



University of
Zurich^{UZH}

Data Exploration and Feature Engineering for an IoT Device Behavior Fingerprinting Dataset

Lucas Krauter
Zurich, Switzerland
Student ID: 22-740-369

Supervisor: Dr. Alberto Huertas Celdran, Chao Feng
Date of Submission: 30. Juli 2024

Independent Study
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: <http://www.csg.uzh.ch/>

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Additionally, I acknowledge generative AI usage to linguistically refine the text of my report and AI-powered code assistance, to support my coding tasks and ensure adherence to best practices and standards.

Zürich,

Signature of student

Zusammenfassung

Die zunehmende Verbreitung von Geräten im Internet der Dinge (IoT) hat gezeigt, wie wichtig effektive Mechanismen zur Erkennung von Schadsoftware sind. Allerdings fehlt es an öffentlich zugänglichen Datensätzen, um IoT-basierte Bedrohungen zu erkennen, was die Forschung erschwert. Um diese Lücke zu schließen, habe ich den IoT-Geräte-Verhaltens-Fingerabdruck-Datensatz genutzt, der im Rahmen eines Masterprojekts an der Universität Zürich entwickelt wurde. Dieser Datensatz erfasst verschiedene Verhaltensweisen von IoT-Geräten und Angriffsszenarien. Ziel meiner Studie ist es, die Erkennung von Schadsoftware durch verbesserte Techniken im Feature-Engineering und in der Merkmalsauswahl zu optimieren.

In diesem Projekt habe ich die Term-Frequency-Inverse-Document-Frequency-Methode (TF-IDF) auf SystemCall-Merkmale angewendet und einen mehrstufigen Auswahlansatz entwickelt. Zunächst wurden konstante Merkmale entfernt, dann eine univariate Merkmalsauswahl durchgeführt, um die Relevanz einzelner Eigenschaften zu bewerten. Aus 260 Merkmalen wählte ich die besten 32 mit einer modellbasierten Methode aus, um den Datensatz mit minimalen Auswirkungen auf die Malware-Erkennung zu reduzieren. Dieses optimierte Feature-Set führte zu einem vernachlässigbaren Verlust von 0,0055 in der F1-Punktzahl, während die effektive Klassifizierungsleistung über verschiedene Gerätefamilien hinweg erhalten blieb.

Zusätzlich habe ich gezeigt, dass der Datensatz sich für das Training von Geräten in föderierten Lernszenarien eignet, was entscheidend zur Verbesserung der Cybersicherheit in IoT-Umgebungen beiträgt. Diese Studie bietet wertvolle Einblicke in effektive Methoden der Merkmalsauswahl und betont die Nützlichkeit des Datensatzes für die Weiterentwicklung der Erkennung von Schadsoftware in IoT-Umgebungen.

Abstract

The increasing prevalence of Internet of Things (IoT) devices has highlighted the critical need for effective malware detection mechanisms. The scarcity of publicly available datasets for IoT malware detection has constrained research progress. To address this gap, I utilized the IoT Device Behavior Fingerprinting Dataset, developed as part of a Master's project at the University of Zurich, which captures diverse IoT device behaviors and malware attack scenarios. This study aims to enhance malware detection by refining feature engineering and selection processes applied to this dataset.

For the contributions, I incorporated Term Frequency-Inverse Document Frequency (TF-IDF) to the SystemCall features and adopted a multi-step feature selection approach. Initially, constant features were removed, followed by Univariate Feature Selection to evaluate feature relevance. From 260 features, I selected the top 32 using a Model-based Feature Selection method, which reduced the dataset with minimal impact on performance. My refined feature set resulted in a negligible loss of 0.0055 in the F1 score while also maintaining effective classification performance across device families (RP3 and RP4).

Furthermore, I demonstrated that the dataset is suitable for training nodes in federated learning scenarios, which is crucial for enhancing cybersecurity in IoT environments. This study provides valuable insights into effective feature selection methods and underscores the dataset's utility for advancing IoT malware detection.

Contents

Declaration of Independence	i
Abstract	iii
1 Introduction	1
2 Related Work	3
3 Dataset Description	7
3.1 Raw Data Organization	7
3.1.1 Data Source	9
3.1.2 Attack Labels	10
3.1.3 Data Statistics	10
3.2 Encoded Data	11
4 Experimental Design, Materials And Methods	13
4.1 Feature Engineering	13
4.1.1 Entropy Data	14
4.1.2 Network Data	14
4.1.3 System Call and Process Data	16
4.2 Feature Exploration	17
4.3 Feature Selection	18
4.3.1 Removing features with low variance	18
4.3.2 Univariate Feature Selection Scores	18

4.3.3	Feature Scoring and Visualization	19
4.3.4	Model-based Feature Selection	21
4.4	Evaluation Setup	22
5	Results	25
5.1	Experiment 1: Feature Selection	26
5.2	Experiment 2: Source Device Comparison	26
5.3	Experiment 3: Federated Learning	27
5.4	Experiment 4: Decentralized Federated Network Topology	28
5.5	Experiment 5: Decentralized Federated Network with Unbalanced Data Distributions	29
6	Conclusion	31
7	Limitations	33
	Bibliography	35
	List of Figures	35
	List of Tables	38
A	Additional Raw Dataset Figures	41
B	Additional Feature Selection Figures	43

Chapter 1

Introduction

As the Internet of Things (IoT) continues to transform the way we live and work, the sheer scale of interconnected devices has created a complex ecosystem that is increasingly vulnerable to cyber threats. The consequences of a successful attack can be catastrophic, with compromised devices providing a foothold for malicious actors to launch further attacks or steal sensitive information. In response, researchers have been working to develop robust IoT malware detection systems. However, the scarcity of publicly accessible datasets for IoT malware detection has hindered research in this area, and traditional machine learning approaches have limitations in adapting to dynamic IoT data and privacy concerns. Recently, a decentralized federated learning (DFL) model was proposed to address these challenges, and a comprehensive dataset was compiled to capture both normal and anomalous device behavior under malware attacks [1].

Building upon this dataset, my work aims to further analyze and improve the feature engineering and selection process to enhance the detection and classification of malware targeting IoT devices. In this article, I describe the steps taken to refine the dataset, including data statistics, encoding, feature exploration, and selection. I then evaluate the performance of different training approaches and discuss the results and limitations of my study.

Chapter 2

Related Work

Extensive research has been conducted in the field of cybersecurity, focusing on anomaly detection and malware classification. To identify potential malware, statistic based anomaly detection techniques, such as One-Class Support Vector Machines (SVM) and Local Outlier Factor (LOF), have been employed to detect patterns that diverge from normal system behavior. Additionally, machine learning-based classification methods, encompassing both supervised and unsupervised learning, have been utilized to categorize malware into distinct families and types. For example, researchers have leveraged features extracted from API calls, system calls, and network traffic to train classification models, achieving high accuracy in malware identification. However, the constantly evolving nature of malware and the increasing complexity of attack vectors have created a pressing need for more robust and adaptive detection and classification techniques, thus motivating further research to explore innovative solutions and push the boundaries of existing methodologies.

IoT Device Behavior Fingerprinting Dataset. The dataset used here was developed as part of a Master's project undertaken by four students at the University of Zurich [1]. The primary objective of this project was to address the lack of available IoT malware detection datasets for researchers in the field of IoT security.

