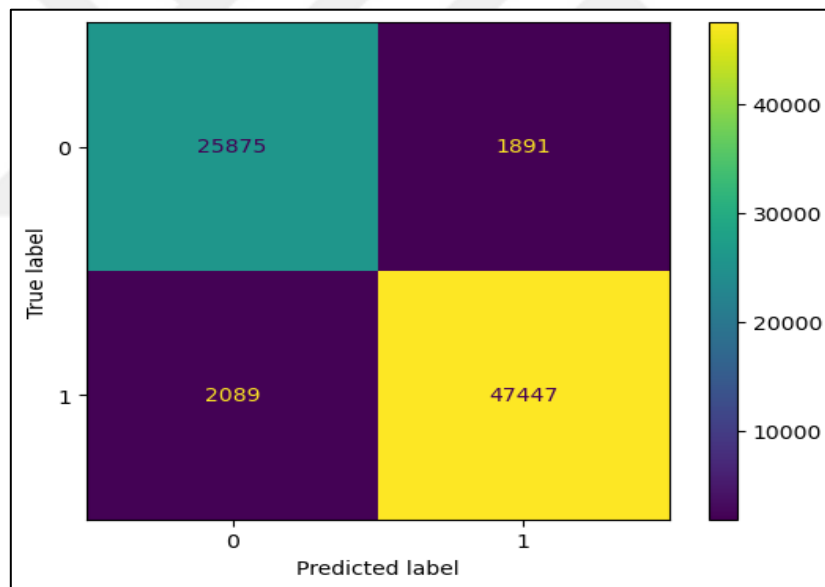


Figure 4.3: Extra Trees Classifier Confusion Matrix Display.

4.1.2 Random Forest Classifier

To train a forest classifier we utilize scikit learns library with the following settings;

n_estimators; 100 (the number of trees, in the forest).

n_jobs; 1 (indicating that all available CPU cores are utilized for processing).

random_state; 0 (ensuring reproducibility).

bootstrap; True (using bootstrap samples for tree construction).

Lets break down what this code block accomplishes;

- a. %%time; This command, typically used in Jupyter Notebook or IPython measures the execution time of the code block.
- b. The line "from sklearn.ensemble import RandomForestClassifier" imports the class from scikit learn.
- c. "start = time.time()" records the starting time of the training process.
- d. By creating an instance of RandomForestClassifier with the specified parameters and fitting it to your training data (X_train and y_train) you train the forest classifier on your data.
- e. "end_train = time.time()" records the end time of the training process.
- f. "y_predictions = model.predict(X_test)" employs the trained forest classifier to make predictions, on your test data (X_test).

The predicted values are saved in the variable y_predictions.

The line end_predict = time.time() is used to mark the ending time of the prediction process. To evaluate the efficiency of training and prediction for the RandomForestClassifier on your dataset you can measure the time it takes by using the %%time magic command and noting down the start and end times.

```
In [26]: %%time
         from sklearn.ensemble import RandomForestClassifier
         start = time.time()
         model = RandomForestClassifier(n_estimators = 100, n_jobs=-1, random_state=0, bootstrap=True).fit(X_train, y_train)
         end_train = time.time()
         y_predictions = model.predict(X_test) # These are the predictions from the test data.
         end_predict = time.time()

CPU times: total: 1min 32s
Wall time: 6.48 s
```

Figure 4.4: Random Forest Classifier Code.

The CPU timings you provided show how much CPU time was used by the code block you executed;

CPU time; 1 minute and 32 seconds

Wall time; 6.48 seconds

Lets break down what these timings mean;

CPU time; This represents the accumulated CPU time used by the code. It includes the time used by all CPU cores in parallel. Reflects the computational work done during training and prediction.

Wall time; This is the time that passed on the clock (the time observed by a user). It includes CPU time as any waiting or I/O operations. The wall time is significantly shorter, than the CPU time indicating that training and prediction processes are efficiently parallelized and

involve waiting or I/O.

Training the RandomForestClassifier took 1 minute and 32 seconds in terms of CPU time.

Making predictions, on the test data took 6.48 seconds in terms of wall time.

```
Accuracy: 95.07%
Recall: 95.07%
Precision: 95.09%
F1-Score: 95.08%
time to train: 6.35 s
time to predict: 0.13 s
total: 6.48 s
```

Figure 4.5: Random Forest Classifier Results.

Based on these metrics it seems like your RandomForestClassifier model is performing well. It has achieved levels of accuracy, recall, precision and F1 score ranging from 95.07%, to 95.09%. This suggests that the model is making predictions when tested.

Training Time; 6.35 seconds

Prediction Time; 0.13 seconds

The time it took to train the model was 6.35 seconds while the prediction process, on the test data only took around 0.13 seconds.

Total Time; 6.48 seconds

When we add up the training and prediction times, we get a time of about 6.48 seconds. These results indicate that both the training and prediction steps are optimized for efficiency and speed.

