TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Paula Pakina 213076IAIB

BEHAVIOR ANALYSIS AND PREDICTION OF VULNERABILITIES IN IOT DEVICES USING DATA MINING AND MACHINE LEARNING TECHNIQUES

Bachelor's Thesis

Supervisor: Mohammad Reza Heidari Iman
MSc

Co-supervisor: Tara Ghasempouri
PhD

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Paula Pakina 213076IAIB

IoT-seadmete haavatavuste analüüs ja ennustamine andmekaeve ja masinõppe meetodite abil

Bakalaureusetöö

Juhendaja: Mohammad Reza Heidari Iman
MSc

Kaasjuhendaja: Tara Ghasempouri
PhD

Tallinn 2024

Author’s Declaration of Originality

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

Author: Paula Pakina

27.05.2024

Abstract

The widespread adoption of the Internet of Things (IoT) introduces significant security challenges, particularly in securing devices from cyber threats. This thesis explores the application of data mining and machine learning (ML) techniques to improve the security of IoT devices by analyzing and predicting vulnerabilities within their networks, aimed to identify patterns and behaviors indicative of potential security threats. The IoT-23 dataset is used, comprising network traffic from various IoT devices, to implement association rule mining (ARM) and ML methodologies.

Two methods for analyzing IoT network traffic are presented. The first approach employs three ARM algorithms: Apriori, FP-Growth, and ECLAT. The goal is to analyze network traffic and discover association rules that show regular patterns indicative of specific traffic activity and potential malicious behavior. The experimental results suggest that FP-Growth offers the best performance in terms of consistent time and memory efficiency, particularly in the larger datasets.

The second method employs three ML techniques: Decision Tree, Random Forest, and K-Nearest Neighbors. The goal is to accurately predict and classify benign and different types of malicious activities within IoT traffic data. The models demonstrated high precision in identifying traffic activities, with Decision Tree and Random Forest achieving an accuracy of 86%, while K-Nearest Neighbors showed a slightly lower accuracy of 85%.

The comparison of ARM and ML results showed that ARM provides clear, interpretable patterns useful to make the decisions of models more clear and comprehensible to people. On the other hand, ML offers stronger predictive capabilities, handling complex data patterns and relationships.

The thesis is written in English and is 47 pages long, including 6 chapters, 3 figures and 12 tables.

Annotatsioon

IoT-seadmete haavatavuste analüüs ja ennustamine andmekaeve ja masinõppe meetodite abil

Asjade Interneti (IoT) laialdane kasutuselevõtt toob kaasa olulisi turvaprobleeme, eriti seadmete kaitsmisel küberohtude eest. See töö uurib andmekaeve ja masinõppe tehnikate rakendamist IoT-seadmete turvalisuse parandamiseks, analüüsides ja ennustades nende võrkude haavatavusi, tuvastades turvaohtudele viitavaid mustreid ja seaduspärasusi andmetes. Metoodikate rakendamiseks kasutatakse IoT-23 andmestikku, mis sisaldab erinevaid võrguliikluse tüüpe mitmesugustest IoT-seadmetest.

IoT-võrguliikluse analüüsimiseks esitatakse kaks meetodit. Esimene kasutab kolme assotsiatsioonireeglite kaevandamise algoritmi: Apriori, FP-Growth ja ECLAT. Eesmärk on analüüsida võrguliiklust ja leida seoseid ja mustreid, mis viitavad või iseloomustavad kindlat võrguliikluse tüüpi või võimalikku küberohtu. Eksperimentide tulemused viitavad, et FP-Growth pakub aja ja mälu kasutuse osas parimat tulemust, eriti suuremate andmemahtude korral.

Teine meetod kasutab kolme masinõppe meetodit: Decision Tree, Random Forest ja K-Nearest Neighbors. Eesmärk on IoT võrguliikluse andmetes täpselt ennustada ja klassifitseerida tavalist käitumist ja erinevat tüüpi küberohte. Treenitud mudelid näitasid võrguliiklustegevuste tuvastamisel head täpsust, Decision Tree ja Random Forest saavutasid 86% täpsuse, samas kui K-Nearest Neighbors näitas veidi madalamat täpsust 85%.

Assotsiatsioonireeglite kaevandamise ja masinõppe tulemuste võrdlus näitas, et assotsiatsioonireeglite kaevandamine pakub selgeid ja tõlgendatavaid mustreid, mis teevad mudelite otsuseid inimestele arusaadavamaks. Masinõpe pakub aga paremaid ennustamisvõimalusi, käsitledes keerulisi andmemustreid ja seoseid.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 47 leheküljel, 6 peatükki, 3 joonist, 12 tabelit.

List of Abbreviations and Terms

ANN	Artificial Neural Network
ARM	Association Rule Mining
CART	Classification and Regression Trees
CHAID	Chi-square Automated Interaction Detection
CNN	Convolutional Neural Network
CPU	Central Processing Unit
C&C	Command and Control
DDoS	Distributed Denial of Service
DM	Data Mining
ECLAT	Equivalence Class Clustering and bottom-up Lattice Traversal
FN	False Negative
FP	False Positive
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet Of Things
KNN	K-Nearest Neighbors
ML	Machine Learning
MLP	Multi-Layer Perceptron
NB	Naive Bayes
PCAP	Packet Capture
SSH	Secure Shell Protocol
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TCP-SYN	Transmission Control Protocol Synchronize
TN	True Negative
TP	True Positive

Table of Contents

List of Figures

List of Tables

1. Introduction

The Internet of Things (IoT) is a broad network where devices, sensors, and everyday items are connected through the internet, allowing fast and efficient communication and data exchange. IoT devices have electronics, sensors, and software that enables them to collect and exchange data automatically, without human intervention [1].

IoT has been actively integrated into different processes of everyday routines, improving efficiency and accuracy in many fields, from healthcare to transportation and automotive industries [2]. The International Data Corporation (IDC) estimates that investments in the IoT ecosystem would exceed $1 trillion by 2026, with a compound annual growth rate (CAGR) of 10.4% between 2023 and 2027 [3].

IoT creates a dynamic infrastructure, where devices automatically communicate and exchange data from sensors with each other. This connectivity results in an effective technology, optimizing processes, offering convenience and efficiency in various aspects of daily living and industrial operations [1].

Unfortunately, the broad adoption of IoT has also led to serious security challenges. Due to interconnected nature, the devices are especially vulnerable to cybersecurity attacks and risks. The devices may leak sensitive data, violating user privacy and potentially resulting in serious consequences like data breaches and unauthorized access [4], [5]. For example, unauthorized access to IoT devices can enable attackers to manipulate critical healthcare devices or disrupt transportation systems, leading to major consequences.

Moreover, traditional security measures are often insufficient for IoT devices, due to their unique characteristics, such as limited processing power, high volume of data, and energy constraints [6], [7], [8]. As a result, there is a need to develop adaptive security solutions capable of protecting IoT ecosystems without compromising their functionality. This work aims to address these gaps by utilizing data mining and machine learning techniques.

The main goal of this work is to improve the security of IoT devices by accurately identifying, analyzing, and predicting the behavior and potential vulnerabilities in these devices, using association rule mining and machine learning techniques. The goal is to uncover patterns and traffic characteristics indicative of various IoT traffic activity types and predict these potential security threats, enabling proactive measures.

The main goal is divided into three objectives:

1. To use association rule mining techniques to analyze and discover interesting patterns and rules from IoT device traffic data.
2. Apply machine learning algorithms to predict and classify different types of traffic activity or potential attacks.
3. Evaluate the effectiveness of different association rule mining and machine learning methods in identifying and predicting various IoT traffic activity types, providing insights into the suitability of each method for addressing the specific challenges of IoT device security.

This work aims to provide insights into mitigating vulnerabilities in IoT devices by analyzing, comparing, and evaluating data mining and machine learning methodologies, thereby contributing to a safer and more secure interconnected society.

2. Background

In the following sections, the related background and concepts of association rule mining and machine learning are elaborated.

2.1 Association Rule Mining

Organizations can now gather large amounts of data as a result of the rapid progress in the collection of user data and storage systems. However, because of the enormous amount of data, the process of selecting and extracting valuable data patterns happens to be quite difficult, standard data analysis tools are frequently useless or inefficient [7].

Data mining (DM) is a process of extracting and analyzing useful information or data patterns in large amounts of data [9]. While data mining includes a range of techniques for data analysis, its main goal is the exploration and description of data patterns. These processes involve identifying significant data patterns or relationships, that represent the underlying connections within the data [10].

Association rule mining (ARM) is a major DM technique used to discover interesting patterns or relationships, known as association rules, within large datasets. Association rules describe the different correlations, frequent patterns, relationships, or dependencies among items in transactions or records [11].

An association rule is an expression of the type $X \Rightarrow Y$, where X and Y are sets of items or variables and $X \cap Y = \varnothing$. X is referred to as the antecedent, it represents a set of items that are found together in transactions or records. Y is known as the consequent, it represents another set of items that are associated with or likely to occur together with the items in set X. The rule $X \Rightarrow Y$ implies that if items in set X are present, then items in set Y are also likely to be present in the same transaction or record [12].

Before association rules can be mined, frequent itemsets need to be identified, which refer to groups of variables in data that appear together with a significant frequency [13]. This frequent occurrence together indicates potential strong relationships or associations between the variables.

Support of the rule evaluates how frequently the combination of items X and Y appears together in the dataset. It is the percentage of transactions that contain both X and Y [11]. For the rule $X \Rightarrow Y$, the support is calculated by the following formula [13]:

$$\mathit{Support}(X \Rightarrow Y) = P(X \cup Y) \tag{1}$$

The probability $P(X \cup Y)$ indicates that a transaction contains both X and Y, which is the union of itemsets X and Y. For example, if we have a dataset of 100 transactions and 40 of them contain both items X and Y, then the support for the association rule $X \Rightarrow Y$ would be 40%.

The **minimum support** threshold is used to determine the frequency of an itemset. It is the minimum value that an itemset must meet to be considered frequent. If the frequency of the itemset meets or exceeds the specified threshold, it is considered frequent [9].

Confidence of the rule is a measure of the strength of the association between X and Y. For the rule $X \Rightarrow Y$, the confidence is calculated by the following formula [13]:

$$\mathit{Confidence}(X \Rightarrow Y) = P(Y|X) \tag{2}$$

It measures the probability of finding item Y in a transaction given that item X is also present there, which is the conditional probability $P(Y|X)$. For example, if out of the 40 transactions containing both X and Y, 30 transactions also contain item X, then the confidence for the association rule $X \Rightarrow Y$ would be 75%.

The **minimum confidence** threshold is the minimum value that the confidence of a rule must meet in order to be considered useful. If the confidence of the rule meets or exceeds the specified threshold, the rule is considered strong [9].

ARM focuses on finding association rules that meet the set minimum support and confidence thresholds [12]. Given the large amount of data processed, users usually set values of support and confidence to exclude rules that may not be interesting or relevant. This allows to focus on more important, useful, and reliable rules.

While ARM techniques have been effectively used in a number of fields, including market basket analysis, recommendation systems, healthcare, and manufacturing sectors, they are not without their limitations [14]. The execution time grows exponentially as the number of items increases. Standard ARM techniques need a substantial amount of computational time when processing immense volumes of data [8]. Along with that, ARM needs data preparation prior to application. The algorithms expect binary data, which requires the converting or discretization of continuous numerical values [15].

In the following sections, the most commonly used ARM algorithms are presented and explained.

2.1.1 Apriori

Apriori is the first ARM technique that introduced the use of support-based pruning to effectively manage the exponential growth of candidate itemsets. The algorithm employs a systematic approach to identify frequent itemsets within a dataset, which are then used to derive meaningful association rules [12].

Apriori is comprised of four primary steps [12]:

1. **Initialization:** Scanning the dataset to calculate the support for each item and collecting those that meet a predefined minimum support threshold.
2. **Candidate Generation:** Using a breadth-first search to iteratively generate larger candidate itemsets based on the frequent itemsets discovered in the previous iteration.
3. **Pruning:** Applying the Apriori principle, where all supersets of an infrequent itemset are also infrequent, thus significantly reducing the number of candidate itemsets.
4. **Termination:** Ending the process when no new frequent itemsets are generated, the algorithm outputs the discovered frequent itemsets with their corresponding support.