In addition to creating the raw dataset, the students also investigated potential scenarios for training anomaly detection and malware classification algorithms using this IoT malware detection dataset. Furthermore, they explored the application of the dataset in a federated training setup using Fedstellar [2], examining various aspects of this approach. In addition, they provide a comprehensive review of anomaly detection and malware classification algorithms [1].

Log Analysis. In the realm of log analysis for anomaly detection, various approaches have been proposed. Conventional methods employ log parsers to extract essential elements, known as log keys, from log files. By counting the frequency of specific keys over time windows, each window can be modeled, effectively encoding the log file for machine learning applications. For instance, [3] adopts this approach, incorporating a TF-IDF algorithm to weight the importance of each key. The resulting vector is then utilized for

binary classification using a Long Short-Term Memory (LSTM) recurrent neural network. However, these approaches generally require domain expertise to tailor the log parser to retrieve relevant information.

Recently, promising novel approaches have shifted towards utilizing Bidirectional Encoder Representations from Transformers (BERT) models. For example, [4] presents a self-supervised method that can adapt to unseen log structures using unlabeled raw log data, eliminating the need for manual adjustments and labeling data.

Network anomaly detection pipeline. [1] created a machine learning system that collects data, extracts features, and classifies IoT traffic to detect DDOS attacks. The features are specifically designed to take advantage of unique IoT network behaviors, as well as characteristics of network flow, such as packet size, time between packets, and communication protocol. The pipeline consists of the following steps:

1. Traffic Capturing on a network middleboxes (e.g. routers, firewalls, or network switches)
2. Grouping of Packets by Device and sampling into time-frames
3. Feature Extraction
4. Binary Classification: Anomaly Detection

Feature Selection. [5] presents Scikit-learn, an open-source Python library that offers a range of tools for data mining and data analysis. Among its extensive functionality, Scikit-learn provides robust feature selection methods, crucial for enhancing model performance by selecting relevant features and eliminating redundant or irrelevant ones. Two notable feature selection techniques included in Scikit-learn are Low Variance Feature Selection and Univariate Feature Selection.

Low Variance Feature Selection focuses on removing features with low variance, which are unlikely to be informative for the model. Features that exhibit little variation across samples are considered less useful for distinguishing between different classes or predicting target values. By eliminating these low variance features, the model can avoid overfitting and improve generalization to new data.

Univariate Feature Selection involves selecting the best features based on univariate statistical tests. This method evaluates each feature individually to determine its relationship with the target variable, using techniques such as chi-squared tests, ANOVA F-tests, or mutual information. This approach is straightforward and effective for identifying significant features that contribute to the predictive power of a model.

According to [6], various approaches to feature selection are reviewed. The paper categorizes feature selection methods into three classes: Wrappers, Filters, and Embedded methods. Wrappers employ the learning machine of interest as a black box to evaluate

subsets of variables based on their predictive capabilities. Filters select subsets of variables as a preprocessing step, independent of the chosen predictor. Embedded methods perform variable selection during the training process and are typically specific to particular learning machines. Wrappers can be further divided into forward selection and backward elimination processes. Forward selection begins with an empty set of features and recursively adds the highest-scoring one, whereas backward elimination starts with all features and recursively removes some. The paper also discusses Greedy search strategies. In the backward step, instead of evaluating the performance for each feature to be removed, the least important feature is eliminated.

Chapter 3

Dataset Description

This chapter provides a detailed description of the dataset. Some of the data is in textual format and needs to be encoded using feature engineering techniques. The first subsection explains the overall structure of the raw data, while the second subsection describes the structure of the processed data.

3.1 Raw Data Organization

The raw data has a complex structure, which is explained below. The raw dataset consists of recordings from IoT devices in various malware scenarios, comprising a diverse range of files. The typical organizational structure of these files is shown in figure 3.1. The number of files associated with each data source is presented in Table 3.1. It is worth noting that entropy files are missing when there were no entropy changes during the respective time window.

Data Source	Count of files
SYS_data	92
KERN_data	91
FLS_data	91
RES_data	91
block_data	91
network_data	91
entropy_data	75

Table 3.1: Count of files per Data Source

The files can be classified into multiple domains, as illustrated in figure 3.2. I will begin with a general overview, followed by a more detailed explanation of the Data Source and Classification Labels.

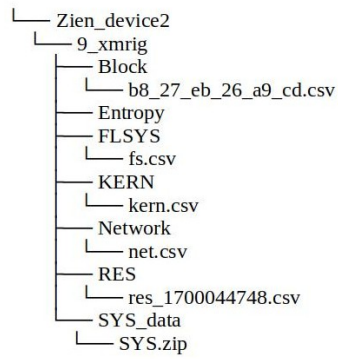


Figure 3.1: Dataset Files Structure

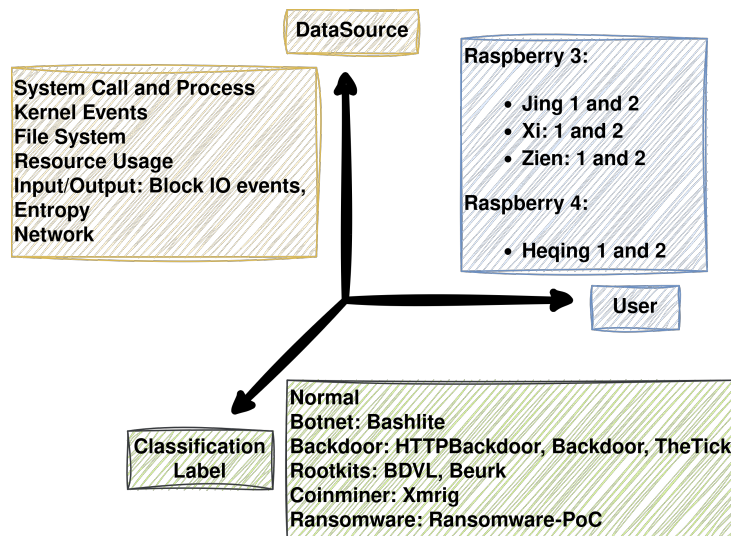


Figure 3.2: Data Axis

User. The dataset is organized along three axes. Firstly, the X-axis represents the different user devices that were monitored and recorded. Each student manually recorded data from two devices, resulting in a total of eight devices. All students used the Raspberry 3 model, except for Heqing, who used two Raspberry 4 devices. It is important to note that devices of different generations, such as Raspberry 3 and 4, may not behave similarly. For instance, the Raspberry 4 has more RAM, which means that features related to free memory will inherently differ between Raspberry 3 and 4 devices. Thus, the raw dataset is splitted into the different users. These files are stored under the respective name of their creator.

Data Source. Secondly, the Y-axis represents the various data sources, which capture different aspects of the IoT device's behavior and store them in the dataset. These categories offer diverse features, such as log files of running processes and system calls, as well as file system activities. These features were researched and developed by [1] in their project.

Classification Label. Thirdly, the Z-axis represents the different labels assigned to the recordings, which were captured in both normal operating scenarios and scenarios where the device was actively attacked by various malware. These labels serve as the target output for classification algorithms, with the goal of predicting which attack is currently running on the machine given the input data.