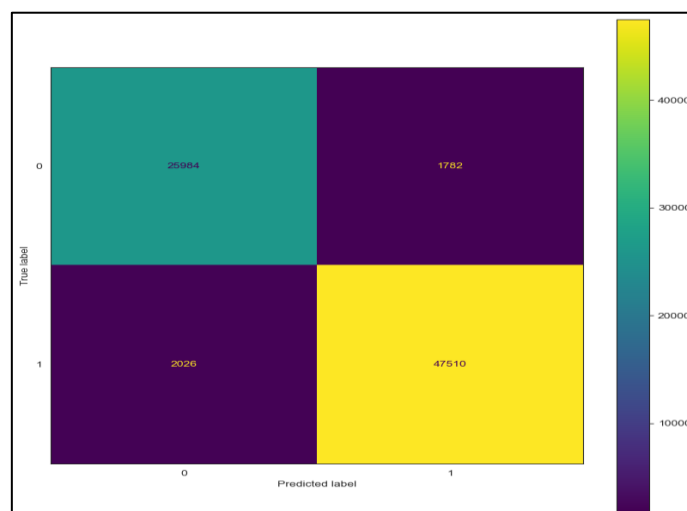


Figure 4.6: Random Forest Classifier Confusion Matrix Display.

4.1.3 Gradient Boosting Classifier

We are utilizing the GradientBoostingClassifier from the scikit learn library to train a classifier using gradient boosting. Let me break down what this code accomplishes;

%%time; This command, specific, to Jupyter Notebook or IPython measures the execution time of the code block.

from sklearn.ensemble import GradientBoostingClassifier; In this line we import the GradientBoostingClassifier class from scikit learn which's a well-known ensemble method used for classification tasks.

start = time.time(); This line captures the starting time of the training process.

model = GradientBoostingClassifier().fit(X_train, y_train); Here we create an instance of the GradientBoostingClassifier. Train it with our training data (X_train and y_train). This step involves training our gradient boosting classifier on our provided data.

end_train = time.time(); We record the ending time of the training process with this line.

y_predictions = model.predict(X_test); After completing training we utilize our trained gradient boosting classifier to make predictions, on our test data (X_test). The predicted values are stored in y_predictions.

end_predict = time.time(); This line records when the prediction process concludes.

By measuring both training and prediction times using %%time and capturing start and end times you can evaluate how computationally efficient it is to train and predict with the GradientBoostingClassifier on your dataset.

Make sure to conduct an analysis of the results by assessing the classifiers performance using metrics and techniques. Simply measuring the execution time is insufficient, for gaining an understanding of the model's quality.

```
In [29]: %%time
from sklearn.ensemble import GradientBoostingClassifier
start = time.time()
model = GradientBoostingClassifier().fit(X_train,y_train)
end_train = time.time()
y_predictions = model.predict(X_test) # These are the predictions from the test data.
end_predict = time.time()

CPU times: total: 1min 14s
Wall time: 1min 28s
```

Figure 4.7: Gradient Boosting Classifier Code.

The CPU timings you have provided indicate the CPU usage, by the code block you ran;
CPU time; 1 minute and 14 seconds

Wall time; 1 minute and 28 seconds

Let's break down what these timings represent;

CPU time; This refers to the amount of CPU time utilized by the code. It includes the time used by all CPU cores in parallel reflecting the work carried out during training and prediction.

Wall time; This represents the duration that passed on a clock (the kind you would observe as a user). It incorporates CPU time. Also considers any waiting or input/output operations. The wall time is slightly longer than the CPU time indicating that there might be some delays related to waiting or input/output activities, during execution.

```
Accuracy: 93.40%  
Recall: 93.40%  
Precision: 93.39%  
F1-Score: 93.39%  
time to train: 88.28 s  
time to predict: 0.09 s  
total: 88.38 s
```

Figure 4.8: Gradient Boosting Classifier Results.

The results of your GradientBoostingClassifier model look quite promising with an accuracy, recall, precision and F1 score all hovering around 93.39% to 93.40%. This indicates that the model is effectively making predictions, on your test data.

When it came to training the GradientBoostingClassifier model it took 88.28 seconds while predicting on the test data was significantly faster, at 0.09 seconds.

Combining the training and prediction times gives us a time of 88.38 seconds. It's worth noting that training the GradientBoostingClassifier requires power and takes longer compared to other models you've experimented with.

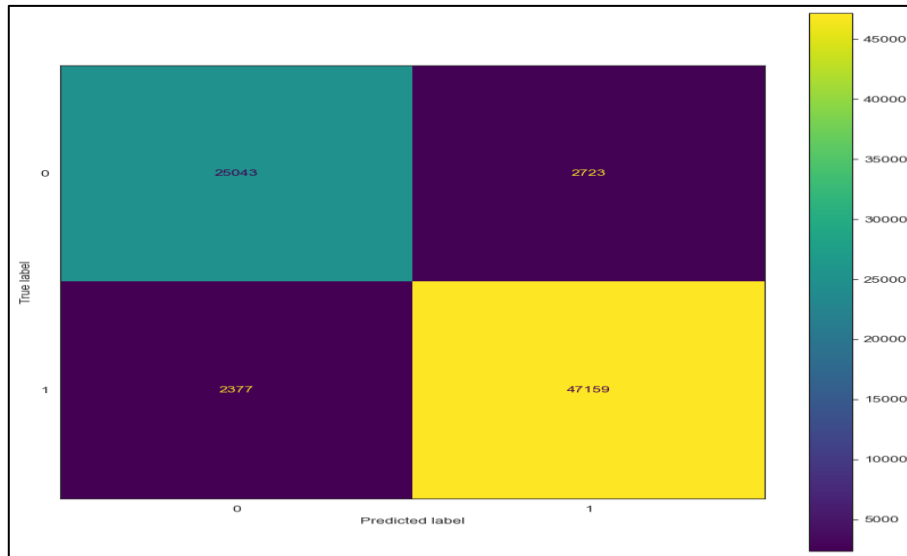


Figure 4.9: Gradient Boosting Classifier Confusion Matrix Display.