Pseudocode for the Apriori Algorithm [16]

Algorithm 1 Apriori

```
 1: Inputs:
 2: $D$: Dataset of transactions;
 3: min_support: Minimum support threshold;
 4: Outputs:
 5: $L$: List of all frequent itemsets;
 6: Procedure:
 7: Initialize $L_1$ with frequent 1-itemsets.                    ▷ Initialization
 8: $k = 2$
 9: while $L_{k-1} \neq \emptyset$ do
10:     Generate $C_k$ by joining $L_{k-1}$ with itself;           ▷ Candidate Generation
11:     Count support for each candidate in $C_k$;
12:     Prune candidates in $C_k$ not meeting min_support;         ▷ Pruning
13:     $L_k = \{c \in C_k \mid \text{Supp}(c) \geq \text{min\_support}\}$;
14:     $k = k + 1$
15: end while                                                      ▷ Termination
16: Concatenate $L_1, L_2, \ldots, L_{k-1}$ into $L$, containing all sizes' frequent itemsets;
17: Return $L$.
```

Algorithm 1 presents the details of the Apriori algorithm. In this algorithm, D denotes the dataset of transactions, where each transaction is a set of items, and *min_support* is the minimum support threshold that determines which itemsets are considered frequent enough for further analysis. In line 7, the initialization process involves identifying individual items in the dataset that meet the minimum support threshold, forming the first layer of potential itemsets L_1. In lines 8 to 15, the algorithm iterates to find all frequent itemsets. In line 10, candidate itemsets C_k are generated by joining itemsets from L_{k-1} with itself. In lines 11 to 14, the support for each candidate in C_k is calculated. Transactions in D are scanned and each candidate's support count is updated. Itemsets not meeting the *min_support* are pruned, and those that do are added to L_k. The algorithm increments k and continues this process until L_{k-1} is empty. Finally, in line 16, all frequent itemsets of various sizes from $L_1, L_2, \ldots, L_{k-1}$ are concatenated into L, which contains all frequent itemsets identified by the algorithm.

Apriori also has its limitations. It can still be computationally expensive, especially for large datasets, due to multiple scans of the data for support counting and the expensive generation of candidate itemsets [17].

2.1.2 FP-Growth

The FP-Growth (Frequent Pattern Growth) algorithm is a notable improvement in ARM. It addresses several limitations of the Apriori algorithm by using a more efficient approach for detecting frequent itemsets without the need for candidate generation and multiple database scans. FP-Growth uses a tree-like structure, referred to as FP-tree, to store compressed information about frequent patterns within the dataset. This method improves the performance, especially in large and sparse datasets, by eliminating the repeated dataset scans required by Apriori [17].

FP-Growth is comprised of two primary steps [17]:

1. Building the FP-tree:
   (a) The dataset is scanned to count the frequency of each item.
   (b) Items are sorted in decreasing order of frequency to form a list.
   (c) Transactions are sorted based on this list and the FP-tree is constructed by inserting sorted transactions into the tree.
2. Mining the FP-tree:
   (a) Frequent itemsets are extracted through a depth-first traversal of the FP-tree.
   (b) For each item in the tree, a conditional FP-tree is constructed, representing a smaller dataset that includes only transactions containing that item.

(c) The conditional FP-tree is then mined recursively to extract frequent itemsets.

Pseudocode for the FP-Growth Algorithm [18]

Algorithm 2 FP-Growth

```
1: Inputs:
2: $D$: Dataset of transactions;
3: min_support: Minimum support threshold;
4: Outputs:
5: $F$: List of all frequent itemsets;
6: Procedure:
7: Scan $D$ to find the support count of 1-itemsets;
8: Sort $D$ and initialize the FP-tree with a root labeled "null";
9: for each transaction $t$ in $D$ do                                    ▷ Building the FP-tree
10:     Order items in $t$ by descending support count;
11:     Insert ordered $t$ into the FP-tree, incrementing counts for existing nodes;
12: end for
13: Initialize $F$;                                                        ▷ Mining the FP-tree
14: for each item $i$ do
15:     Extract paths leading to $i$ in FP-tree;
16:     Construct conditional FP-tree for $i$;
17:     Recursively mine the conditional FP-tree;
18:     Collect all frequent itemsets involving $i$ into $F$;
19: end for
20: Return $F$.
```

Algorithm 2 presents the details of the FP-Growth algorithm. In this algorithm, D denotes the dataset of transactions, each transaction being a set of items, and *min_support* is the minimum support threshold, that determines which itemsets are considered frequent enough for further analysis. In line 7, the dataset D is scanned to find the support count of 1-itemsets. In lines 8 to 12, the dataset D is sorted based on the frequency of items, for the efficient construction of the FP-tree. The root of the FP-tree is created and labeled as "null," serving as the starting point for all transactions. Each transaction t from the sorted dataset is inserted into the FP-tree. Transactions are broken down by items and each item is inserted into the tree according to the frequency order. If an item already exists in the current path, its count is incremented. Otherwise, a new node is created. This step iteratively builds the tree by adding all transactions. In lines 13 to 18, an empty list F is initialized to hold all frequent itemsets discovered during the mining process. For each item i in the FP-tree, paths leading to i are extracted to form a conditional pattern base. This pattern base represents a collection of paths in the FP-tree that end with the item i and include all the items that appear before i in those paths. A conditional FP-tree is then constructed from this pattern base by counting the itemsets in the paths that meet the *min_support*. The algorithm recursively mines these conditional FP-trees, continually breaking the problem into smaller sub-problems. Frequent itemsets involving i are collected into the list F. After all items have been processed and their sub-trees explored, the complete list of frequent itemsets F is returned.

FP-Growth outperforms Apriori in terms of performance and scalability by eliminating candidate generation and multiple database scans, which is particularly advantageous for large datasets. Although FP-Growth requires more memory to store the FP-tree data structure, the overall advantages greatly outweigh this limitation. The compact structure of the FP-tree improves scalability and reduces the computational load. Furthermore, FP-Growth divides the mining process into smaller tasks using conditional FP-trees, enabling parallel processing that further improves its performance [17].

2.1.3 ECLAT

ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal) offers a distinct approach to ARM by using a vertical data structure. This method differs from the horizontal approach of the Apriori and FP-Growth algorithms, focusing instead on transaction ID intersections to find frequent itemsets. ECLAT transforms the dataset into a vertical format where each item is associated with a list of transaction IDs in which it appears. This structure allows ECLAT to efficiently calculate itemset supports through set intersection operations, which can be significantly faster than the repeated data scans used in other methods [17].

ECLAT is comprised of four primary steps [17]:

1. **Initialization:** Converting the dataset into a vertical format, where each item is associated with a list of transaction IDs that contain it.
2. **Candidate Generation:** Generating candidate itemsets for each item by intersecting transaction ID lists and calculating their support.
3. **Recursive Mining:** Expanding itemsets recursively by intersecting transaction IDs of combined items and checking if the resulting set meets the minimum support threshold.
4. **Termination:** Ending the process when no further itemsets can be generated that meet the required support level.

Algorithm 3 presents the details of the ECLAT algorithm. In this algorithm, D denotes the dataset of transactions, each transaction being a set of items, and *min_support* is the minimum support threshold that determines which itemsets are considered frequent enough for further analysis. In line 7, the initial step involves transforming the horizontal transaction dataset into a vertical format. This means instead of listing transactions by items, each item is associated with a list of transaction IDs where it appears, referred to as the TID set. The vertical format is needed for quick set intersections in later steps. In line 8, items that do not meet the minimum support threshold are filtered out. An empty list E is initialized in line 9 to store all frequent itemsets discovered. In lines 10 to 16, for each item i, candidate itemsets are generated by considering pairs of items i and j. The support for each candidate pair $\{i, j\}$ is calculated by intersecting their TID sets. If the support meets the minimum support threshold, the itemset $\{i, j\}$ is added to the list E. This process is recursively applied to generate larger itemsets from $\{i, j\}$, continuing until no further itemsets meet the required support threshold. Finally, the algorithm returns the list E containing all the frequent itemsets discovered.

Pseudocode for the ECLAT Algorithm [19]

Algorithm 3 ECLAT

```
 1: Inputs:
 2: $D$: Dataset of transactions;
 3: min_support: Minimum support threshold;
 4: Outputs:
 5: $E$: List of frequent itemsets;
 6: Procedure:
 7: Convert $D$ into a vertical data format;                          ▷ Initialization
 8: Filter items with support $\geq$ min_support;
 9: Initialize $E$ as an empty list to store frequent itemsets;
10: for each item $i$ in $D$ do
11:     Generate candidate itemsets starting with $i$;           ▷ Candidate Generation
12:     for each item $j$ occurring with $i$ do
13:         Compute support for $\{i, j\}$ by intersecting their TID sets;
14:         if support of $\{i, j\}$ meets min_support then
15:             Add $\{i, j\}$ to $E$;
16:             Recursively generate larger itemsets from $\{i, j\}$;    ▷ Recursive mining
17:         end if
18:     end for
19: end for
20: Return $E$;
```

However, while ECLAT offers notable advantages in terms of efficiency and memory usage, it also has its limitations. One notable drawback is its performance degradation with increasing dataset sparsity. As ECLAT relies on vertical data representation, where transactions are stored as lists of items, sparse datasets may lead to longer transaction ID lists and subsequently slower intersection operations, reducing its efficiency compared to Apriori and FP-Growth in such scenarios [17].

2.2 Machine Learning

Machine Learning (ML) offers a range of methods and techniques with the potential to improve the security of IoT systems. At its core, ML involves the use of intelligent algorithms to optimize performance based on historical data or examples. Unlike traditional programming paradigms, where rules are explicitly programmed, ML systems learn from data and iteratively improve their performance over time. The ability to learn and adapt makes ML a powerful tool in the context of IoT security [20].

One of the primary applications of ML in IoT security is anomaly detection. By applying ML algorithms, IoT devices can learn to recognize normal behavior patterns and identify deviations in data that may indicate suspicious or malicious activity [21]. The integration of ML into IoT security offers a significant benefit by allowing the processing and analysis of large volumes of data produced by IoT devices. Traditional security mechanisms often struggle to cope with the large volume and variety of data [7]. Conversely, ML algorithms are capable of processing and analyzing this data to identify abnormal patterns indicative of cyber attacks or breaches. Additionally, ML models can continuously evolve and self-improve over time as they are given new data, making them highly suitable for dynamically changing IoT environments [20].

However, ML is not without its limitations. One of the primary shortcomings is the inherent complexity of ML algorithms, which can cause computational and memory challenges in IoT devices with limited resources. Additionally, ML algorithms often require large amounts of labeled data for training, which may not always be available. Finally, the complexity of data management and analytics in IoT network traffic poses challenges for some ML algorithms. The data generated within IoT networks has a wide variety of characteristics, with different data types, formats, and semantics. Having a lot of such variations may pose challenges in terms of efficient and unified generalization, specifically in case of big data and various datasets with many diverse attributes [20], [22].

Evaluation Metrics

The performance of ML models is usually evaluated using the evaluation metrics: accuracy, precision, recall, and F1-score. These metrics are based on the outcomes classified as True Positives, True Negatives, False Positives, and False Negatives [23]:

- True Positive (TP): Instances where the model correctly predicts the positive class. For example, IoT traffic labeled as DDoS and correctly identified by the model as DDoS.

- True Negative (TN): Instances where the model correctly predicts the negative class. For example, IoT traffic that is not DDoS and is accurately identified by the model as non-DDoS.
- False Positive (FP): Instances where the model incorrectly predicts the positive class. For example, benign traffic that is mistakenly identified as DDoS.
- False Negative (FN): Instances where the model incorrectly predicts the negative class. For example, DDoS traffic that is wrongly classified as benign.