3.1.1 Data Source

The data sources represent a set of features collected by the same monitoring script. To maintain a clean structure, the students divided the dataset into several source domains based on the origin of the data. The selected data sources were carefully chosen by the students following a thorough literature review on features for malware classification. They then developed customized scripts to record each individual feature of the data source. All the features were collected as time-series data, meaning that the state of the device was recorded at regular intervals, capturing its behavior over time. However, something to keep in mind is that the selected intervals are different per data sources, making it necessary to align the features at some point. Overview over the data sources:

- **System calls and Processes:** Invoked System-calls, Processes and their statistics such as CPU time. The data from this source initially only consists of a collection of log recordings, one for each timestamp. I bundle them into Zip files for each recording session.
- **Kernel events:** Various system metrics: disk I/O, CPU usage, kernel memory
- **File System:** all file system-related perf events
- **Resource Usage:** device's hardware statistics

- **Input/Output:**
 - { Block IO events: input/output statistics related to devices and partitions
 - { Entropy: file creation and modification events
- **Network:** network traffic from network interface

3.1.2 Attack Labels

As mentioned earlier, the devices were intentionally infected with various malware to collect features during an attack. These different types of malware also serve as labels for the classification task.

- **Normal:** Not infected with any malware.
- **Botnet:** The device acts as a client in a botnet, querying commands from an external command and control system that manages the botnet.
- **Backdoors:** The device runs malware that allows unauthorized access to the system.
 - { HttpBackdoor: The device can receive commands via HTTP requests, such as executing a command.
 - { Backdoor: Executes a reverse shell; similar to a botnet, the device queries commands from a command server.
 - { TheTick: A botnet where the device listens for commands from the server but using advanced communication channels.
- **Rootkits:** Malware that hides within the operating system but still provides a backdoor.
 - { Beurk: A rootkit for GNU/Linux that focuses on avoiding detection by exploiting debugging mechanisms.
 - { Bdvl: A toolset that includes hidden backdoors.
- **Coinminer:** The device is used for mining digital coins using XMRig.
- **Ransomware:** The malware encrypts all files on the system.

3.1.3 Data Statistics

As discussed in the previous chapter, some of the raw data is not encoded (such as system log files). Consequently, I only focus on the number of data points collected. I present several plots illustrating the distribution of data across the domains previously outlined. Figure 3.3 reveals that the Raspberry 3 has a significantly larger number of data points

compared to the Raspberry 4. A more detailed distribution of data across different users is provided in figure A.1 in the appendix.

In figure 3.4 one can see that the dataset is well balanced across all classes (labels) of the dataset.

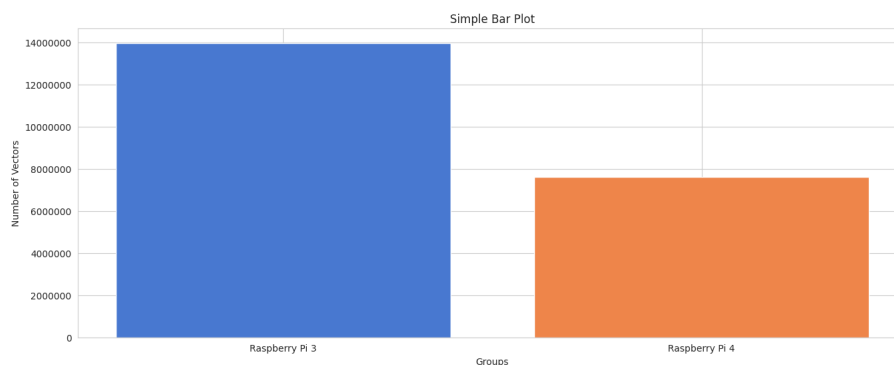


Figure 3.3: Number of Vectors per Device Family

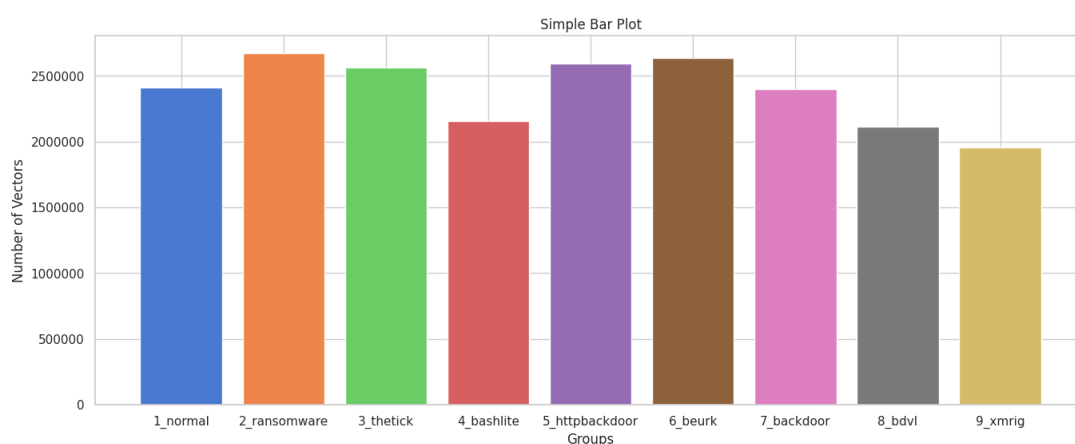


Figure 3.4: Number of Vectors per class

Besides simply counting the vectors one can also plot the distribution over time. Figure 3.5 shows the distribution over time, effectively displaying when each device was used to record data. In the appendix, you can find other distributions over time.

3.2 Encoded Data

Some parts of the raw data are additionally encoded because they were textual or ordinal features. The following gives an overview:

- **Network:** This subsection covers data captured from network interfaces. The raw data, consisting of sniffed packets, is encoded into scalable features, such as the `TcpUdpProtocolRatio`.

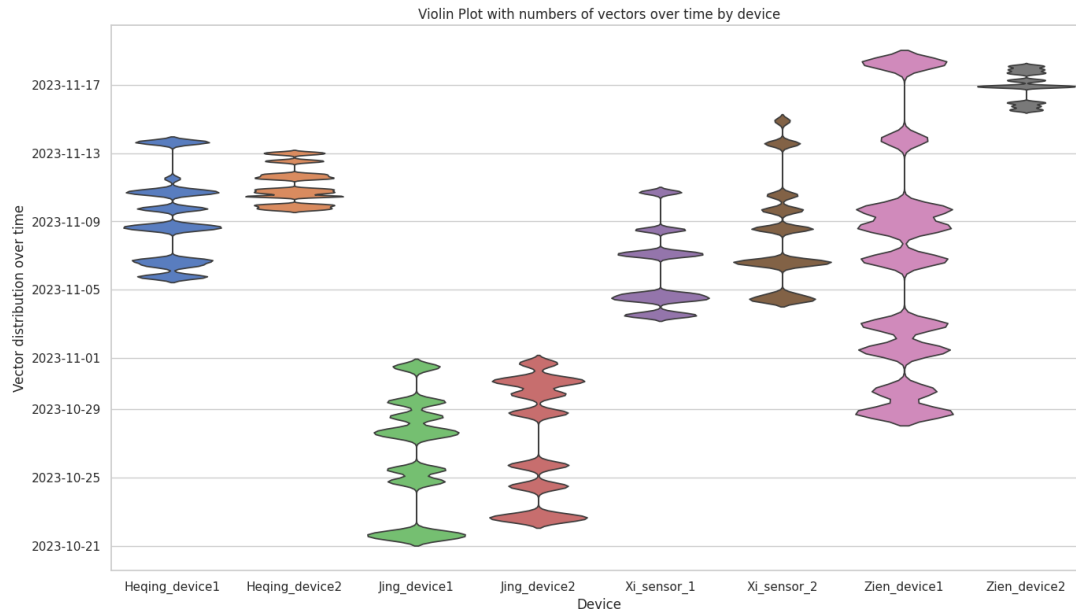


Figure 3.5: Vector distribution over time by device

- **Entropy:** The entropy is aggregated and iterated within a short time window, following the method used in [1].
- **System Log Files:** I used a Bag of Words (BOW) approach as described in [1] and also implemented additional methods.