4.1.4 Compare The Machine Learning Algorithms Results

Out[32]:

	Accuracy	Recall	Precision	F1-Score	time to train	time to predict	total time
Extra Trees	94.85%	94.85%	94.86%	94.86%	12.3	1.0	13.3
Random Forest	95.07%	95.07%	95.09%	95.08%	6.4	0.1	6.5
Gradient Boosting Classifier	93.40%	93.40%	93.39%	93.39%	88.3	0.1	88.4

Figure 4.10: Compare the Results.

The comparison shows that the random forest algorithm is superior to other algorithms in terms of time and accuracy.

4.1.5 Linear Model ElasticNet

We are utilizing scikit learns ElasticNet model for regression. Let me break down the functionality of this code;

%%time; This represents a Jupyter Notebook or IPython command that measures the execution time of the code block.

from sklearn.linear_model import ElasticNet; This line imports the ElasticNet class from scikit learn which is a regression model that incorporates both L1 (Lasso) and L2 (Ridge) penalties, for regularization.

start = time.time(); This line records the time of the training process.

model = ElasticNet(alpha=0.1, l1_ratio=0.5).fit(X_train, y_train); Here an instance of the ElasticNet model is created with hyperparameters (alpha and l1_ratio). It's fitted to your training data (X_train and y_train). This step trains the ElasticNet model on your training data.

= time.time()); This line records the end time of the training process.

y_predictions = model.predict(X_test); After training you can utilize the trained ElasticNet model to make predictions, on your test data (X_test). The predicted values will be stored in y_predictions.

end_predict = time.time(); This line records the end time of the prediction process.

You can evaluate the efficiency of training and prediction for the ElasticNet model, on your dataset by using the %%time magic command to measure the time it takes to train and predict and comparing the start and end times.

```
In [33]: %%time
from sklearn.linear_model import ElasticNet
import time

start = time.time()
model = ElasticNet(alpha=0.1, l1_ratio=0.5).fit(X_train, y_train)
end_train = time.time()

y_predictions = model.predict(X_test) # These are the predictions from the test data.

end_predict = time.time()

# Print the training and prediction times
print("Training Time:", end_train - start, "seconds")
print("Prediction Time:", end_predict - end_train, "seconds")
```

Figure 4.11: ElasticNet Code.

Here's a breakdown of the findings;

Time taken for training; 0.108 seconds

Time taken for prediction; 0.0 seconds

In addition, the overall CPU time and wall time, for executing the code block are as follows;

CPU time; 46.9 milliseconds

Wall time; 108 milliseconds

These findings suggest that the ElasticNet regression model requires very little training time, approximately 0.108 seconds and the prediction time is negligible at practically 0 seconds. The total execution time for running the code block is also relatively short.

The ElasticNet model is recognized for its efficiency particularly when compared to models like gradient boosting and neural networks. It is well suited for tasks involving regression with regularization and, in this case, it performs efficiently and swiftly.

```
Training Time: 0.10763263702392578 seconds
Prediction Time: 0.0 seconds
CPU times: total: 46.9 ms
Wall time: 108 ms
```

Figure 4.12: ElasticNet Results.

The regression metrics, for the model are as follows;

Mean Absolute Error (MAE); 0.2945

This means that on average the models' predictions have an error of around 0.2945 units when compared to the target values. It provides insight into the magnitude of errors in the predictions.

Mean Squared Error (MSE); 0.1269

The MSE is another measure of prediction accuracy. It represents the average of the squared differences between your predictions and the actual values. In this case the average squared error is 0.1269.

R squared (R^2); 0.4485

The R squared value indicates how much of the variability in the target variable can be explained by your model. An R squared value of 0.4485 suggests that your model explains about 44.85% of that variability. Remember that R squared values range from 0, to 1 with 1 indicating a fit.

```
In [34]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Assuming you have already loaded y_test and y_predictions
# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_predictions)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_predictions)

# Calculate R-squared ( $R^2$ )
r_squared = r2_score(y_test, y_predictions)

# Print the regression metrics
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("R-squared ( $R^2$ ):", r_squared)

Mean Absolute Error (MAE): 0.29449084930696606
Mean Squared Error (MSE): 0.1269341038074679
R-squared ( $R^2$ ): 0.4485253828492769
```

Figure 4.13: The Regression Metrics.

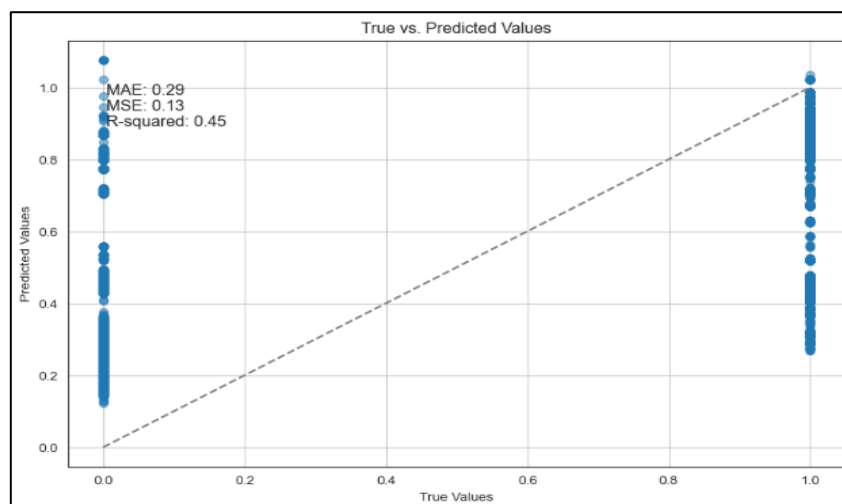


Figure 4.14: Plotting the True Values vs. Predicted Values.