Accuracy measures the overall correctness of the model. It is the ratio of correctly predicted observations to the total observations, calculated by the following formula [24]:

$$Accuracy = \frac{TP + TN}{Total\ Observations} \tag{3}$$

Precision indicates the reliability of the positive predictions. It is the ratio of correctly predicted positive observations to the total predicted positives, calculated by the following formula [24]:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

Recall measures the ability of the model to detect all relevant instances. It is the ratio of correctly predicted positive observations to all observations in the actual class, calculated by the following formula [24]:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

F1-score evaluates the accuracy of a model by considering both its precision and recall. The score helps in understanding how well the model performs in terms of not just identifying true positives, but also in minimizing false positives and negatives. The F1-score is calculated by the following formula [24]:

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

Machine Learning Types

There are four types of ML algorithms: supervised, unsupervised, semi-supervised, and reinforcement learning [20].

Supervised learning involves training models with labeled data to predict targets based on input features. The goal is to identify patterns within data to classify elements into predefined classes. Common examples include classification and regression tasks. Regression techniques such as Support Vector Regression (SVR), linear regression, and polynomial regression are used for continuous outputs, while classification algorithms like K-Nearest Neighbor, logistic regression, and Support Vector Machine (SVM) handle discrete outputs. Neural networks can handle both classification and regression tasks [20].

Unsupervised learning, on the other hand, does not require labeled data and focuses on finding similarities among unlabeled data to classify them into different groups. This approach is used when outputs are not well-defined and there is a need to discover the structure within the raw data. Clustering techniques, such as K-means clustering, are commonly used to group data based on specified similarity criteria [20].

Semi-supervised learning falls between supervised and unsupervised learning, as it requires the labeling of only a part of the data. This scenario occurs, when labeling data is costly or impractical for the entire data but rather for only a subset. Semi-supervised algorithms use both labeled and unlabeled data for building the models, making them suitable for situations with limited labeled data [20].

Reinforcement learning differs from the previous methods as it does not rely on predefined outcomes. Agents interact with the environment, receive feedback in the form of rewards or penalties, and learn optimal strategies to maximize long-term rewards. Reinforcement learning is usually used in dynamic environments, where systems adapt and learn to accomplish tasks without explicit programming [20].

In the following sections, some of the most commonly used ML algorithms are presented and explained.

2.2.1 Decision Tree

Decision Tree (DT) is a supervised learning algorithm, used for both classification and regression problems. It operates by partitioning the feature space into a tree-like structure consisting of decision nodes and leaf nodes [25].

The construction of a DT involves two main phases [25]:

1. Induction Phase:
   (a) Recursively selects the most informative features that split the dataset effectively.

   (b) Creates a tree where each decision node corresponds to a feature that best separates the data into subsets, optimizing the class separation or prediction.

2. Inference Phase:
   (a) Classifies new instances by traversing the tree based on their features.
   (b) The path followed depends on the feature values of the instance, leading to a prediction that matches the label of the leaf node.

Pseudocode for the Decision Tree Algorithm [26]

Algorithm 4 Decision Tree

```
1:  Inputs:
2:  $D$: Training dataset;
3:  $F$: Feature set;
4:  Outputs:
5:  Decision tree model
6:  Procedure:
7:  Initialize with dataset $D$ as the root node;
8:  while stopping criteria not met do
9:      Check stopping criteria (purity, max depth, min information gain);
10:     if stopping criterion is met then
11:         Return node as a leaf node;
12:     else
13:         Calculate information gain for all features in $F$;
14:         Select the feature with the highest information gain;
15:         Split $D$ into subsets based on the selected feature;
16:         for each subset do
17:             Create a new node;
18:             Recur with the subset and remaining features;
19:         end for
20:     end if
21: end while
22: Return the Decision Tree model
```

Algorithm 4 presents the details of the Decision Tree algorithm. In this algorithm, D denotes the training dataset, where each instance is composed of a set of features and class labels or target value. The F is a list of features used to split the data. The algorithm starts from line 7, where the initialization involves setting the entire dataset as the root node, from which all decisions will branch out. In line 8, the algorithm enters a loop that continues until a stopping criterion is met. The stopping criteria include node purity, when all instances at a node belong to the same class, reaching a predefined maximum depth of the tree, or achieving a minimum information gain, where information gain from splitting a node is below a threshold. In lines 10 to 11, if a stopping criterion is met, the node is marked as a leaf node and the algorithm stops further splitting at this node. If no stopping criterion is met, lines 12 to 14 involve calculating the information gain for all features at the current node. The feature with the highest information gain is selected for splitting the dataset. In line 15, the dataset is divided into subsets based on the values of the selected feature. Each subset corresponds to a new branch in the tree, leading to the creation of new nodes. From lines 16 to 18, the algorithm recursively applies the same process to each new node created after the split. This involves repeating the steps of checking stopping criteria, calculating information gain, and splitting the dataset until all nodes either become leaf nodes or meet a stopping criterion. The algorithm terminates when all nodes meet a stopping criterion, and returns the constructed model.

One of the key advantages of Decision Trees is that they can handle both categorical and numerical data, eliminating the need for extensive preprocessing steps such as normalization or encoding. Additionally, Decision Trees are highly interpretable, allowing users to easily understand and visualize the decision-making process [25].

Despite the advantages, Decision Trees are prone to overfitting, especially when the tree depth grows too large or when dealing with noisy data. Overfitting can lead to poor generalization performance, where the model performs well on the training data but fails to accurately classify unseen instances. Techniques such as pruning, which involves removing nodes or subtrees, can help simplify the tree and therefore prevent this issue [25].

2.2.2 Random Forest

Random Forest (RF) is a supervised learning algorithm used for both classification and regression tasks. It is an extension of the Decision Tree algorithm, belonging to the category of ensemble learning methods, which involve combining multiple individual models to improve predictive performance [27].

The RF uses three main techniques [27]:

- Ensemble of Decision Trees: Multiple decision trees are trained independently and then their results are combined to make a final prediction.
- Bootstrapping: Each decision tree is trained independently using a subset of the original dataset.
- Feature Randomness: At each node of the decision tree, only a random subset of features is considered for splitting. This random selection helps to make the trees less correlated and prevent overfitting on the training data.

Pseudocode for the Random Forest Algorithm [28]

Algorithm 5 Random Forest

```
 1: Inputs:
 2: $D$: Training dataset;
 3: $N$: Number of trees to grow;
 4: $k$: Percentage of features to consider for the best split;
 5: Outputs:
 6: Random Forest model
 7: Procedure:
 8: Initialize the forest as an empty list;
 9: for $i = 1$ to $N$ do
10:     Randomly sample $D$ with replacement to create $D_i$;
11:     Create a root node $N_i$ containing $D_i$;
12:     Call BUILDTREE($N_i$);
13:     Add the tree to the forest;
14: end for
15: Use majority voting or averaging to combine tree predictions;
16: Return the Random Forest model;
17: function BUILDTREE(N)
18:     if N contains instances of only one class then
19:         return
20:     else
21:         Randomly select $k\%$ of the features;
22:         Select the feature $F$ with the highest information gain;
23:         Create child nodes $N_1, \ldots, N_f$ based on $F$;
24:         for each child node $N_i$ do
25:             Set $N_i$ to instances in $N$ matching $F_i$;
26:             Call BUILDTREE($N_i$);
27:         end for
28:     end if
29: end function
```

Algorithm 5 presents the details of the Random Forest algorithm. In this algorithm, D denotes the training dataset, where each instance is composed of a set of features and class labels or target values. N is the number of trees to grow in the forest, and k is the percentage of features to consider for the best split at each node in a tree. The algorithm begins in line 8, by initializing the forest as an empty list. In lines 9 to 14, for each of the N trees, bootstrap sampling is performed on D to create a new dataset D_i. This is done by randomly selecting samples from the original dataset D with replacement, ensuring each tree gets a different sample of the data, making the overall model more robust and less prone to overfitting. Each tree is constructed independently using its respective bootstrapped dataset D_i. At every decision node, instead of considering all features, the algorithm randomly selects $k\%$ of them. From these selected features, the one with the highest information gain is chosen to make the split. This randomness in feature selection ensures that the trees are less correlated with each other, improving the generalization capability. The trees are grown to their maximum depth or until a minimum node size is reached, without any pruning, allowing them to capture more complex patterns in the data. The function BUILDTREE is used recursively to construct each tree. Finally, in line 15, the individual predictions of each tree are combined to form the final prediction for the model. For classification tasks, majority voting is used, where the most common prediction among all trees is chosen as the final result. For regression tasks, the model calculates the average of all tree predictions to get a continuous output.

However, it is important to note that the performance of RF may vary depending on the dataset characteristics, hyperparameter tuning, and the specific problem. While RF generally offers robust predictions, it can be computationally intensive and slower in comparison to some other algorithms, as the parallel construction of multiple decision trees can impact processing resources [27].

Although these factors should be considered, RF remains a versatile and powerful tool in both classification and regression tasks. Its robustness, scalability, and accurate predictions are useful across a wide range of applications, including the detection of malicious network traffic and anomalies in IoT networks [27].

2.2.3 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric supervised learning algorithm mostly used for classification, but also suitable for regression tasks [29].

The main KNN components [29]:

1. Number of Neighbors: Select k, the number of nearest neighbors to consult, when predicting the label of a new sample.
2. Distance Metrics: The distance metric measures the closeness or similarity between data points. Common metrics are [30]:
   (a) Euclidean Distance: Defined as the length of a segment between two points.
   (b) Manhattan Distance: Calculates the sum of the absolute differences between points across all dimensions.
   (c) Minkowski Distance: Generalized metric that includes both Euclidean and Manhattan distances as special cases. It introduces a parameter p, by changing the p, different distance measures can be calculated.
3. Majority Voting or Averaging:
   (a) For classification, majority voting among the k nearest neighbors is used to

determine the predicted class. The class that appears the most among the neighbors is assigned to the new instance.

(b) For regression, the value of the target variable is calculated based on an average, maximum or minimum aggregation of its nearest neighbor's values.

Pseudocode for the K-Nearest Neighbors Algorithm [31]

Algorithm 6 K-Nearest Neighbors

```
1: Inputs:
2: $D$: Training dataset;
3: $k$: Number of neighbors;
4: A distance metric;
5: Outputs:
6: A predicted class label for new samples;
7: Procedure:
8: Load the training dataset $D$;
9: Initialize $k$ to the chosen number of neighbors;
10: for each new sample $x$ do
11:     for each example $d_i$ in $D$ do
12:         Compute the distance between $x$ and $d_i$;
13:         Add distance and index to an ordered collection;
14:     end for
15:     Sort the collection by distance;
16:     Select the top $k$ nearest neighbors;
17:     Get the labels of the selected $k$ entries;
18:     Return the most common of the $k$ labels;
19: end for
```

Algorithm 7 presents the details of the K-Nearest Neighbors algorithm for our classification task. The dataset D contains features and corresponding class labels. The number k specifies how many neighbors to consider when making a prediction. The distance metric, often Euclidean, Manhattan, or Minkowski, determines how the distance between data points is calculated. The algorithm starts from line 8, where the entire dataset D is stored in memory. In lines 9 to 14, for each new sample x, the algorithm calculates the distance from x to every sample in D and stores these distances along with their indices. In lines 15 to 19, it then sorts these distances from nearest to farthest and selects the top k nearest neighbors. The class of x is the most frequent class label among the selected k neighbors. After processing all new samples, the algorithm returns their predicted class labels.

KNN is simple to understand and implement, making it useful for various applications, such as pattern recognition, anomaly, and intrusion detection [32]. However, its performance can be sensitive to the choice of distance metric and the number of neighbors. It may also suffer from computational inefficiency with large datasets [22], [29].

Nonetheless, with appropriate parameter tuning and preprocessing techniques, KNN can be a powerful tool for various classification and regression tasks.

3. Related Work

The following subsections provide an overview of the related work on association rule mining and machine learning applications in IoT security, summarizing key studies and their findings.

3.1 ARM Applications in IoT Security

Despite ARM's ability to extract valuable insights from large amounts of data, it has not been widely studied within the context of IoT security. This gap suggests an area for potential development and studies of ARM-based security improvements in IoT systems.