Chapter 4

Experimental Design, Materials And Methods

In this study, I aim to utilize the dataset to classify malware specifically designed for IoT devices. My objective is to investigate the effectiveness of the dataset in training a neural network for malware classification. To achieve this, I adopt a three-step approach:

1. I encode all features in the dataset, which involves feature engineering for the data sources of Entropy Data, Network, and System Calls.
2. I select the most promising features from the dataset, which initially contains around 700 features depending on the encoding method. My goal is to create a compact dataset of approximately 30 features that are most relevant for malware classification.
3. I train a neural network to evaluate the dataset's performance in classifying malware. I also test the dataset in the proposed federated learning setup.

4.1 Feature Engineering

Most raw data sources have undergone encoding already, where features are extracted and transformed into a suitable format for analysis. For example, the Block IO Events data source includes a feature that counts write operations within a specific time interval. However, it is essential to note that these data sources have been recorded at varying time intervals, necessitating the merging of datasets through resampling to a uniform time interval.

In addition to this, feature engineering is required for the Entropy Data, Network, and System Calls data sources. The encoding approaches for these data sources are described in the following subsections.

4.1.1 Entropy Data

The feature engineering process for entropy data follows the methodology described in [1].

When a file is either modified or created, its entropy value is recorded. For the purpose of this study, the entropy value is calculated based on the first 100 bytes of the given file.

To systematically analyze the entropy data, the recorded values were partitioned into 20-second time windows. Within each time window, I calculated the number of files that had entropy values greater than or equal to 6. This threshold was chosen based on preliminary observations indicating it as a marker of significant entropy changes.

The resulting feature from this process is termed *entropy_file_count*. This feature represents the count of files within each 30-second interval that have high entropy values, specifically those greater than or equal to 6.

4.1.2 Network Data

Network data in this context originates from raw packet information. Each packet is recorded with the following features: Protocol, SourceIP, SourcePort, DestIP, DestPort, and Length. This data is aggregated into 40-second time windows for further analysis.

Feature design involves transforming raw network data into a set of representative features that can be used for various analytical and machine learning tasks. This chapter outlines feature extraction methods from the two main sources, the master project and a study on Machine learning ddos detection for consumer internet of things devices [1], [7].

Basic Features

Basic features are derived directly from the packet data within each time window [1]. The total number of packets in each 40-second window is recorded as "PacketCount".

Aggregations of packet lengths within the time window provide insights into traffic characteristics:

- **Total Length:** Sum of all packet lengths ("TotalLength").
- **Average Length:** Mean length of packets ("AverageLength").
- **Median Length:** Median length of packets ("MedianLength").
- **Minimum Length:** Minimum packet length ("MinLength").
- **Maximum Length:** Maximum packet length ("MaxLength").
- **Variance Length:** Variance of packet lengths ("VarianceLength").

Ports are crucial for identifying communication patterns:

- **Different Source Ports:** Count of unique source ports ("DifferentSourcePorts").
- **Different Destination Ports:** Count of unique destination ports ("DifferentDestPorts").

Advanced Features

Advanced features consider additional characteristics of the network data, including protocol information and temporal aspects [7].

Utilizing protocol information enhances feature richness:

- **TCP Packet Count:** Number of TCP packets ("TcpPacketCount").
- **UDP Packet Count:** Number of UDP packets ("UdpPacketCount").
- **TCP/UDP Protocol Ratio:** Ratio of TCP to UDP packets ("TcpUdpProtocolRatio").

Time intervals between packets are important for temporal analysis:

- **Mean Inter-Packet Interval:** Average time between packets ("MeanInterPacketInterval").
- **Variance Inter-Packet Interval:** Variance of inter-packet intervals ("VarianceInterPacketInterval").
- **Minimum Inter-Packet Interval:** Shortest time between packets ("MinInterPacketInterval").
- **Maximum Inter-Packet Interval:** Longest time between packets ("MaxInterPacketInterval").
- **First Derivative of Inter-Packet Interval:** Change rate of inter-packet intervals ("FirstDerivativeInterPacketInterval").
- **Second Derivative of Inter-Packet Interval:** Acceleration of inter-packet interval changes ("SecondDerivativeInterPacketInterval").

Examining the length of packets over time provides bandwidth-related features:

- **Average Bandwidth:** Mean bandwidth consumption ("AverageBandwidth").
- **Variance Bandwidth:** Variance in bandwidth usage ("VarianceBandwidth").
- **Minimum Bandwidth:** Lowest bandwidth consumption ("MinBandwidth").

- **Maximum Bandwidth:** Highest bandwidth consumption ("MaxBandwidth").

The number of different destination IPs provides insights into the diversity of communication:

- **Different Destination IPs:** Count of unique destination IPs ("DifferentDestIPs").

These features collectively enable a comprehensive analysis of network traffic, aiding in the detection of patterns and anomalies for various applications such as network monitoring, intrusion detection, and performance analysis.

4.1.3 System Call and Process Data

My approach to system call and process data analysis is based on the method outlined in [1]. Specifically, I collect raw data on all system calls executed within a short time window. To encode this data, I employ a Bag of Words (BOW) approach, where each time window is treated as a document, and the system calls within it are represented as words.

In addition to the BOW approach, I incorporate the term frequency-inverse document frequency (TF-IDF) method to provide a more nuanced representation of system call data. TF-IDF extends the BOW approach by assigning weights to each system call based on its frequency of occurrence. The term frequency component of TF-IDF counts the occurrences of each system call within a document, while the inverse document frequency component measures the rarity of the system call across all documents. This means that rare system calls are assigned higher weights, as they convey more information than commonly occurring ones.

The output of the TF-IDF algorithm is a set of values for each system call in each document. I store these values in a database for further analysis. In plot 4.1 one can see that the distribution of output values is in a range from 0 to 1.