4.2 RESULTS OF DEEP LEARNING ALGORITHMS

4.2.1 Gated Recurrent Unit (GRU)

To develop a network model using TensorFlow/Keras for a sequence, to sequence problem we utilize an architecture consisting of GRU layers followed by a dense layer for multiclass classification. Here's an overview of the code;

Model Construction;

We establish a model by invoking Sequential().

We incorporate three GRU (Gated Recurrent Unit) layers, each containing 64, 64 and 32 units respectively. These layers are responsible for handling sequence modeling.

Additionally we include a layer with 10 units. Apply the softmax activation function to facilitate multiclass classification.

The model is compiled using the loss function RMSprop optimizer with a learning rate of 0.001 and accuracy as the metric.

Data Reshaping;

networks, including GRUs require input data to be formatted in a manner. To meet this requirement we reshape our training and testing data dimensions to (samples, time steps features) utilizing reshape().

By employing reshape() we create X_train_resaped and X_test_resaped variables that possess the shape, for feeding into the model.

Model Training;

We instantiate the model using the build_model() function.

Training commences while tracking elapsed time with start = time.time().The training of the model involves using the training data (X_train_resaped) and labels (y_train) for 500 epochs, where each epoch consists of a batch size of 250 samples.

During the training process, which is executed through model.fit() the models weights are optimized based on a loss function and optimizer. This is an approach, in learning to progressively enhance the models performance on the training data.

Once the training is complete you can assess how well the model performs, make predictions and analyze its training history to track changes in loss and accuracy across epochs. It may also be useful to save the trained model, for use or further evaluations.

```
In [33]: loss, accuracy = model.evaluate(X_test_resaped, y_test)
# loss, accuracy, f1s, precision, recall = model.evaluate(X_test_resaped, y_test)
end_predict = time.time()
model_performance.loc['GRU (Keras)'] = [accuracy, accuracy, accuracy, accuracy, end_train-start,end_predict-end_train,end_predict-end_predict]

515/515 [=====] - 2s 2ms/step - loss: 0.1285 - accuracy: 0.9708
```

Figure 4.15: Gated Recurrent Unit Results.

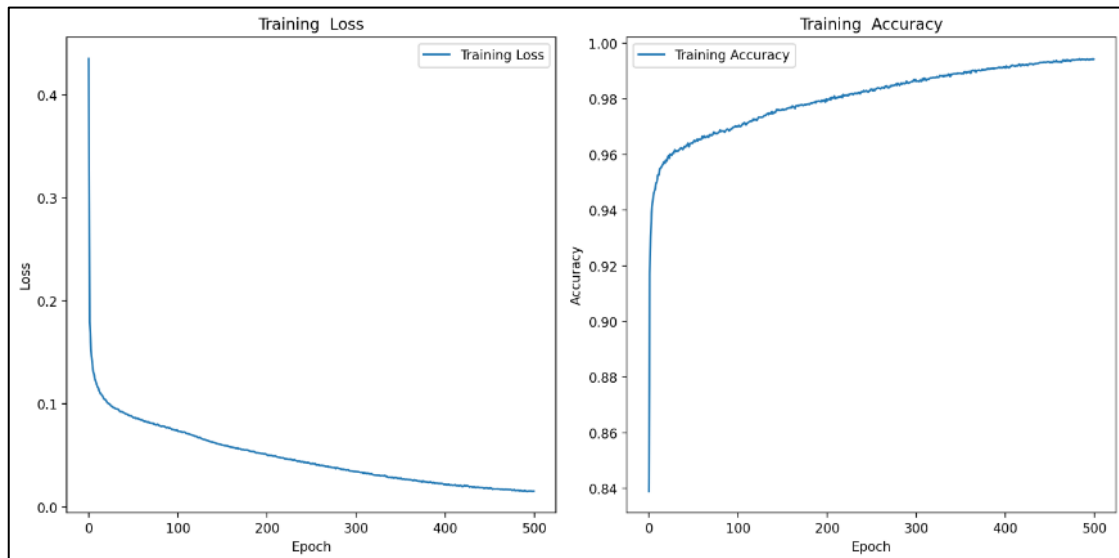


Figure 4.16: Plot Training Accuracy and Plot Training Loss to GRU.

4.2.2 Long Short-Term Memory

To create and train a network (RNN) model, with LSTM layers for a multiclass classification problem follow these steps;

Model Architecture; Construct the model using the Keras Sequential API. It should consist of three LSTM layers and one Dense layer for classification. The first LSTM layer should have 64 units and return sequences. The second LSTM layer should also have 64 units and return sequences. The third LSTM layer should have 32 units and return sequences. The output layer should be a layer with 10 units using softmax activation.