In the study presented in [33], ARM is applied to darknet sensor data to detect IoT malware, focusing on identifying attack patterns of Mirai malware. Through TCP-SYN packets analysis before and after the release of Mirai's source code, the research validated the effectiveness of ARM in early malware detection and provided insights into evolving attack methodologies.

Similarly, the study [34] revealed the effectiveness of ARM in identifying behavioral patterns associated with the Satori botnet, evolving from the Mirai botnet source code. By analyzing darknet traffic data, specifically TCP-SYN packets, the study found key patterns indicative of Satori's activity, such as destination ports, TCP window size, and Types of Service (ToS). While the study's primary focus was on Satori, it also made comparisons with Hajime, another IoT malware. This comparison showed the potential of ARM not just for detecting a single type of malware, but for distinguishing between different malware based on their unique behavioral patterns.

However, as both studies are limited to only darknet data, using data from other sources across different networks, such as real IoT network traffic, could improve these analyses and help validate the findings.

The study in [35], proposed a hybrid intrusion detection method that uses ARM, specifically the Apriori algorithm, to extract features from network data. These features then serve as input for a classification model that combines Artificial Neural Networks (ANN) and the AdaBoost algorithm. This approach aimed to enhance the accuracy and efficiency of anomaly detection in communication networks. When tested on the KDDCUP99 dataset, the model outperformed existing techniques like classification tree methods and regression models, showing improvements in the accuracy of intrusion detection. Given that the study was conducted on communication network data, additional research on the relevance of the findings to IoT traffic data is needed.

3.2 ML Applications in IoT Security

With the rapid expansion of IoT deployments, traditional security approaches are no longer sufficient to defend against growing cyber threats [36]. As a result, ML has become a key tool for improving the security of IoT systems. By continuously analyzing large volume of data from IoT devices, ML algorithms can detect and respond to anomalous behavior.

The study presented in [37], focused on detecting IoT network attacks by applying different ML techniques on the Bot-IoT dataset, used for training models to detect botnet attacks in IoT networks. The CICFlowMeter was employed to extract 84 network traffic features defining network flows from raw traffic logs. The Random Forest Regressor algorithm was used to determine feature importance weights. Two different approaches were employed: one calculated weights separately for each attack type and the other determined weights for a group of attacks to identify common key properties. Then, seven ML algorithms were applied to the dataset. The results showed varying performance among algorithms, with Random Forest, ID3, and AdaBoost showing the highest effectiveness. The study suggests further research into the performance of unsupervised algorithms and the integration of diverse ML models to enhance detection capabilities.

The [38] study highlighted the effectiveness of Random Forest in mitigating cyberattacks on IoT networks, particularly its precision in predicting attack types. The dataset used contains 13 features, with 347,935 samples classified as normal and 10,017 samples classified as anomalous. The dataset contains eight distinct classes. Hence, additional research is needed to determine the scalability of the algorithms to big data, resilience to unforeseen challenges, and effectiveness under diverse circumstances.

The study in [39], conducted a comparative analysis of various ML algorithms, evaluating their performance on both weighted and non-weighted Bot-IoT dataset. The study found Random Forest as being particularly effective in accuracy and precision on the non-weighted dataset, while Artificial Neural Networks (ANN) showed higher accuracy for binary classification on the weighted dataset. In multi-class scenarios, KNN and ANN demonstrated high accuracy on weighted and non-weighted datasets, respectively.

The paper [40] introduced three ML based Intrusion Detection Systems (IDS) for detecting wormhole attacks in IoT networks: K-means clustering based, a Decision Tree based, and a hybrid approach combining both methods. The K-means based IDS achieved a detection rate between 70%-93% across different IoT network sizes. The Decision Tree based IDS had a detection rate of 71%-80%, while the hybrid approach had a detection rate of 71%-75%. Although the hybrid IDS had a slightly lower detection rate, it reduced the number of false positives compared to the other two methods.

The paper [41] explored both ML and Deep Learning (DL) methods for anomaly detection on the IoT-23 dataset. They tested models like Support Vector Machine (SVM), Decision Tree, Naive Bayes, and Convolutional Neural Networks (CNN). The Decision Trees model provided the best performance in terms of accuracy and computational efficiency. The research noted the computational demand of DL models, making them less suitable for real-time applications without significant hardware resources. The study faced challenges with the imbalance in the dataset, leading to potential overfitting towards more frequent attack types, suggesting the need for improved sampling techniques or cost-sensitive learning approaches.

Similarly, the study presented in [42], explored the efficiency of various ML algorithms, including Decision Tree, SVM, Naive Bayes, and CNNs, for detecting IoT botnets using the IoT-23 dataset. The Decision Tree outperformed other models, achieving the highest accuracy of 73% among the tested algorithms. The study indicates the potential of using Decision Trees for effective anomaly detection in IoT networks. However, the need to reduce feature dimensions due to computational challenges, suggests a need for more efficient data handling strategies, improving model performance without compromising the detection capabilities.

The study in [43], conducted a comparative study using multiple ML and DL techniques to detect anomalies in the IoT-23 dataset. The study used algorithms such as Decision Tree, Random Forest, Naive Bayes, and Bagging classifiers, with Decision Tree achieving the best accuracy of 95.6% and F-Measure of 76%. However, the minority classes were removed from the data, which could have improved the accuracy by simplifying the classification problem and reducing noise from less representative classes, but might limit the generalizability of the models.

The study in [44], employed various ML algorithms to detect anomalies within IoT network traffic, using the IoT-23 dataset. The study also implemented algorithms like Random Forest, Naive Bayes, Multi-Layer Perceptron, SVM, and AdaBoost. Among these, Random Forest achieved the highest accuracy of 99.5% and precision of 88%. However, the dataset had to be split into smaller parts and encoded to have less categories, to manage computational issues. This processing potentially oversimplified the data, which might limit the generalizability of the model to more complex real-world data. Hence, further research is needed to address the scalability challenges.

These studies show a trend towards using tree-based models for their efficiency in handling IoT data. However, common limitations occur, such as the need for extensive data pre-processing, potential overfitting, and the lack of generalization to more diverse real-world scenarios. These gaps suggest the need for more research on data mining and machine learning, addressing these common challenges.

4. Methodology

Figure 1 illustrates the general flow of the proposed methodology for the analysis of traffic behavior and prediction of vulnerabilities in IoT devices, utilizing ARM and ML techniques. The approach is divided into two main phases. The first involves using ARM to analyze the IoT network traffic behavior, to find underlying patterns and associations indicative of benign or various malware activities.

The second phase uses ML for the prediction of IoT network traffic types, distinguishing between benign and potentially malware traffic of different types. The success of this phase is measured by the model's ability to accurately classify traffic types, thereby enhancing the security framework for IoT devices.

After these two phases have been completed, a comparison of the results is made. The comparison evaluates the effectiveness and performance of the proposed methods in identifying associations, patterns, and classifying the IoT network traffic data.

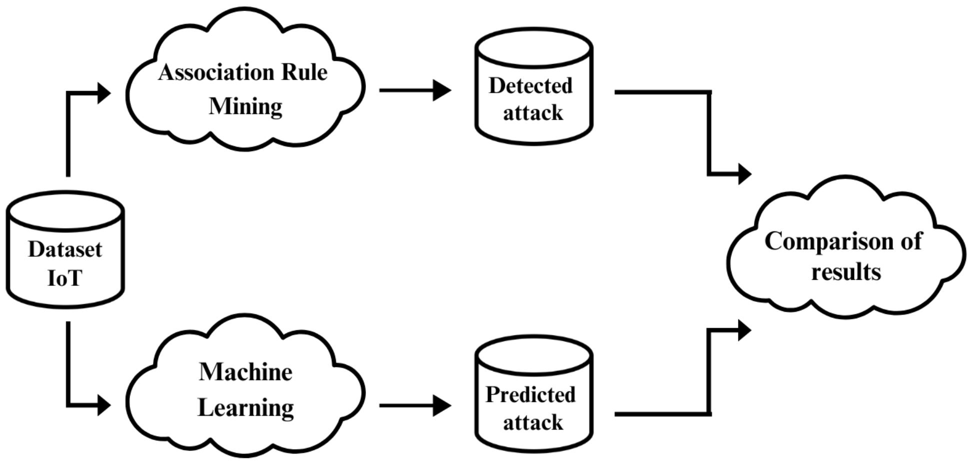

Figure 1. *Proposed Methodology*

4.1 Dataset Description

The dataset used for both ARM and ML is the IoT-23 dataset [45], developed by the Avast AIC laboratory with the Czech Technical University in Prague and released in January 2020. It consists of a collection of network traffic data that includes both malicious and benign activities within various IoT devices.

The dataset is expansive, containing over 325 million captures, with a majority, approximately 294 million, classified as malicious. The dataset splits into 23 scenarios, with 20 of them from IoT devices infected with malware. These scenarios show the behavior of the devices under attack. The other 3 scenarios are from devices, like a Philips HUE smart lamp, an Amazon Echo, and a Somfy smart door lock, providing a baseline of normal network behavior. These devices were not simulated but operated in a controlled network environment, representing realistic usage patterns and IoT network traffic.

The dataset is available in two formats: the original network captures .pcap files for direct analysis and conn.log.labeled files, derived by processing the .pcap files through the network analyzer Zeek. The conn.log.labeled files contain detailed network connection flows and labels which were assigned to them based on manual analysis and predefined rules. These files include 23 columns of metadata, from basic information like timestamps and IP addresses to advanced details such as connection states and specific malware activities. For practical reasons, this thesis focuses on the conn.log.labeled files, which are more accessible for analysis without the need for specialized software required to parse .pcap files.

Table 1 describes the main IoT traffic activities captured in the IoT-23 dataset. The dataset also contains combinations of these activity types, for example the label '*C&C-FileDownload*' indicates that C&C file was downloaded, or '*C&C-Mirai*' indicates that the attack was performed by the Mirai botnet.

Table 1. *IoT Traffic Activities Captured in The Dataset*

Activity Type	**Description**
Attack	Indicates an attempt to attack a vulnerable service, such as brute-forcing telnet logins or command injections in GET requests.
Benign	Marks connections with no detected suspicious or malicious activities, representing normal network traffic.
C&C	Indicates that an infected device is communicating with a control server, either regularly for downloading malware or for command exchange.
DDoS	Applied to network flows involved in a DDoS attack, overloading a target with excessive traffic.
FileDownload	Used for connections where the infected device downloads a file, often from a known malicious server.
HeartBeat	Indicates small, periodic packets sent to keep track of the infected host by the C&C server.

Continues...

Table 1 – *Continues...*

Activity Type	**Description**
Mirai, Okiru, Torii	Indicate connections with characteristics of respective botnet families, determined by similarities to known attack patterns.
PartOfAHorizontalPortScan	Indicates connections involved in scanning multiple IP addresses across the same port to gather information for future attacks.

Table 2 lists the traffic features included in the dataset. Each feature provides a specific type of data about the network flows, helping to distinguish between normal behaviors and potential security risks or attacks.

Table 2. *Description of Dataset Traffic Features*

Feature	**Description**
ts	Flow start time
uid	Unique ID
id.orig_h	Source IP address
id.orig_p	Source port
id.resp_h	Destination IP address
id.resp_p	Destination port
proto	Transaction protocol
service	Network service type
duration	Record duration
orig_bytes	Bytes sent from source to destination
resp_bytes	Bytes sent from destination to source
conn_state	Connection state
local_orig	Indicates if source is a local address
local_resp	Indicates if destination is a local address
missed_bytes	Missing bytes during transaction
history	History of source packets
orig_pkts	Total original packets sent from source
orig_ip_bytes	Total IP bytes sent from source
resp_pkts	Total packets received from destination
resp_ip_bytes	Total IP bytes received from destination
label	Type of traffic or attack

4.2 Association Rule Mining Methodology

Figure 2 presents the ARM methodology employed to analyze IoT network traffic data and detect different traffic activities or potential attacks on IoT devices. This process begins with the IoT-23 dataset, which undergoes preprocessing steps. These steps include data cleaning, feature selection, reduction, binning, and encoding, ensuring that the data is suitable for ARM algorithms.