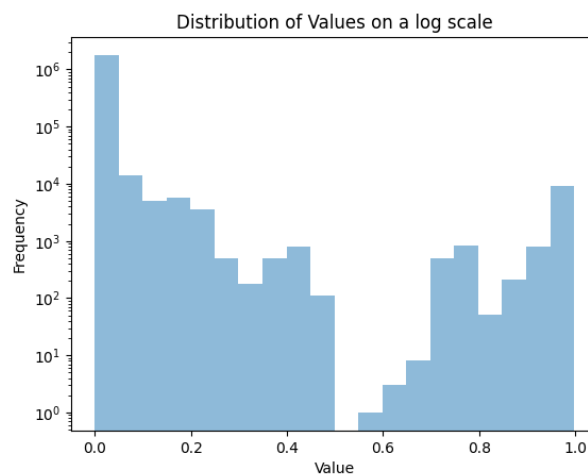


Figure 4.1: Distribution of assigned values to system calls for the device Heqing device2

4.2 Feature Exploration

To gain insights into the dataset, I performed basic data exploration by examining the distribution of various features with respect to their labels. This analysis helps in understanding the underlying patterns and relationships within the data, which can be crucial for feature selection and model building.

In this feature exploration part, I solely focus on the device "Heqing_device2", because it has the most samples. The features were normalized across all labels before plotting.

The following serves as an example. One of the features explored in detail was related to input/output monitoring, specifically *Block IO events*. This feature captures the input/output statistics concerning devices and partitions. The result of this exploration can be seen in figure 4.2.

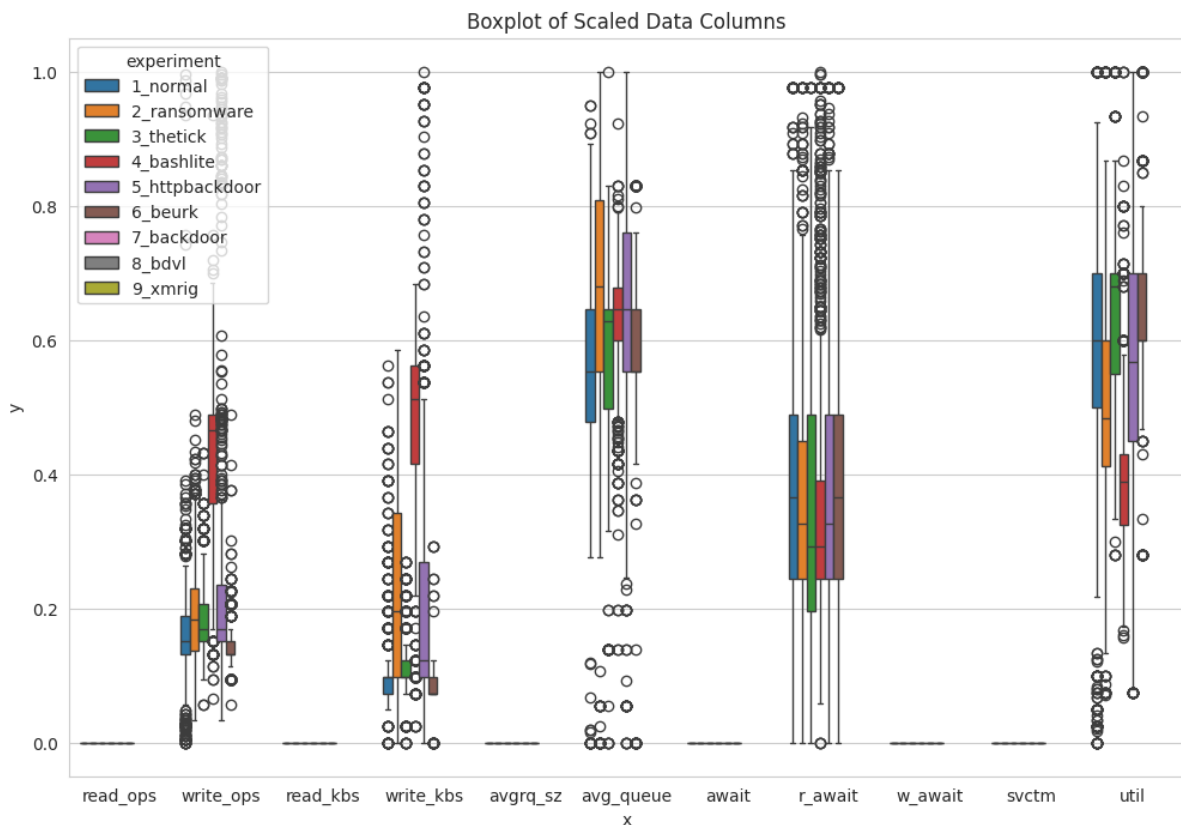


Figure 4.2: Data Distribution of each feature colored for each label of the Block IO events

Further distributions for additional features, along with their respective analyses, are provided in the public code repository ¹. This extended exploration allows for a comprehensive understanding of how each feature behaves in relation to the labels, facilitating informed decisions in the feature engineering process.

¹Link to additional feature analysis plots https://github.com/LucK1Y/iot-feature-engineering/tree/main/plots/data_exploration_boxplots

4.3 Feature Selection

The first feature selection process for the dataset is based on methodologies described in [5]. Effective feature selection is crucial for enhancing the performance of machine learning models by reducing overfitting, improving accuracy, and decreasing training times.

I adopt a multi-step approach to feature selection to identify the most informative features in the dataset. The first step involves removing constant features, as they do not provide any valuable information for my analysis. Next, I apply Univariate Feature Selection, which operates under the assumption that each feature is independent. This method measures the correlation between each feature and each label, allowing us to evaluate the relevance of each feature individually. I then select the top 260 features based on their correlation scores. Finally, I perform a Model-based Feature Selection on the top 32 features to further refine my selection. This step involves training a machine learning model on the selected features and evaluating their importance based on the model's weights.

4.3.1 Removing features with low variance

The first step in my feature selection process involves removing features with low variance. I use the *VarianceThreshold* with a threshold of 0. This step eliminates features that have the same value across all samples, as they do not provide any useful information for the model.

4.3.2 Univariate Feature Selection Scores

Univariate feature selection involves ranking the features to identify the top-k features for each label. This process helps in selecting features that have a significant relationship with the target variable. I employ the following statistical tests for this purpose:

Chi-Squared Test (*chi2*)

The *chi2* test computes the chi-squared statistics between each non-negative feature and the class label. This score is used to select the features with the highest chi-squared values, indicating a stronger dependency on the class labels. The chi-squared test helps in weeding out features that are likely to be independent of the class, thus irrelevant for classification.

ANOVA F-value (*f_classif*)

The *f_classif* function computes the ANOVA F-value for the provided samples. This test assesses the significance of the features by comparing the variance between the classes

to the variance within the classes. Features with higher F-values are more likely to be relevant for distinguishing between classes.

Mutual Information (`mutual_info_classif`)

The `mutual_info_classif` function estimates the mutual information between each feature and the discrete target variable. Mutual information measures the dependency between variables, with higher values indicating a greater dependency. This method relies on nonparametric entropy estimation from k-nearest neighbors distances, providing a robust measure of feature relevance.

4.3.3 Feature Scoring and Visualization

I calculated the scores of each feature for each label using the aforementioned methods. These scores help in identifying the most relevant features for the classification tasks. Below is a plot 4.3 illustrating the distribution of feature scores across all features for the data from the block event monitoring.



Figure 4.3: Scores for each feature of the Block data source averaged across all labels.

However, this ranking might not lead to the best features for the dataset. Therefore, I added an even more detailed analysis, scoring each feature for each label. In plot 4.4, one can clearly see, that the feature ranks very high only for a certain subset of labels.