Model Compilation; Compile the model using 'sparse_categorical_crossentropy' as the loss function RMSprop optimizer with a learning rate of 0.001 and accuracy as the evaluation metric.

Data Preparation; Reshape the input data X_train to have dimensions (number of samples, time steps features). In this case reshape it to (number of samples 1 56). Apply the reshaping process to the test data X_test.

Model Training; Train the model using the training data (X_train_resaped) and labels (y_train) for 500 epochs with a batch size of 250. Track the training progress, in the history variable.

The duration required for training the model is calculated by utilizing the time module.

```
In [39]: #Evaluate the neural network
loss, accuracy = model.evaluate(X_test_resaped, y_test)
# loss, accuracy, f1s, precision, recall = model.evaluate(X_test_resaped, y_test)
end_predict = time.time()
model_performance.loc['LSTM (Keras)'] = [accuracy, accuracy, accuracy, accuracy, end_train-start, end_predict-end_train, end_predict-start, end_predict-end_predict]

515/515 [=====] - 2s 2ms/step - loss: 0.1090 - accuracy: 0.9716
```

Figure 4.17: Long Short-Term Memory Results.

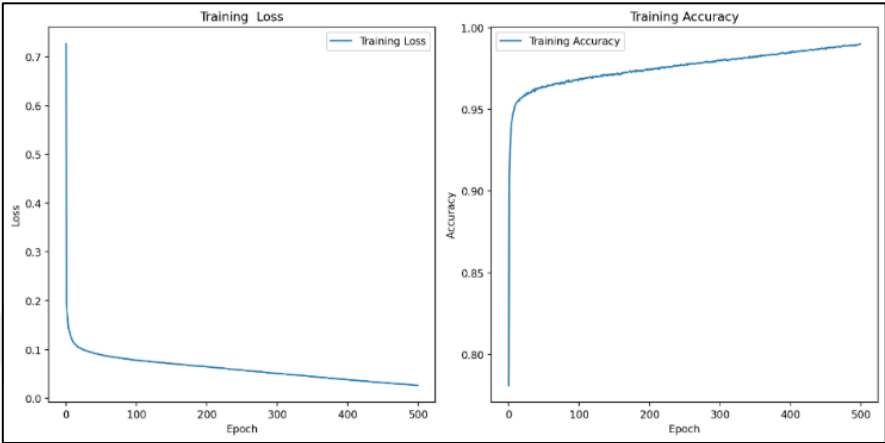


Figure 4.18: Long Short-Term Memory Results.

4.2.3 Compare The Deep Learning Algorithms Results

The provided table displays performance metrics for two types of neural network (RNN) architectures, namely GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) implemented using Keras. Here's an explanation of each column, in the table;

Accuracy; This column represents the accuracy of the models, measured as a percentage. It indicates how the models' classified samples within the test dataset.

Recall; The recall metric measures how effectively the model identifies positive samples, specifically true positives out of all actual positive samples (true positives + false negatives). It is also expressed as a percentage.

Precision; Precision is a metric that quantifies how well the model correctly identifies samples out of all predicted samples (true positives + false positives). It is also expressed as a percentage.

F1 Score; The F1 Score is a measure that combines precision and recall using their mean. It provides an assessment of model performance. Is also expressed as a percentage.

Time to Train; In this column we can see the duration it took to train each model in seconds.

Time to Predict; This column indicates the time it took for the models to make predictions, on test data after completing their training phase, measured in seconds. The total time includes the

combined duration of training and prediction representing the time spent on both tasks. Based on the information provided in the table it seems that the LSTM model performs better, than the GRU model in terms of accuracy, recall, precision and F1 Score. However, it took time to train the GRU model compared to the LSTM model. The choice between LSTM and GRU often depends on factors such as the problem. Dataset at hand as well as considerations regarding computational resources and training time. Both models exhibit performance with accuracy, recall, precision and F1 Score scores hovering around 97%. It's important to evaluate these metrics within your problems context and determine which tradeoffs are acceptable, for your application.

Out[41]:

	Accuracy	Recall	Precision	F1-Score	time to train	time to predict	total time
GRU (Keras)	97.08%	97.08%	97.08%	97.08%	1231.7	1.8	1233.5
LSTM (Keras)	97.16%	97.16%	97.16%	97.16%	723.1	2.1	725.2

Figure 4.19: The Deep Learning Algorithms Compare.

4.3 COMPARISON WITH PREVIOUS STUDIES

Table 4.1: Comparison of Previous Studies.

IDS	Model(s)	Dataset	Accuracy (%)
AL-Hawawreh [37]	Autoencoder	UNSW-NB15	92.4
Zhou [38]	gradient-boosted trees	UNSW-NB15	91.22
	Knn		91.9
	DT		92.29
	logistic regression		90.35
	gaussianNB		92.52
	SVM		92.32
TAMA [39]	Rotation	UNSW-NB15	72.52
	Forest+bagging		
This study	ExtraTreesClassifier	UNSW-NB15	94.85%
	RandomForestClassifier		95.07%
	GradientBoostingClassifier		93.40%
	GRU		97.08%
	LSTM		97.16%