The preprocessed data is analyzed using three different ARM algorithms: Apriori, FP-Growth, and ECLAT. The selection of these algorithms is based on their unique strengths. The Apriori algorithm is a fundamental and widely used method in data mining, making it a reliable baseline. The FP-Growth was chosen for its scalability and efficiency in processing large datasets. ECLAT's vertical data format provides a different perspective compared to horizontal approaches of Apriori and FP-Growth. The algorithms are employed to identify underlying patterns in the data that indicate various traffic activities or attacks, capturing these detections in form of association rules for further analysis.

The final step involves performance comparison among the algorithms to evaluate their effectiveness in identifying different IoT traffic activities and attacks. This comparative analysis can help to determine which algorithm performs best with IoT traffic data, highlighting their strengths and limitations.

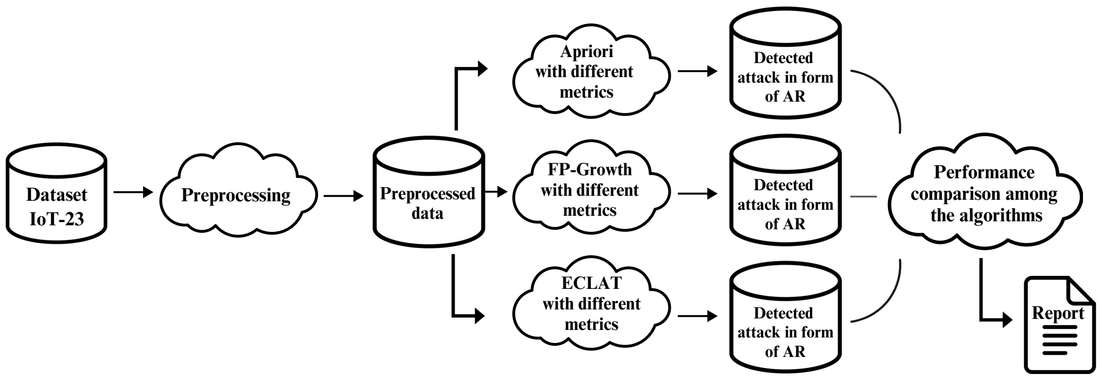
Dataset IoT-23
Preprocessing
Preprocessed data
Apriori with different metrics
FP-Growth with different metrics
ECLAT with different metrics
Detected attack in form of AR
Detected attack in form of AR
Detected attack in form of AR
Performance comparison among the algorithms
Report

Figure 2. *Association Rule Mining Methodology*

In the following sections, each phase of the ARM methodology is elaborated.

4.2.1 Preprocessing

Data Collection and Cleaning

The initial step of preprocessing for ARM involved loading the network connection logs, the aggregation of logs from specified folder paths, ensuring the collection of data across multiple files.

The next step was the standardization of traffic labels. The dataset initially contained synonymous labels for traffic types, that essentially described the same network behavior. A mapping strategy was employed, that involved creating a standardized dictionary where each variant of a label was mapped to a single, unified representative label. This ensured that all synonymous labels were treated as one, thereby maintaining consistency across the dataset. For example, labels like '*- Benign -*', '*- benign -*', and '*(empty) Benign -*' were all standardized to label '*Benign*'.

Feature Selection and Reduction

In the feature selection and reduction, specific columns were removed from the dataset to improve the data quality for more effective ARM analysis. The timestamp (*ts*) and unique identifier (*uid*) columns were removed due to their unique values for each record, which do not contribute to pattern recognition. Similarly, the source and destination IP addresses (*id.orig_h*, *id.resp_h*) were removed as ARM searches for generalizable patterns rather than instance specific details. The port numbers (*id.orig_p*, *id.resp_p*), were removed due to their high variance, which could result in a high number of rules with little significance. Lastly, local origin and response flags (*local_orig*, *local_resp*) and the connection history (*history*) were removed, as they are not very useful in detecting the kind of recurring associations that ARM searches for.

Binning and Feature Encoding

Feature engineering was crucial to make data suitable for ARM, particularly through the binning of continuous variables into categorical ones, which are more manageable for ARM analysis. Bins were chosen based on the distribution and characteristics of the dataset. For instance, the '*duration*' variable was binned into '*very_short*', '*short*', '*medium*', and '*long*' categories to capture both typical short-term connections and longer ones, indicative of sustained data transfers or potential attacks. Similarly, '*orig_bytes*', '*resp_bytes*', '*orig_-ip_bytes*', and '*resp_ip_bytes*' were binned into '*very_small*', '*small*', '*medium*', '*large*', and '*very_large*' to distinguish between routine small packets and larger data transfers, which could involve file downloads or unauthorized data transfer. Converting continuous variables into these discrete categories helps to identify broader patterns in the data, thereby improving the interpretability and performance of the further analysis. The bin thresholds were set to ensure distinctions and balanced distributions within each category.

Following binning, one-hot encoding was applied to the categorical variables, including those created during the binning process. One-hot encoding converts categorical data into a binary format. This step is crucial for ARM, as it relies on binary or discrete data to identify patterns and associations. This encoding creates a binary column for each category, marking the presence or absence of the category with a 1 or 0, respectively. The resulting dataset significantly expands the set of attributes, as each attribute represents a unique category or bin.

4.2.2 Apriori Implementation

The Apriori algorithm was applied to the preprocessed dataset using Python's *mlxtend* library. The algorithm was executed at support thresholds of 0.7, 0.8, and 0.9. This was done to discover patterns and compare performance at varying levels of item frequency. The experiments were not limited to the full dataset, subsets of half and a quarter of the data were also analyzed, to evaluate the impact of data volume on rule discovery and algorithm's performance. For each label in the dataset, association rules were generated based on the frequent itemsets extracted by Apriori. The rules were filtered to include only those with the traffic activities or attacks as their consequent.

4.2.3 FP-Growth Implementation

Similar to Apriori, FP-Growth was applied to the preprocessed dataset with support thresholds of 0.7, 0.8, and 0.9. The algorithm was evaluated using the entire dataset and its half and quarter subsets, and implementation utilized the *mlxtend* library. Association rules were extracted using the generated frequent itemsets and again filtered to include only those with the traffic activity of interest in their consequents. The tree-based approach of FP-Growth was expected to offer computational advantages, particularly when processing the full and half datasets, which was confirmed during the execution. The FP-Growth application confirmed the results obtained from the Apriori application, while also assessing the scalability of ARM techniques when applied to large-scale data.

4.2.4 ECLAT Implementation

The ECLAT algorithm, although known for its efficient depth-first search mechanism, encountered performance issues when applied to the whole and half of the dataset. As the *mlxtend* library does not support ECLAT, alternative methods, including a self-implementation and older libraries, were assessed. However, these approaches were too computationally intensive and impractical for the full and half dataset sizes. As the goal of this work was not to implement the algorithms, but to compare and evaluate their performance and ability to find patterns in IoT traffic data, the ECLAT analysis was limited to a quarter of the dataset and implemented with a *pyECLAT* package. This solution enabled the execution of the algorithm within a reasonable time frame.

The performance issues with ECLAT were mainly due to its complexity and the intensive computational resources required, which were not feasible for the full extent of the dataset in our environment. Despite these challenges, the quarter-dataset analysis was completed and provided insights into the potential and constraints of using ECLAT in large-scale ARM applications.

4.3 Machine Learning Methodology

Figure 3 illustrates the ML methodology applied to predict and classify network traffic types using the IoT-23 dataset. The initial phase is data preprocessing, which involves data cleaning and encoding to prepare the dataset for ML applications.

Following the preprocessing, three different ML models are employed: Decision Tree, Random Forest, and K-Nearest Neighbors. The algorithms were selected as they offer versatile approaches. Decision Trees can handle both numerical and categorical data, making them suitable for varied data generated by IoT devices. Random Forests, with their ensemble approach, improve prediction reliability and handle large datasets with high dimensionality effectively. KNN was chosen for its distinct instance-based approach, it can adapt to new data points without needing to retrain the entire model, useful in IoT environments where data is continuously generated.

Each model is further optimized to address the class imbalances and potential overfitting risks. Then, each optimized model processes the preprocessed data to classify and predict traffic types. Following prediction, the models are evaluated using various metrics, including accuracy, precision, recall, and F1-score.

The final phase involves a performance comparison among the models based on the evaluation metrics to determine which model performs best in predicting and classifying network traffic. The models are also compared based on training and prediction times. The results of these comparisons are detailed in a report, highlighting the strengths and potential improvements for each model in the context of IoT network security.

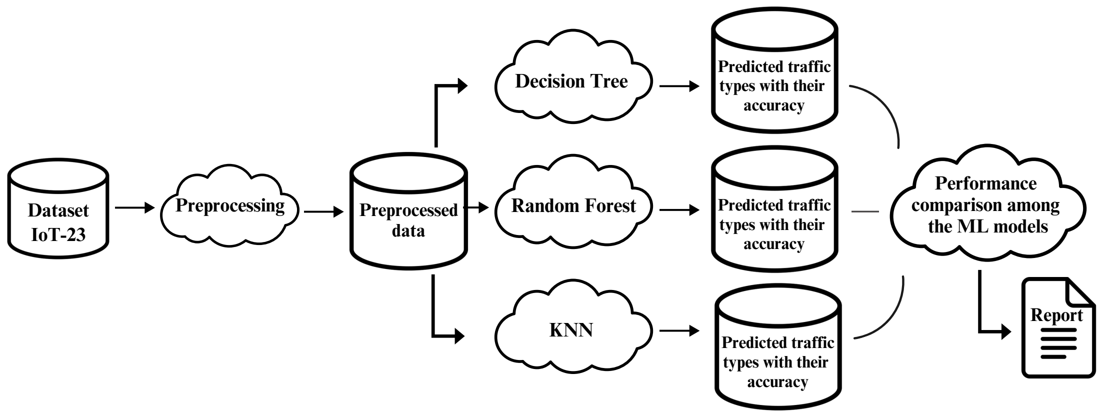
Dataset IoT-23
Preprocessing
Preprocessed data
Decision Tree
Random Forest
KNN
Predicted traffic types with their accuracy
Predicted traffic types with their accuracy
Predicted traffic types with their accuracy
Performance comparison among the ML models
Report

Figure 3. *Machine Learning Methodology*

In the following sections, each phase of the ML methodology is elaborated.

4.3.1 Preprocessing

Data Collection and Cleaning

The initial step of preprocessing was the same as in the ARM preprocessing, loading the network connection logs and ensuring the collection of data across multiple files. The process iterates through the dataset's directory, reads the conn.log.labeled files, containing labeled network traffic data. These files are then parsed into a pandas DataFrame, for easier data manipulation and further analysis.

Then label normalization process was also applied, where various labels that essentially described the same network behavior were unified under a single representative label, to ensure consistency across the dataset.

Additionally, the challenge of missing or undefined values represented by dashes was addressed, as these placeholders are unsuitable for numerical analysis and ML model training. Therefore, these dashes were replaced with zeroes in relevant columns.

Feature Encoding

The next step of preprocessing the dataset for ML involved converting categorical variables into a format that ML algorithms can efficiently process. The one-hot encoding was applied to categorical columns *proto*, *service*, and *conn_state*, transforming each category within these columns into separate binary column.

4.3.2 Decision Tree Implementation

The Decision Tree model was applied to the preprocessed dataset. The Python library *pandas* was used to load and process the dataset, while the *imblearn* package's *RandomOverSampler* was employed to optimize the model and address class imbalances by oversampling the minority classes. The Decision Tree classifier from *sklearn* library was chosen for the model and its hyperparameters were optimized using *GridSearchCV* from *sklearn* library. This method ensured the selection of parameters combination (*max_-depth*, *min_samples_split*, *min_samples_leaf*), that optimized the model's accuracy. Here *max_depth* determines the maximum depth of the tree, *min_samples_split* parameter defines the minimum number of samples a node must have before it can be split, and the *min_samples_leaf* parameter sets the minimum number of samples that a leaf node must have.