I can also put these values into a heat-map, that can show the scores of a single statistical test for each feature and label. In figure 4.5, I pick the *Mutual Information* Score.

As soon as I calculated the scores, I can select the top k features on each statical test for each label. I plotted the scores for the top 5 features for label 1: normal in figure 4.6.

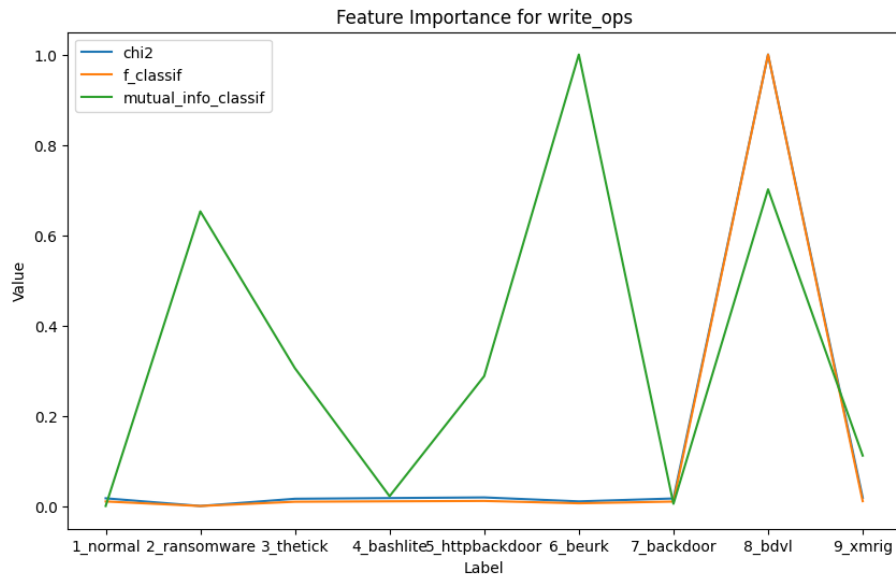


Figure 4.4: Performance metrics for the *write_ops* feature of the Block data source. Note that the Chi-Squared values align perfectly with the f-classif values for the last two labels, rendering them indistinguishable and therefore not visible.

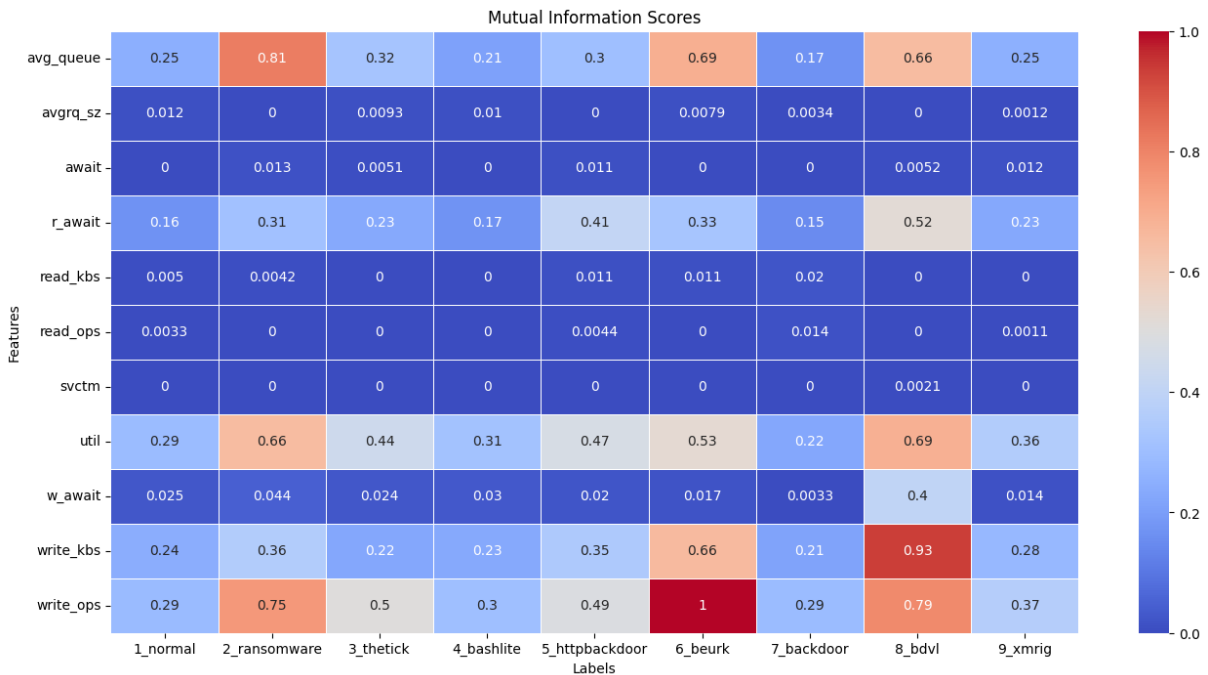


Figure 4.5: Heatmap of the *Mutual Information* Score across all labels and features of the block event subset

For more detailed plots and additional feature analyses, please refer to the appendix B. There I added figures for other labels.