5. CONCLUSION

This research used a total of six algorithms, four of which were derived from the field of machine learning, namely ElasticNet, Extra Trees, GBC, and RF. The other two methods were sourced from the realm of deep learning, namely LSTM and GRU. The findings demonstrate that the ElasticNet model had higher performance in terms of both accuracy and time, as shown by the results. Specifically, it outperformed both the GRU and LSTM models in terms of time efficiency. On the other hand, when considering precision, recall, precision, and F1-score, the LSTM model outperformed the GRU model. The Long Short-Term Memory (LSTM) model has a higher rate of training efficiency compared to the Gated Recurrent Unit (GRU) model. The selection between LSTM and GRU is contingent upon several aspects, including the nature of the issue, characteristics of the dataset, and the computing resources at hand. In terms of performance, both LSTM and GRU models exhibit comparable accuracy, recall, precision, and F1 score, all of which are at a level of 97%. The random forest algorithm is a popular machine learning technique that combines many decision trees to make predictions. Among the models under consideration, the Random Forest model has notable performance in terms of accuracy, recall, precision, and F1 score. Additionally, this model exhibits notable training and prediction times, making it a formidable candidate for achieving a favorable trade-off between performance and efficiency. The Extra Trees algorithm has shown strong performance in terms of accuracy and other evaluation measures. Despite the increased training time compared to Random Forest, the efficiency of the model is still maintained. The Gradient Boosting Classifier is a machine learning algorithm that demonstrates high accuracy in classification tasks. However, it is important to note that this algorithm requires much longer training time compared to other classifiers. When evaluating the trade-off between cost and improvement, the significance of performance becomes a crucial factor to consider. The current study has conducted a comparative analysis between our work and earlier research, revealing that the set of algorithms proposed in this study exhibits greater performance. These models have the potential to undergo testing using an alternative data source. Hence, the models possess the capability of being generalized, hence enhancing the intrusion detection procedure and facilitating the use of an integrated system including interconnected components that collaboratively oversee and detect illegal actions occurring either inside the network or on a designated host.

REFERENCES

- [1] I. Lee, "Internet of Things (IoT) cybersecurity: Literature review and IoT cyber risk management," *Futur. internet*, vol. 12, no. 9, p. 157, 2020.
- [2] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, "Internet of things (IoT) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5G-IoT scenarios," *Ieee Access*, vol. 8, pp. 23022–23040, 2020.
- [3] W. Gong, "The Internet of Things (IoT): What is the potential of the internet of things (IoT) as a marketing tool?" University of Twente, 2016.
- [4] M. Rumez, D. Grimm, R. Kriesten, and E. Sax, "An overview of automotive service-oriented architectures and implications for security countermeasures," *IEEE access*, vol. 8, pp. 221852–221870, 2020.
- [5] M. Vishwakarma and N. Kesswani, "DIDS: A Deep Neural Network based real-time Intrusion detection system for IoT," *Decis. Anal. J.*, vol. 5, p. 100142, 2022.
- [6] A. Mahfouz, A. Abuhussein, D. Venugopal, and S. Shiva, "Ensemble classifiers for network intrusion detection using a novel network attack dataset," *Futur. Internet*, vol. 12, no. 11, p. 180, 2020.
- [7] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.
- [8] K. Chen, A. R. Junuthula, I. K. Siddhrau, Y. Xu, and H. J. Chao, "SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane," in *2016 IEEE conference on communications and network security (CNS)*, pp. 28–36, 2016.
- [9] H. T. Nguyen and K. Franke, "Adaptive Intrusion Detection System via online machine learning," in *2012 12th international conference on hybrid intelligent systems (HIS)*, pp. 271–277, 2012.
- [10] A. K. Nair, E. D. Raj, and J. Sahoo, "A robust analysis of adversarial attacks on federated learning environments," *Comput. Stand. Interfaces*, p. 103723, 2023.
- [11] O. Vermesan *et al.*, "Internet of things strategic research roadmap," in *Internet of things-global technological and societal trends from smart environments and spaces to green ICT*, River Publishers, pp. 9–52, 2022.

- [12] K. Kimani, V. Oduol, and K. Langat, "Cyber security challenges for IoT-based smart grid networks," *Int. J. Crit. Infrastruct. Prot.*, vol. 25, pp. 36–49, 2019.
- [13] N. Moustafa and J. 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)*, pp. 1–6, 2015.
- [14] Z. Zhang, J. Yan, S. Liu, Z. Lei, D. Yi, and S. Z. Li, "A face antispoofing database with diverse attacks," in *2012 5th IAPR international conference on Biometrics (ICB)*, pp. 26–31, 2012.
- [15] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home IoT devices," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 9042–9053, 2019.
- [16] B. Madhu, M. V. G. Chari, R. Vankdothu, A. K. Silivery, and V. Aerranagula, "Intrusion detection models for IOT networks via deep learning approaches," *Meas. Sensors*, vol. 25, p. 100641, 2023.
- [17] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," *Inf. Secur. J. A Glob. Perspect.*, vol. 25, no. 1–3, pp. 18–31, 2016.
- [18] M. A. Umar and C. Zhanfang, "Effects of Feature Selection and Normalization on Network Intrusion Detection," *Authorea Prepr.*, 2023.
- [19] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.
- [20] M. Hammad, W. El-Medany, and Y. Ismail, "Intrusion detection system using feature selection with clustering and classification machine learning algorithms on the unsw-nb15 dataset," in *2020 international conference on innovation and intelligence for informatics, computing and technologies (3ICT)*, pp. 1–6, 2020.
- [21] M. K. Gupta and P. Chandra, "A comprehensive survey of data mining," *Int. J. Inf. Technol.*, vol. 12, no. 4, pp. 1243–1257, 2020.
- [22] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *Ieee Access*, vol. 7, pp. 41525–41550, 2019.