The validation process included 5-fold cross-validation, which involves partitioning the original dataset into five equal segments, referred to as folds. Then, the model is trained on four of these folds, while the fifth is used as the test set to evaluate performance. This process is repeated five times, with each of the five folds used exactly once as the test set. The results from the five iterations are then averaged to produce a single estimation. The accuracy and classification report were derived from testing the optimized model.

4.3.3 Random Forest Implementation

The implementation of the Random Forest model followed a similar preprocessing approach, using the *pandas* library for data loading and addressing class imbalances by randomly oversampling the minority classes. The Random Forest classifier from *sklearn* library was selected to construct the ensemble model. Hyperparameter tuning to improve the model's accuracy was also done with *GridSearchCV* from *sklearn*, focusing on parameter *n_estimators*, which specifies the number of trees in the forest, and also on parameters *max_depth*, *min_samples_split* and *min_samples_leaf*. The method also included 5-fold cross-validation, evaluating the model's performance across various data subsets, thus preventing overfitting and ensuring the model's generalizability. After the testing of the optimized model, accuracy and classification report were presented.

4.3.4 K-Nearest Neighbors Implementation

The KNN algorithm's implementation starts with standard preprocessing steps, including data loading and addressing class imbalances by randomly oversampling the minority classes. The KNN classifier from *sklearn* library was applied and optimized through *GridSearchCV*, focusing on parameters *n_neighbors*, *weights*, and *metric*. Where *n_-neighbors* is the number of neighbors to consider when making a prediction, the *weights* parameter determines the weight that is given to each point in computing the prediction, and the *metric* defines the distance metric used to measure the closeness between points. Validation incorporated 5-fold cross-validation, evaluating the model's performance. The performance of the KNN model was also presented by the accuracy score and classification report.

5. Experimental Results

This section provides an overview of the experiments conducted for association rule mining and machine learning. The experiments were repeated multiple times to ensure their results were consistent and reliable. In the following subsections, the results for each method are presented.

5.1 Association Rule Mining Results

5.1.1 Association Rules

The application of the Apriori, FP-Growth, and ECLAT algorithms to the preprocessed dataset generated a series of association rules that provide insights into various types of network traffic activities. For each traffic activity, the most common and generalizable rule was selected based on the following criteria:

1. **Coverage:** The rule that captures a wide range of traffic characteristics specific to each traffic activity was chosen, ensuring that the traffic patterns are comprehensive.
2. **Minimal Redundancy:** To avoid redundancy and overlap, unique rules or those that are not closely correlated with other rules were preferred, focusing on more distinct patterns for each traffic activity.
3. **High Support and Confidence:** High values of confidence and support for these rules ensured that they are common and reliable patterns associated with each activity type.

Below, the key association rules organized by traffic activity type are presented:

Table 3. *Association Rules for Vulnerability Analysis*

Traffic Activity	**Rule**	**Support**
C&C-FileDownload	service_http & resp_bytes_very_large & resp_ip_bytes_very_large & proto_tcp & conn_state_SF ⇒ label_C&C-FileDownload	1.000

Continues...

Table 3 – *Continues...*

Traffic Activity	**Rule**	**Support**
Attack	resp_ip_bytes_medium & proto_tcp & conn_state_SF & orig_pkts_moderate & orig_bytes_small & resp_pkts_moderate & resp_bytes_medium & orig_ip_bytes_medium & service_ssh & missed_bytes_none ⇒ label_Attack	0.910
C&C	resp_pkts_very_few & orig_bytes_very_small & resp_ip_bytes_very_small & conn_state_S0 & proto_tcp & service_none & missed_bytes_none & resp_bytes_very_small ⇒ label_C&C	0.889
Benign	conn_state_S0 & missed_bytes_none & resp_pkts_very_few & orig_bytes_very_small & service_none & resp_ip_bytes_very_small & resp_bytes_very_small ⇒ label_Benign	0.893
C&C-HeartBeat	missed_bytes_none & service_none & resp_pkts_very_few & resp_bytes_very_small & orig_bytes_very_small & resp_ip_bytes_very_small & proto_tcp ⇒ label_C&C-HeartBeat	0.857
C&C-HeartBeat-FileDownload	proto_tcp & missed_bytes_none & orig_pkts_very_many & orig_bytes_small & resp_pkts_very_many & resp_ip_bytes_very_large & conn_state_SF & service_http & resp_bytes_very_large ⇒ label_C&C-HeartBeat-FileDownload	1.000
C&C-Mirai	conn_state_RSTO & proto_tcp & orig_pkts_moderate & orig_ip_bytes_medium & orig_bytes_very_small & service_none & resp_bytes_very_small & resp_pkts_moderate & missed_bytes_none & resp_ip_bytes_small & duration_very_short ⇒ label_C&C-Mirai	1.000
C&C-Torii	orig_bytes_very_small & resp_pkts_very_few & proto_tcp & service_none & resp_bytes_very_small & missed_bytes_none & resp_ip_bytes_very_small ⇒ label_C&C-Torii	0.800
DDoS	resp_pkts_very_few & missed_bytes_none & resp_bytes_very_small & duration_very_short & proto_tcp & orig_bytes_very_small & orig_pkts_very_few & service_none & orig_ip_bytes_very_small & resp_ip_bytes_very_small ⇒ label_DDoS	1.000

Continues...

Table 3 – *Continues...*

Traffic Activity	**Rule**	**Support**
FileDownload	resp_bytes_very_large & orig_pkts_very_many & conn_state_SF & resp_ip_bytes_very_large & proto_-tcp & duration_short & service_http ⇒ label_File-Download	0.846
Okiru	missed_bytes_none & orig_pkts_very_few & resp_-bytes_very_small & duration_very_short & orig_-bytes_very_small & service_none & conn_state_S0 & resp_pkts_very_few & resp_ip_bytes_very_small & orig_ip_bytes_very_small & proto_tcp ⇒ label_Okiru	0.999
PartOfAHorizon-talPortScan	resp_pkts_very_few & resp_bytes_very_small & ser-vice_none & proto_tcp & conn_state_S0 & orig_-bytes_very_small & missed_bytes_none ⇒ label_-PartOfAHorizontalPortScan	0.998

Table 3 provides details on the key association rules discovered in the application of the Apriori, FP-Growth, and ECLAT algorithms on the preprocessed IoT-23 dataset. The key association rules derived were consistent for all three algorithms. Below is the explanation of each rule:

C&C-FileDownload

This rule indicates a direct correlation between the use of HTTP service and the transfer of very large files over TCP protocol, a common characteristic of C&C file download activities. The 100% support suggests that whenever these conditions are met, a C&C file download activity is always present.

Attack

The combination of medium-sized responses and moderate packet volumes over SSH services, TCP protocol usage, and a stable connection, is indicative of an attack attempt. The support of 0.91 indicates a significant pattern in the attack related data.

C&C

This rule shows the nature of C&C communication attempts, featuring very minimal traffic characteristics and the initial connection state (*S0*). With a support of 0.889, it indicates a significant pattern of low-volume data exchanges designed to establish or maintain control over compromised devices.

Benign

Benign activities are characterized by minimal traffic and the absence of data loss, indicating safe network interactions. The support of 0.893 suggests a significant traffic pattern across benign activities, serving as a baseline against which anomalous activities might be detected.

C&C-HeartBeat

C&C-HeartBeat traffic is characterized by minimal or low network traffic over TCP protocol, indicating periodic communication with a control server. The 0.857 support level indicates a significant frequency of such patterns, highlighting the need for constant alert for these small indications of security issues.

C&C-HeartBeat-FileDownload

This rule shows the scenario where malware not only indicates its presence through heartbeats but also actively downloads files. With a 100% support in the data across all C&C-HeartBeat-FileDownload interactions, the presence of very large IP response bytes and the use of HTTP service are particularly indicative.

C&C-Mirai

This rule, with 100% support, identifies the traffic pattern of the Mirai botnet, combining moderate traffic volume with specific TCP protocols and connection resets (*RSTO*), showing the distinctive characteristics of the Mirai botnet attack.

C&C-Torii

With a support of 0.8, this rule represents the Torii botnet's communication pattern, using minimal traffic and communication over TCP protocol. It shows the botnet's hidden operational strategies, making it a challenging malware to detect.

DDoS

This rule defines the basic but effective strategy behind DDoS attacks, characterized by very few packets and bytes transferred, over a very short period of time. The 100% support confirms the rule's universality in detecting DDoS activities.

FileDownload

This rule corresponds to file downloads, involving large volume data transfers, particularly through HTTP service, and suggests the potential for unauthorized or malicious file transfers. The support of 0.846 suggests the significant pattern of such traffic characteristics in download interactions.

Okiru

With nearly perfect support, the rule shows the Okiru traffic patterns, including minimal packets, very short durations, and TCP protocol found together. The high support level indicates the common occurrence of such patterns in Okiru related interactions.

PartOfAHorizontalPortScan

This rule indicates extensive port scanning activities, which often are an initial step to more targeted attacks. The antecedents capture a pattern where minimal or no data is exchanged, but connection attempts across ports are frequent. The very high support of this rule, suggests a major pattern across related interactions.

5.1.2 Algorithm Comparison

Table 4 provides the performance comparison of Apriori and FP-Growth application on the full preprocessed dataset. FP-Growth consistently outperformed Apriori in terms of execution time across all support thresholds. For example, at a support level of 0.9, Apriori takes about 61 seconds, whereas FP-Growth only requires 19 seconds. This trend is evident at lower support levels as well, where the time difference becomes even more significant, particularly at a support level of 0.7, where Apriori takes about 9 minutes, whereas FP-Growth only takes around 26 seconds. However, FP-Growth tends to use slightly more memory than Apriori. At the highest support level, the difference is maximal, about 133.7 MB more for FP-Growth. Both algorithms generate the same number of rules at each support level, suggesting that the rule generation capability is consistent between the algorithms, regardless of the execution time and memory usage differences.

Table 4. *Performance of Algorithms on Full Dataset*

Support	Algorithm	Number of Rules	Execution Time	Max Memory
0.9	Apriori	7096	61.4 seconds	1560.2 MB
	FP-Growth	7096	19.2 seconds	1693.9 MB
0.8	Apriori	11172	3 min 36 seconds	1584.0 MB
	FP-Growth	11172	23 seconds	1689.4 MB
0.7	Apriori	13980	9 min 6 seconds	1665.0 MB
	FP-Growth	13980	26.2 seconds	1776.5 MB

Table 5 provides the performance comparison of Apriori and FP-Growth application on half of the preprocessed dataset. The trend of FP-Growth outperforming Apriori in execution time continues even when the dataset size is halved. At each support level, Apriori's execution time is almost twice that of FP-Growth. In terms of maximum memory usage, both algorithms show almost identical memory usage patterns for all support levels. Similarly, both algorithms generate the same number of rules across different support levels, suggesting that the reduction in dataset size does not benefit one algorithm over the other in terms of rule generation.

Table 5. *Performance of Algorithms on Half of the Dataset*

Support	**Algorithm**	**Number of Rules**	**Execution Time**	**Max Memory**
0.9	Apriori	7156	20.9 seconds	1145.1 MB
	FP-Growth	7156	12.4 seconds	1145.1 MB
0.8	Apriori	11908	33.2 seconds	1192.7 MB
	FP-Growth	11908	16.3 seconds	1193.0 MB
0.7	Apriori	14004	48.7 seconds	1249.2 MB
	FP-Growth	14004	18.7 seconds	1250.2 MB

Table 6 provides the performance comparison of Apriori, FP-Growth, and ECLAT application on a quarter of the preprocessed dataset. The execution time advantage of FP-Growth over Apriori is consistent even with a quarter of the dataset. The inclusion of the ECLAT algorithm at this dataset size shows it to be significantly slower than both Apriori and FP-Growth, at all support levels. The ECLAT algorithm was not tested on the full and half-sized datasets due to its intensive computational demands. Other approaches proved too computationally intensive for practical execution within our environment. Thus, to maintain the focus on algorithm comparison rather than implementation, ECLAT's analysis was limited to a quarter of the dataset. In terms of maximum memory usage, ECLAT uses substantially more memory compared to Apriori and FP-Growth, likely due to its different approach to ARM. ECLAT generates significantly more rules than Apriori and FP-Growth, which could explain its longer execution times and higher memory usage. However, this could also indicate that ECLAT analyzes the itemsets with more depth.