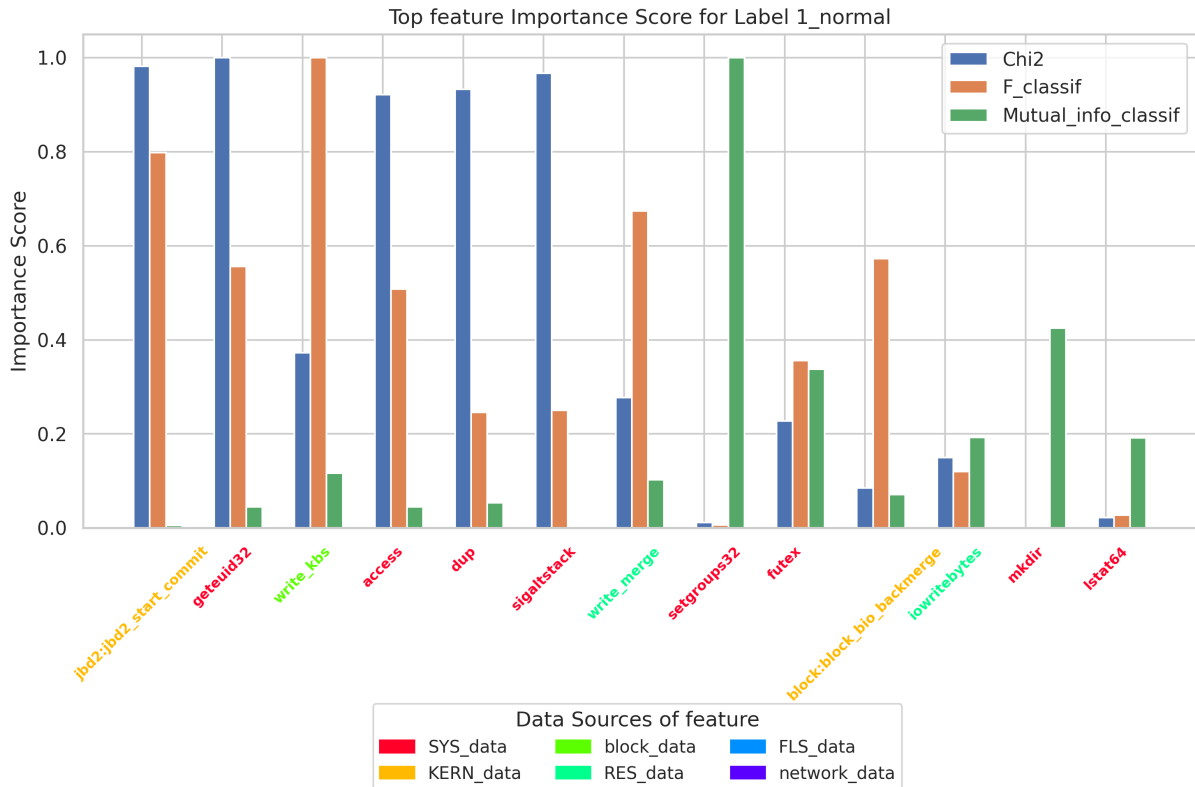


Figure 4.6: Top 5 features for each statistical test are plotted for label 1: **normal device operation**. The color of the feature title indicates the data source.

4.3.4 Model-based Feature Selection

In this step, I further reduce the feature set by employing a linear model to select the most important features. This approach, as categorized by [6], is a type of Wrapper method that utilizes a backward elimination process. Specifically, I use the correlation between feature weights and class labels to determine feature importance. Instead of recursively eliminating features, which would require training the model multiple times, I opt for a single feature selection step to minimize computational overhead and reduce the training process to a single iteration.

At this stage, I require a unified dataset, which involves integrating the disparate raw data files from various data sources. To achieve this, I resample each data source to align with a common time window of 30 seconds and then merge them based on these synchronized time intervals.

The linear model consists of a single layer with a weight matrix of dimensionality $\mathbf{n}_{input} \times \mathbf{n}_{output}$, where \mathbf{n}_{input} and \mathbf{n}_{output} represent the number of input features and output classes, respectively. I apply the softmax function to obtain the model's predictions. To train the model, I use a training-validation split of 80:20 and employ the EarlyStopping algorithm to monitor the accuracy on the validation set, thereby preventing overfitting on the training set.

After training, I analyze the learned weights, whose distribution is shown in Figure 4.3.4.

Notably, the weights for class 1 are significantly larger than those for class 9. Therefore, I select the top weights per class, rather than the overall top weights. Specifically, I choose the top 4 weights per class, resulting in a total of 32 weights. Finally, I select the input features corresponding to these 32 weights as my next set of selected features. An overview over the plots that shows all the distributions per class can be found in the appendix at B.9.

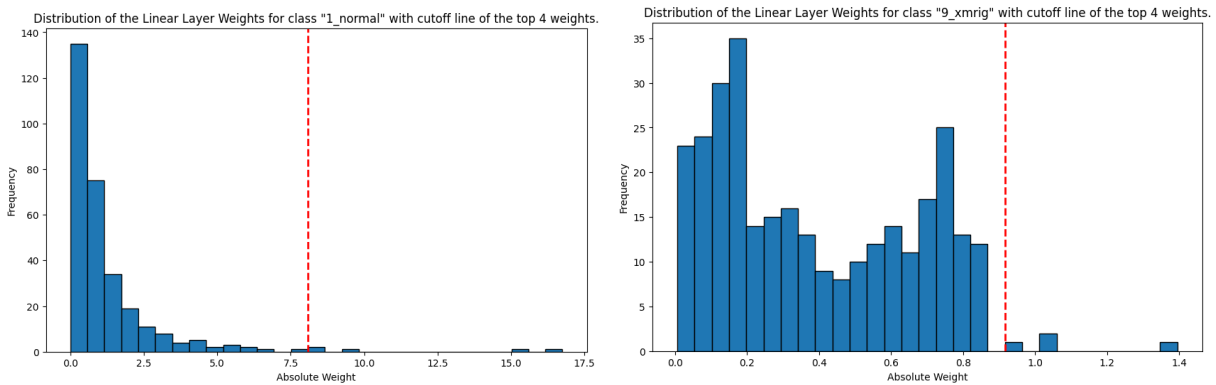


Figure 4.7: Distribution of Linear Layer weights for class 1 and class 9. The red line indicates the cutoff line.

4.4 Evaluation Setup

To assess the quality of the resulting dataset, I employ an extrinsic evaluation method. This involves training a machine learning model on various subsets of the dataset and measuring its performance. Specifically, I adopt the training scenarios proposed by [1] as the basis for my evaluation.

I report the performance metrics of the trained model on a test split, including Accuracy, F1 score, Loss, Precision, and Recall.

I run the following experiments

- Feature selection: I compare the model's performance on the Heqing_device2 subset using 260 features versus 32 features, selected based on their prominence using model-based feature selection.
- Device comparison: I evaluate the model's performance when trained on 32 features from Heqing_device2, Xi_sensor_1, and a merged dataset combining both.
- Federated learning: I utilize FedStellar to examine the model's performance in different federated learning scenarios, including centralized federated learning (CFL) and decentralized federated learning (DFL). In the CFL setup, several nodes are utilized, comprising one central server (aggregator) that stores the model weights and some participant nodes that train the model locally. In contrast, the DFL setup

consists of several fully connected nodes, where each node assumes both participant and aggregator roles, training locally and aggregating the propagated weights from neighboring nodes. In all federated scenarios, I employ either a subset of the Heqing_device2 data or the complete dataset comprising data from all devices. Additionally, I vary the number of nodes to demonstrate the transferability of the dataset across different settings.

- Decentralized Federated Learning: FedStellar also enables the simulation of decentralized federated learning scenarios, where there is no central server to aggregate weights and coordinate learning. Instead, nodes are organized in a decentralized manner, relying on voting mechanisms to facilitate learning, and weights are shared only with directly connected neighbors. Multiple simulation scenarios are available. The typologies can be separated into the following setups:
 - { Fully connected networks, where every node is connected to each other node.
 - { Ring topologies, where each node has exactly two neighbors.
 - { Star topologies, where a single node connects all others, although this node is not a central server and actively participates in the learning process.

In addition to the experiments on topologies, I also investigate the impact of different data distributions on the performance of the decentralized federated learning network. Specifically, I compare the results of using a uniform data distribution versus a Dirichlet distribution. The latter approach simulates a more realistic scenario, where some nodes may not have access to a balanced dataset for training, and may even lack exposure to certain classes altogether.

Chapter 5

Results

This chapter presents the outcomes of the methodology described in the previous chapter. I first present the selected features and then discuss the training results of neural networks across various scenarios.

Univariate Feature Selection. In the initial feature selection phase, my objective was to create a dataset comprising approximately 300 features. To achieve this, I selected the top 200 features according to each univariate feature selection score. Since the statistical tests yielded similar rankings for most features, this process resulted in a total of 260 features¹. Figure 5.1 illustrates the distribution of data sources for the selected features, revealing that the majority originate from system calls of the log files.

Model-based Feature Selection. In the subsequent stage of feature selection, I further refined the dataset by identifying the top 4 features associated with each of the 9 classes,

¹A list of the selected features is available at the public repository at https://github.com/Luck1Y/iot-feature-engineering/blob/main/top_260_features.txt.