- [23] T. M. Fernandez-Carames and P. Fraga-Lamas, "A review on the application of blockchain to the next generation of cybersecure industry 4.0 smart factories," *Ieee Access*, vol. 7, pp. 45201–45218, 2019.
- [24] S. Laqtib, K. El Yassini, and M. L. Hasnaoui, "A deep learning methods for intrusion detection systems based machine learning in MANET," in *Proceedings of the 4th international conference on smart city applications*, pp. 1–8, 2019.
- [25] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, no. 1–2, pp. 18–28, 2009.
- [26] J. Gaspar, T. Cruz, C.-T. Lam, and P. Simões, "Smart Substation Communications and Cybersecurity: A Comprehensive Survey," *IEEE Commun. Surv. Tutorials*, 2023.
- [27] H. He and J. Yan, "Cyber-physical attacks and defences in the smart grid: a survey," *IET Cyber-Physical Syst. Theory Appl.*, vol. 1, no. 1, pp. 13–27, 2016.
- [28] K. Siddique, "A game theoretic framework to secure cyber physical systems (CPS) against cyber attacks," 2018.
- [29] A. L. Buczak, B. Baugher, E. Guven, L. Moniz, S. M. Babin, and J.-P. Chretien, "Prediction of Peaks of Seasonal Influenza in Military Health-Care Data: Supplementary Issue: Big Data Analytics for Health," *Biomed. Eng. Comput. Biol.*, vol. 7, p. BECB-S36277, 2016.
- [30] F. Abri, S. Siامي-Namini, M. A. Khanghah, F. M. Soltani, and A. S. Namin, "Can machine/deep learning classifiers detect zero-day malware with high accuracy?," in *2019 IEEE international conference on big data (Big Data)*, pp. 3252–3259, 2019.
- [31] M. F. Elrawy, A. I. Awad, and H. F. A. Hamed, "Intrusion detection systems for IoT-based smart environments: a survey," *J. Cloud Comput.*, vol. 7, no. 1, pp. 1–20, 2018.
- [32] P. Samani, "Synergies and gaps between circularity assessment and Life Cycle Assessment (LCA)," *Sci. Total Environ.*, p. 166611, 2023.
- [33] K. Steverson, C. Carlin, J. Mullin, and M. Ahiskali, "Cyber intrusion detection using natural language processing on windows event logs," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*, pp. 1–7, 2021.
- [34] V. N. S. S. Chimakurthi, "The Challenge of Achieving Zero Trust Remote Access in Multi-Cloud Environment," *ABC J. Adv. Res.*, vol. 9, no. 2, pp. 89–102, 2020.

- [35] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Appl. Sci.*, vol. 9, no. 20, p. 4396, 2019.
- [36] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *J. big data*, vol. 2, no. 1, pp. 1–21, 2015.
- [37] A.-H. Muna, N. Moustafa, and E. Sitnikova, "Identification of malicious activities in industrial internet of things based on deep learning models," *J. Inf. Secur. Appl.*, vol. 41, pp. 1–11, 2018.
- [38] Y. Zhou, M. Han, L. Liu, J. S. He, and Y. Wang, "Deep learning approach for cyberattack detection," in *IEEE INFOCOM 2018-IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pp. 262–267, 2018.
- [39] B. A. Tama, M. Comuzzi, and K.-H. Rhee, "TSE-IDS: A two-stage classifier ensemble for intelligent anomaly-based intrusion detection system," *IEEE access*, vol. 7, pp. 94497–94507, 2019.
- [40] U. Noreen, A. Bounceur, and L. Clavier, "A study of LoRa low power and wide area network technology," in *2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, pp. 1–6, 2017.
- [41] H. Guo, G. Xie, and J. He, "Hardware design of a wireless video processing module based on TMS320DM6446," in *2010 3rd International Congress on Image and Signal Processing*, vol. 8, pp. 3587–5390, 2010.
- [42] D. Ucuz, "Comparison of the IoT platform vendors, microsoft Azure, Amazon web services, and Google cloud, from users' perspectives," in *2020 8th international symposium on digital forensics and security (ISDFS)*, pp. 1–4, 2020.
- [43] A. Abadie, M. Roux, S. Chowdhury, and P. Dey, "Interlinking organisational resources, AI adoption and omnichannel integration quality in Ghana's healthcare supply chain," *J. Bus. Res.*, vol. 162, p. 113866, 2023.
- [44] B. Alamri, I. T. Javed, and T. Margaria, "A GDPR-compliant framework for IoT-based personal health records using blockchain," in *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, 2021.
- [45] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, 2019.