Table 6. *Performance of Algorithms on Quarter of the Dataset*

Support	**Algorithm**	**Number of Rules**	**Execution Time**	**Max Memory**
0.9	Apriori	6211	16.6 seconds	844.5 MB
	FP-Growth	6211	9.1 seconds	844.9 MB
	ECLAT	24708	8 min 8 seconds	8449.8 MB

Continues...

Table 6 – *Continues...*

Support	Algorithm	Number of Rules	Execution Time	Max Memory Usage
0.8	Apriori	6399	17.1 seconds	844.5 MB
	FP-Growth	6399	8.9 seconds	844.3 MB
	ECLAT	25448	8 min 14 seconds	8465.8 MB
0.7	Apriori	8567	42.6 seconds	844.6 MB
	FP-Growth	8567	10.7 seconds	844.4 MB
	ECLAT	34102	13 min 2 seconds	8345.8 MB

This comparison suggests that FP-Growth is an optimal choice among the tested ARM algorithms for IoT data analyses, as it showed the best efficiency, consistently demonstrating shorter execution times and comparable memory usage to Apriori, across all tested support levels and dataset sizes. ECLAT, while capable of generating more rules, required substantially more time and showed considerably higher resource consumption, suggesting that it might be less suitable for large and memory-constrained IoT environments.

5.2 Machine Learning Results

5.2.1 Classification Reports

Table 7 presents the performance comparison for the ML classifiers applied. Here, the macro average metric computes the performance metric independently for each class and then takes the average, treating all classes equally. On the other hand, the weighted average takes into account the class imbalance by weighting the performance score of each class by its occurrences in the data. The results indicate that all three classifiers perform comparably in terms of precision, each achieving scores around 0.90. In terms of recall, the scores are slightly lower, approximately 0.86 for Decision Tree and Random Forest, and 0.85 for KNN. The F1-scores, mirror these results, also averaging around 0.86, indicating a balanced performance between precision and recall. The overall accuracy is also similar, with Decision Tree and Random Forest at 0.86, and KNN slightly lower at 0.85. The metrics show the effectiveness of each model in accurately classifying different network behaviors, with slight variations in their performance.

Table 7. *Overall Performance of Classifiers*

Metrics	**Decision Tree**		**Random Forest**		**K-Nearest Neighbors**	
	Weighted	**Macro**	**Weighted**	**Macro**	**Weighted**	**Macro**
Precision	0.90	0.90	0.90	0.90	0.89	0.89
Recall	0.86	0.86	0.86	0.86	0.85	0.85
F-1 Score	0.86	0.86	0.86	0.86	0.85	0.85
Accuracy	0.86		0.86		0.85	

Decision Tree Model

Table 8 provides details on the performance of the Decision Tree model in detecting various IoT traffic activity types. The model showed high precision across most activities, particularly in identifying malicious behavior. However, its recall varies significantly, especially for the '*Benign*' and '*C&C-Torii*' classes, indicating some difficulty in consistently identifying these behaviors. This suggests that they share characteristics with other classes or lack distinctive features, making accurate classification difficult. The F1-score for '*C&C*' of 0.68, despite a high recall of 0.99, suggests the model's tendency to overpredict this class, affecting its precision negatively.

Table 8. *Classification of Traffic Behavior for the Decision Tree Model*

Label	**Precision**	**Recall**	**F1-Score**
Attack	1.00	1.00	1.00
Benign	0.99	0.60	0.75
C&C	0.52	0.99	0.68
C&C-FileDownload	1.00	1.00	1.00
C&C-HeartBeat	1.00	0.89	0.94
C&C-HeartBeat-FileDownload	1.00	1.00	1.00
C&C-Mirai	1.00	1.00	1.00
C&C-Torii	1.00	0.54	0.70
DDoS	0.99	0.87	0.93
FileDownload	1.00	1.00	1.00
Okiru	0.63	0.91	0.74
PartOfAHorizontalPortScan	0.69	0.50	0.58

Random Forest Model

Table 9 provides details on the performance of the Random Forest model in detecting various IoT traffic activity types. The model performed with similar high precision on most classes, challenges were again evident in the '*Benign*' and '*C&C-Torii*' behaviors, with recall scores of 0.61 and 0.54. Additionally, the model's precision for '*C&C*' at 0.52, with a recall of 0.99, indicates overclassification in this category, similar to the Decision Tree model's performance. While both models showed similar metrics, the Random Forest is generally considered more robust due to its ensemble approach. This characteristic might not be evident in the overall accuracy, but can be crucial in more complex applications.

Table 9. *Classification of Traffic Behavior for the Random Forest Model*

Label	**Precision**	**Recall**	**F1-Score**
Attack	1.00	1.00	1.00
Benign	0.98	0.61	0.75
C&C	0.52	0.99	0.68
C&C-FileDownload	1.00	1.00	1.00
C&C-HeartBeat	1.00	0.89	0.94
C&C-HeartBeat-FileDownload	1.00	1.00	1.00
C&C-Mirai	1.00	1.00	1.00
C&C-Torii	1.00	0.54	0.70
DDoS	0.99	0.87	0.93
FileDownload	1.00	1.00	1.00
Okiru	0.63	0.91	0.74
PartOfAHorizontalPortScan	0.69	0.50	0.58

K-Nearest Neighbors Model

Table 10 provides details on the performance of the KNN model in detecting various IoT traffic activity types. The model demonstrated strong precision in identifying distinct malicious activities, similar to the other models. However, there was some variability in the recall, particularly for the '*Benign*' and '*C&C-Torii*' behaviors, where it similarly to the other models struggled to consistently identify instances correctly. The '*C&C*' class, with a precision of 0.50 and a high recall of 0.95, also indicated a tendency towards overclassification, but to a smaller extent than the tree-based models.

Table 10. *Classification of Traffic Behavior for the KNN Model*

Label	**Precision**	**Recall**	**F1-Score**
Attack	1.00	1.00	1.00
Benign	0.92	0.58	0.71
C&C	0.50	0.95	0.66

Continues...

Table 10 – *Continues...*

Label	**Precision**	**Recall**	**F1-Score**
C&C-FileDownload	1.00	1.00	1.00
C&C-HeartBeat	1.00	0.89	0.94
C&C-HeartBeat-FileDownload	1.00	1.00	1.00
C&C-Mirai	1.00	1.00	1.00
C&C-Torii	1.00	0.54	0.70
DDoS	0.96	0.87	0.91
FileDownload	1.00	1.00	1.00
Okiru	0.63	0.91	0.74
PartOfAHorizontalPortScan	0.68	0.47	0.56

Table 11 compares the performance of the ML models from the Related Work chapter's studies, which also used the IoT-23 dataset, alongside the results from the current work. Previous studies [41] and [42] employed Support Vector Machine (SVM), Naive Bayes (NB), and Decision Tree (DT) algorithms, achieving an accuracy of 0.73 and precision of 0.63 for DT. The study [44] evaluated SVM, Random Forest, Naive Bayes, and AdaBoost algorithms. The best performance was achieved by the Random Forest model with an accuracy of 1.0 and precision of 0.88. However, the dataset had to be split into smaller parts and encoded to have less categories, to manage computational issues. The study [43] evaluated the Decision Tree, Random Forest, Naive Bayes, and Bagging classifiers, with the Decision Tree achieving an accuracy of 0.96 and F-Measure of 0.76. However, this study removed the minority classes, which could have improved the accuracy by simplifying the classification problem and reducing noise from less representative classes.

In contrast, our work maintains the complexity of the original dataset, employing Decision Tree and Random Forest models to achieve an accuracy of 0.86 and a precision of 0.90, without extensive data modification. This approach ensures a more robust and generalizable model performance, suggesting predictive capabilities across all activities and attack classes. Additionally, the inclusion of the KNN algorithm broadens the scope of comparison.

Table 11. *Comparison of Performance with Related Studies*

Study	**ML Algorithms Used**	**Best**	**Performance**	**Notes**
[41], [42]	SVM, NB, DT	DT	Accuracy = 0.73, Precision = 0.63	-
[44]	SVM, RF, NB, AdaBoost	RF	Accuracy = 1.0, Precision = 0.88	Data split and re-encoded.
[43]	DT, RF, NB, Bagging	DT	Accuracy = 0.96, F-Measure = 0.76	Removed minority classes.
Present Work	DT, RF, KNN	DT, RF	Accuracy = 0.86, Precision = 0.90	-

Although our accuracy metrics may appear a bit lower in comparison to some studies, it is important to emphasize that our approach focuses on maintaining the complexity and diversity of the dataset. This ensures, that our models are tested against more realistic data, reflecting practical scenarios more accurately than those studies that modified the dataset extensively. Such alterations can potentially oversimplify the real-world applicability of the results.

5.2.2 Training and Prediction Times

Table 12 presents the average training and prediction times for the ML classifiers applied. These metrics are important as they reflect the applicability of models in real-time scenarios, where efficiency and speed are critical.

The Decision Tree model shows to be highly efficient, with a training time of 14.7 seconds and a prediction time of 0.16 seconds. This indicates that once trained, the Decision Tree model can make quick decisions. The Random Forest model requires more time to train, approximately 4 minutes and 24 seconds. However, its prediction time is moderately quick at 8.7 seconds. This indicates that while the model takes longer to train due to its ensemble nature, it can still make predictions relatively fast. The KNN model shows the fastest training time of 4.5 seconds, as this algorithm primarily involves storing training data rather than explicit model training. However, the prediction time is significantly longer, taking about 17 minutes. This is due to the need to compute the distance between a new point and all training points, which becomes computationally expensive as the dataset size increases.

Table 12. *Training and Prediction Times for Classifiers*

Model	Training Time	Prediction Time
Decision Tree	14.7 seconds	0.16 seconds
Random Forest	4 min 24 seconds	8.7 seconds
K-Nearest Neighbors	4.5 seconds	17 min 1 second

Overall, the Decision Tree and Random Forest models, with their reasonable prediction times, provide an efficient option for IoT environments, where fast processing is crucial. In contrast, the KNN model, despite its advantages in maintaining training simplicity, may not be as useful for real-time predictions due to its long prediction time.

5.3 ARM and ML Comparison

The ARM results provided easily understandable rules, identifying underlying patterns and associations within the dataset, and linking particular feature combinations with specific types of network activities. On the other hand, the ML models provided a predictive capability, handling complex patterns and allowing for the automated detection and classification of traffic types based on learned patterns.

While both approaches offer valuable insights, there are notable differences in their focus and application. ARM clarity is beneficial for straightforward pattern recognition, finding system vulnerabilities, and making model's decisions more understandable for humans, but it may struggle with capturing complex relationships and might not generalize well to unseen data. ML models demonstrated high predictive accuracy and the ability to capture more complex patterns, but their interpretability is rather limited compared to ARM.

The distinct capabilities of ARM and ML suggest that a combined approach could benefit from the strengths of both. Association rules and features derived from ARM can then be included as input features in ML models, potentially increasing their accuracy and reliability. Additionally, the predictive power of ML could validate and refine the findings from ARM, creating a more dynamic and effective tool for monitoring and securing IoT networks.

The comparative analysis shows that both ARM and ML techniques provide useful insights into behavior analysis and vulnerability prediction in IoT devices. Future work could explore the benefit of implementing a hybrid approach, combining ARM's interpretability and ML's predictive capability to build a more dynamic, precise, and scalable framework for proactive and more effective security in IoT environments.

6. Summary

This work focused on analyzing and predicting the vulnerabilities and behavior patterns of IoT devices using ARM and ML techniques. The main goal was to identify patterns and traffic characteristics indicative of various IoT traffic activity types and predict these potential security threats, allowing for proactive security measures.

In the first part of the work, ARM techniques were employed to analyze IoT network traffic data. Beginning with preprocessing the IoT-23 dataset to improve data quality and consistency for analysis. The ARM was executed using three algorithms: Apriori, FP-Growth, and ECLAT. Each algorithm was evaluated across various thresholds to assess their effectiveness in discovering meaningful association rules. The algorithms were applied to the entire dataset, as well as subsets to explore their performance under different data volumes. The process successfully generated a set of association rules, finding distinct traffic patterns for multiple types of IoT traffic activities. The comparative analysis of the algorithms revealed differences in execution time and memory usage, with FP-Growth generally outperforming other algorithms in efficiency, especially with larger datasets.

The second part of the work focused on the prediction of IoT network traffic types, classifying traffic as either benign or indicative of various malware activities. To ensure the consistency across both parts, the IoT-23 dataset was also used here. Data preprocessing included the normalization of traffic labels and transformation of categorical variables into binary columns through one-hot encoding. Three ML models were implemented: Decision Tree, Random Forest, and K-Nearest Neighbors. For each model, hyperparameter tuning and validation through cross-validation were performed to optimize performance and prevent overfitting. Each model was evaluated based on its accuracy, classification reports, as well as on training and prediction time. Both the Decision Tree and Random Forest models achieved an accuracy of 86%, but the Decision Tree was faster in both training and prediction phases. These models showed high precision in identifying malicious activities, but faced some challenges in consistently classifying the 'Benign' and 'C&C-Torii' traffic types, which appeared to have overlapping characteristics with other classes, affecting their recall and precision negatively. K-Nearest Neighbors model achieved slightly lower accuracy of 85% and was slower in the prediction phase. However, the model was still comparable in its ability to identify distinct malicious activities.

While both ARM and ML showed effectiveness in their respective areas, future work could consider a hybrid approach combining both methods to provide a more comprehensive solution. Future work could explore the integration of ARM derived rules as features in ML models to enhance accuracy and reliability, as well as develop methods to improve model interpretability and generalizability in IoT security.

References

[1] Antar Shaddad Abdul-Qawy et al. "The internet of things (iot): An overview". In: *International Journal of Engineering Research and Applications* 5.12 (2015), pp. 71–82.

[2] "Knowledge growth and development: internet of things (IoT) research, 2006–2018". In: *Heliyon* 5.8 (2019), e02264. DOI: `https://doi.org/10.1016/j.heliyon.2019.e02264`.

[3] International Data Corporation. *Worldwide Spending on the Internet of Things is Forecast to Surpass $1 Trillion in 2026.* [Accessed: 08-03-2024]. 2023. URL: `https://www.idc.com/getdoc.jsp?containerId=prUS50936423`.

[4] Hamed Taherdoost. "Security and internet of things: benefits, challenges, and future perspectives". In: *Electronics* 12.8 (2023). DOI: `https://doi.org/10.3390/electronics12081901`.

[5] Miloud Bagaa et al. "A machine learning security framework for iot systems". In: *IEEE Access* 8 (2020). DOI: `https://doi.org/10.1109/ACCESS.2020.2996214`.

[6] Wan Haslina Hassan et al. "Current research on Internet of Things (IoT) security: A survey". In: *Computer networks* 148 (2019), pp. 283–294. DOI: `https://doi.org/10.1016/j.comnet.2018.11.025`.

[7] Adrien Hemmer et al. "Comparative assessment of process mining for supporting IoT predictive security". In: *IEEE Transactions on Network and Service Management* 18.1 (2020), pp. 1092–1103. DOI: `10.1109/TNSM.2020.3038172`.

[8] Iqbal H Sarker et al. "Internet of things (iot) security intelligence: a comprehensive overview, machine learning solutions and research directions". In: *Mobile Networks and Applications* 28.1 (2023), pp. 296–312. DOI: `https://doi.org/10.1007/s11036-022-01937-3`.

[9] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[10] Feng Chen et al. "Data mining for the internet of things: literature review and challenges". In: *International Journal of Distributed Sensor Networks* 11.8 (2015), p. 431047. DOI: `https://doi.org/10.1155/2015/431047`.

[11] Akbar Telikani, Amir H Gandomi, and Asadollah Shahbahrami. “A survey of evolutionary computation for association rule mining”. In: *Information Sciences* 524 (2020), pp. 318–352. DOI: https://doi.org/10.1016/j.ins.2020.02.073.

[12] *Association Analysis: Basic Concepts and Algorithms*. [Accessed: 12-03-2024]. URL: https://www-users.cse.umn.edu/~kumar001/dmbook/ch6.pdf.

[13] Jiawei Han, Micheline Kamber, and Jian Pei. “6-mining frequent patterns, associations, and correlations: Basic concepts and methods”. In: *Data mining*. Morgan Kaufmann Boston. 2012, pp. 243–278.

[14] Julius Olufemi Ogunleye. “The Concept of Data Mining”. In: *Data Mining*. IntechOpen, 2021. DOI: 10.5772/intechopen.99417.

[15] Vahid Beiranvand, Mohamad Mobasher-Kashani, and Azuraliza Abu Bakar. “Multi-objective PSO algorithm for mining numerical association rules without a priori discretization”. In: *Expert systems with applications* 41.9 (2014), pp. 4259–4273. DOI: https://doi.org/10.1016/j.eswa.2013.12.043.

[16] Cynthia Rudin. *Prediction: Machine Learning and Statistics*. MIT OpenCourseWare: Massachusetts Institute of Technology. [Accessed: 23-05-2024]. 2012. URL: https://ocw.mit.edu/courses/15-097-prediction-machine-learning-and-statistics-spring-2012/resources/mit15_097s12_lec01/.

[17] Haosong Li and Phillip C-Y Sheu. “A scalable association rule learning heuristic for large datasets”. In: *Journal of Big Data* 8.1 (2021), p. 86. DOI: https://doi.org/10.1186/s40537-021-00473-3.

[18] *FP-Growth Algorithm in Data Mining*. [Accessed: 23-05-2024]. URL: https://www.javatpoint.com/fp-growth-algorithm-in-data-mining.

[19] Joos K. *The Eclat Algorithm*. [Accessed: 23-05-2024]. 2021. URL: https://medium.com/towards-data-science/the-eclat-algorithm-8ae3276d2d17.

[20] Fatima Hussain et al. “Machine Learning in IoT Security: Current Solutions and Future Challenges”. In: *IEEE Communications Surveys Tutorials* (2020).

[21] Adel Abusitta et al. “Deep learning-enabled anomaly detection for IoT systems”. In: *Internet of Things* 21 (2023), p. 100656. DOI: https://doi.org/10.1016/j.iot.2022.100656.

[22] Syeda Manjia Tahsien, Hadis Karimipour, and Petros Spachos. “Machine learning based solutions for security of Internet of Things (IoT): A survey”. In: *Journal of Network and Computer Applications* 161 (2020), p. 102630. DOI: https://doi.org/10.1016/j.jnca.2020.102630.

[23] Oona Rainio, Jarmo Teuho, and Riku Klén. “Evaluation metrics and statistical tests for machine learning”. In: *Scientific Reports* 14.1 (2024), p. 6086. DOI: https://doi.org/10.1038/s41598-024-56706-x.

[24] Isabella Lindgren. *Evaluation metrics for classification problems in machine learning*. [Accessed: 17-04-2024]. 2020. URL: https://towardsdatascience.com/evaluation-metrics-for-classification-problems-in-machine-learning-d9f9c7313190.

[25] Malti Bansal, Apoorva Goyal, and Apoorva Choudhary. “A comparative analysis of K-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning”. In: *Decision Analytics Journal* 3 (2022), p. 100071. DOI: https://doi.org/10.1016/j.dajour.2022.100071.

[26] GopenAI. *Decision Tree Algorithm*. [Accessed: 23-05-2024]. 2023. URL: https://blog.gopenai.com/decision-tree-algorithm-484ec33387f9.

[27] Rasheed Ahmad and Izzat Alsmadi. “Machine learning approaches to IoT security: A systematic literature review”. In: *Internet of Things* 14 (2021), p. 100365. DOI: https://doi.org/10.1016/j.iot.2021.100365.

[28] Naphaporn Sirikulviriya and Sukree Sinthupinyo. “Integration of rules from a random forest”. In: *International Conference on Information and Electronics Engineering*. Vol. 6. 2011, pp. 194–198.

[29] Jim Jeffers, James Reinders, and Avinash Sodani. “Chapter 24 - Machine learning”. In: *Intel Xeon Phi Processor High Performance Programming (Second Edition)*. Morgan Kaufmann, 2016.

[30] Maarten Grootendorst. *9 distance measures in Data Science*. [Accessed: 26-04-2024]. 2021. URL: https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa.

[31] Adrian Rosebrock. *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. [Accessed: 23-05-2024]. 2020. URL: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[32] Javed Asharf et al. "A review of intrusion detection systems using machine and deep learning in internet of things: Challenges, solutions and future directions". In: *Electronics* 9.7 (2020), p. 1177. DOI: https://doi.org/10.3390/electronics9071177.

[33] Seiichi Ozawa et al. "A study of IoT malware activities using association rule learning for darknet sensor data". In: *International Journal of Information Security* 19 (2020), pp. 83–92. DOI: https://doi.org/10.1007/s10207-019-00439-w.

[34] Romil Rawat et al. "Association rule learning for threat analysis using traffic analysis and packet filtering approach". In: *International Journal of Information Technology* 15.6 (2023), pp. 3245–3255. DOI: https://doi.org/10.1007/s41870-023-01353-0.

[35] Fatemeh Safara, Alireza Souri, and Masoud Serrizadeh. "Improved intrusion detection method for communication networks using association rule mining and artificial neural networks". In: *IET Communications* 14.7 (2020), pp. 1192–1197. DOI: https://doi.org/10.1049/iet-com.2019.0502.

[36] Elisa Bertino and Nayeem Islam. "Botnets and Internet of Things Security". In: *Computer* 50.2 (2017), pp. 76–79. DOI: 10.1109/MC.2017.62.

[37] Jadel Alsamiri and Khalid Alsubhi. "Internet of things cyber attacks detection using machine learning". In: *International Journal of Advanced Computer Science and Applications* 10.12 (2019). DOI: https://doi.org/10.14569/IJACSA.2019.0101280.

[38] Mahmudul Hasan et al. "Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches". In: *Internet of Things* 7 (2019), p. 100059. DOI: https://doi.org/10.1016/j.iot.2019.100059.

[39] Andrew Churcher et al. "An experimental analysis of attack classification using machine learning in IoT networks". In: *Sensors* 21.2 (2021), p. 446. DOI: https://doi.org/10.3390/s21020446.

[40] Prachi Shukla. "ML-IDS: A machine learning approach to detect wormhole attacks in Internet of Things". In: *2017 intelligent systems conference (IntelliSys)*. IEEE. 2017, pp. 234–240. DOI: 10.1109/IntelliSys.2017.8324298.

[41] Y Liang and N Vankayalapati. *Machine Learning and Deep Learning Methods for Better Anomaly Detection in IoT-23 Dataset Cybersecurity*. [Accessed: 20-04-2024]. 2022. URL: https://github.com/yliang725/Anomaly-Detection-IoT23.

[42] Falaq Jeelani et al. “The detection of IoT botnet using machine learning on IoT-23 dataset”. In: *2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM)*. Vol. 2. IEEE. 2022, pp. 634–639. DOI: 10.1109/ICIPTM54933.2022.9754187.

[43] Alfredo Nascita et al. “Machine and deep learning approaches for IoT attack classification”. In: *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2022, pp. 1–6. DOI: 10.1109/INFOCOMWKSHPS54753.2022.9797971.

[44] Nicolas-Alin Stoian. *Machine Learning for anomaly detection in IoT networks : Malware analysis on the IoT-23 data set*. [Accessed: 27-04-2024]. 2020. URL: http://essay.utwente.nl/81979/.

[45] Sebastian Garcia, Agustin Parmisano, and Maria Jose Erquiaga. *IoT-23: A labeled dataset with malicious and benign IoT network traffic*. http://doi.org/10.5281/zenodo.4743746. Dataset. 2020.

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

I Paula Pakina

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Behavior Analysis and Prediction of Vulnerabilities in IoT Devices Using Data Mining and Machine Learning Techniques”, supervised by Mohammad Reza Heidari Iman and Tara Ghasempouri
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

27.05.2024

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