- [46] A. Z. M. Alhowaide, "Towards autonomous IoT IDSs using ensemble learning and feature selection methods." Memorial University of Newfoundland, 2021.
- [47] A. Walden, M. Garza, and L. Rasmussen, "Best Practices for Research Data Management," in *Clinical Research Informatics*, Springer, pp. 255–289, 2023.
- [48] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Comput. networks*, vol. 51, no. 12, pp. 3448–3470, 2007.
- [49] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 16–24, 2013.
- [50] A. Patel, H. Alhussian, J. M. Pedersen, B. Bounabat, J. C. Júnior, and S. Katsikas, "A nifty collaborative intrusion detection and prevention architecture for smart grid ecosystems," *Comput. Secur.*, vol. 64, pp. 92–109, 2017.
- [51] M. Saxena, *Next Generation Intelligent Network Intrusion Prevention System*. Lulu. com, 2017.
- [52] D. Teixeira, L. Assunção, T. Pereira, S. Malta, and P. Pinto, "OSSEC IDS extension to improve log analysis and override false positive or negative detections," *J. Sens. Actuator Networks*, vol. 8, no. 3, p. 46, 2019.
- [53] P. Maniriho, E. Niyigaba, Z. Bizimana, V. Twiringiyimana, L. J. Mahoro, and T. Ahmad, "Anomaly-based intrusion detection approach for IoT networks using machine learning," in *2020 international conference on computer engineering, network, and intelligent multimedia (CENIM)*, pp. 303–308, 2020.
- [54] A. Blaise, M. Bouet, V. Conan, and S. Secci, "Detection of zero-day attacks: An unsupervised port-based approach," *Comput. Networks*, vol. 180, p. 107391, 2020.
- [55] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–8, 2019.
- [56] X. Feng, X. Zhu, Q.-L. Han, W. Zhou, S. Wen, and Y. Xiang, "Detecting vulnerability on IoT device firmware: A survey," *IEEE/CAA J. Autom. Sin.*, vol. 10, no. 1, pp. 25–41, 2022.
- [57] K. Nova, "Security and Resilience in Sustainable Smart Cities through Cyber Threat Intelligence," *Int. J. Inf. Cybersecurity*, vol. 6, no. 1, pp. 21–42, 2022.
- [58] A. K. H. Hasan, "Characteristics of machine learning algorithms to improve student evaluation." Altınbaş Üniversitesi/Lisansüstü Eğitim Enstitüsü, 2022.

- [59] R. Mbuva, P. Zondo, A. Mauda, and T. Marwala, "Predicting Higher Education Throughput in South Africa Using a Tree-Based Ensemble Technique," *arXiv Prepr. arXiv2106.06805*, 2021.
- [60] B. S. Bhati and C. S. Rai, "Ensemble based approach for intrusion detection using extra tree classifier," in *Intelligent Computing in Engineering: Select Proceedings of RICE 2019*, pp. 213–220, 2020.
- [61] A. Berrouachedi, R. Jaziri, and G. Bernard, "Deep cascade of extra trees," in *Trends and Applications in Knowledge Discovery and Data Mining: PAKDD 2019 Workshops, BDM, DLKT, LDRC, PAISI, WeL, Macau, China, April 14–17, 2019, Revised Selected Papers 23*, pp. 117–129, 2019.
- [62] S. Peter, F. Diego, F. A. Hamprecht, and B. Nadler, "Cost efficient gradient boosting," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.
- [63] D. Upadhyay, J. Manero, M. Zaman, and S. Sampalli, "Gradient boosting feature selection with machine learning classifiers for intrusion detection on power grids," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 1, pp. 1104–1116, 2020.
- [64] O. Demir-Kavuk, M. Kamada, T. Akutsu, and E.-W. Knapp, "Prediction using step-wise L1, L2 regularization and feature selection for small data sets with large number of features," *BMC Bioinformatics*, vol. 12, pp. 1–10, 2011.
- [65] <https://analyticsarora.com/the-most-important-things-you-need-to-know-about-elastic-net/>
- [66] A. F. M. Agarap, "A neural network architecture combining gated recurrent unit (GRU) and support vector machine (SVM) for intrusion detection in network traffic data," in *Proceedings of the 2018 10th international conference on machine learning and computing*, pp. 26–30, 2018.
- [67] W. Zheng and G. Chen, "An accurate GRU-based power time-series prediction approach with selective state updating and stochastic optimization," *IEEE Trans. Cybern.*, vol. 52, no. 12, pp. 13902–13914, 2021.
- [68] J. Oruh, S. Viriri, and A. Adegun, "Long short-term memory recurrent neural network for automatic speech recognition," *IEEE Access*, vol. 10, pp. 30069–30079, 2022.
- [69] R. Al-Shabandar, A. Jaddoa, P. Liatsis, and A. J. Hussain, "A deep gated recurrent neural network for petroleum production forecasting," *Mach. Learn. with Appl.*, vol. 3, p. 100013, 2021.

- [70] N. M. Rezk, M. Purnaprajna, T. Nordström, and Z. Ul-Abdin, “Recurrent neural networks: An embedded computing perspective,” *IEEE Access*, vol. 8, pp. 57967–57996, 2020.
- [71] <http://www.sefidian.com/2018/01/30/gated-recurrent-unit-gru-with-pytorch/>
- [72] D. Zhang, J. Wang, and X. Zhao, “Estimating the uncertainty of average F1 scores,” in *Proceedings of the 2015 International conference on the theory of information retrieval*, pp. 317–320, 2015.
- [73] R. Yacouby and D. Axman, “Probabilistic extension of precision, recall, and f1 score for more thorough evaluation of classification models,” in *Proceedings of the first workshop on evaluation and comparison of NLP systems*, pp. 79–91, 2020.
- [74] G. Brassington, “Mean absolute error and root mean square error: which is the better metric for assessing model performance?,” in *EGU General Assembly Conference Abstracts*, p. 3574, 2017.
- [75] K. Hou, L. Peng, and W. Xiong, “Is R-squared a measure of market inefficiency?,” 2013.