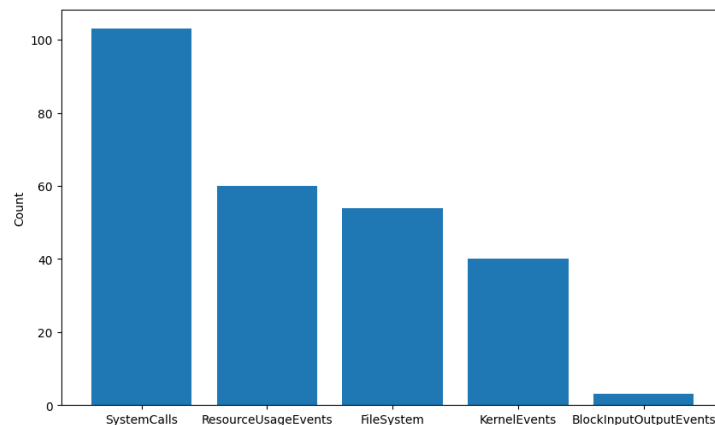


Figure 5.1: Data sources of the Univariate Feature Selection.

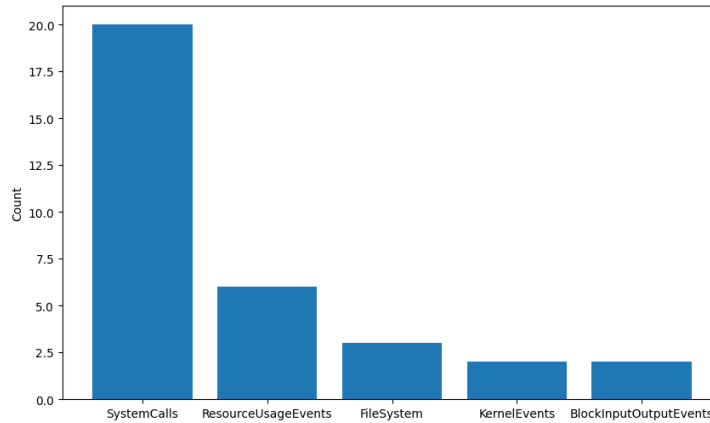


Figure 5.2: Data sources of the Model-based Feature Selection.

ultimately yielding a total of 32 features². The distribution of data sources in the resulting dataset is depicted in Figure 5.2, which reveals an even clearer preference for the system calls data source.

5.1 Experiment 1: Feature Selection

As previously discussed, I utilize the weights of a single linear layer to identify the most salient features. As evident in Table 5.1, the single layer model achieves a notable F1 score of 0.95. By applying the weights to select the top 32 features, I successfully reduce the dataset from 260 to 32 features, with a minimal loss of 0.0055 in the F1 score. These results support the effectiveness of my feature selection method. Notably, the number of training steps varies due to the implementation of early stopping, which prevents the model from overfitting.

Model Layers	Input Dimension	Accuracy	F1	Precision	Recall	Steps
1	306	0.9524	0.9516	0.9519	0.9524	980
3	306	0.9971	0.9971	0.9973	0.9971	1225
3	32	0.9920	0.9916	0.9915	0.9920	1645

Table 5.1: Performance metrics for different model architectures

5.2 Experiment 2: Source Device Comparison

Thus far, my experiments have been limited to the Heping_device2 subset, which corresponds to a Raspberry Pi 4. In this experiment, I investigate the transferability of the feature selection by evaluating its performance on a Raspberry Pi 3. To achieve this, I

²A list of the selected features is available at the public repository at https://github.com/LucK1Y/iot-feature-engineering/blob/main/top_32_features.txt.

select the subset from device `Xi_sensor_1` and train the same three-layer model with 32 input features on this dataset. I then compare the results with the previous scores obtained using the `Heqing_device2` dataset and a combined dataset comprising both. The comparable performance of both subsets on the F1 metric suggests that the feature selection method is effective for both device families.

Dataset	Accuracy	F1	Precision	Recall	Steps
<code>Heqing_device2</code>	0.9920	0.9916	0.9915	0.9920	1645
<code>Xi_sensor_1</code>	0.9859	0.9861	0.9868	0.9859	455
Both	0.9868	0.9868	0.9871	0.9868	3195

Table 5.2: Performance metrics for different sub-datasets

5.3 Experiment 3: Federated Learning

This experiment involves a comparative analysis of various federated learning scenarios. To ensure a fair comparison, I adopt a traditional machine learning approach with a fixed number of epochs, thereby maintaining a consistent number of training steps across all scenarios. In the Centralized Federated Learning (CFL) setup, I have n nodes, where one node serves as the aggregator and the remaining $n-1$ nodes act as participants, each training a local model. As shown in Table 5.3, the Decentralized Federated Learning (DFL) setup employs a fully connected network topology.

I reran this experiment using the complete dataset and a more diverse set of nodes, and the results are presented in Table 5.4. Additionally, I provide the standard deviation across the nodes, as this value may increase with a larger number of nodes.

The results indicate that on the smaller dataset, the three approaches exhibit similar performance, which I attribute to the limited dataset size and node count. In such cases, the results are closer to those obtained through classical machine learning approaches.

In contrast, on the larger dataset, I observe several notable trends. Firstly, while the performance of Centralized Federated Learning (CFL) improves with an increase in node count, Decentralized Federated Learning (DFL) exhibits a decline. I attribute this to the aggregator node in CFL benefiting from a more diverse set of learned weights, leading to enhanced overall performance on the dataset. In the fully connected DFL setup, the increase in nodes results in the dataset being distributed across more devices, leading to decreased performance. Secondly, the standard deviation in the CFL setup is significant, suggesting that some nodes did not learn as effectively within the federated learning framework.

Scenario	Nodes	Accuracy	F1	Precision	Recall	Steps
ML	1	0.9564	0.9535	0.9571	0.9564	5450
CFL	3	0.9499	0.9480	0.9533	0.9499	5450
DFL	3	0.9558	0.9535	0.9579	0.9558	5450

Table 5.3: Average Performance metrics for different training scenarios on the subset of `Heqing_device2`.

Scenario	Nodes	Accuracy			F1	Precision			Recall	Steps
ML	1	0.9447			0.9369	0.9474			0.9447	76970
CFL	4	0.7656	0.16	0.7331	0.18	0.7498	0.18	0.7656	0.16	5437
CFL	8	0.6652	0.19	0.6265	0.22	0.6531	0.22	0.6652	0.19	5269
CFL	16	0.9242	0.16	0.9119	0.17	0.9141	0.17	0.9242	0.16	13108
DFL	4	0.8858	0.00	0.8665	0.00	0.8835	0.00	0.8858	0.00	11110
DFL	8	0.8318	0.04	0.8064	0.04	0.8342	0.04	0.8318	0.04	11517
DFL	16	0.7286	0.09	0.6792	0.10	0.6895	0.10	0.7286	0.09	6590

Table 5.4: Average Performance metrics for different training scenarios over the **complete** dataset.

5.4 Experiment 4: Decentralized Federated Network Topology

In this experiment, I investigate various decentralized federated network topologies. I evaluate their performance on both the small dataset and the complete dataset.

As shown in Table 5.5, on the small dataset, the fully connected network outperforms the other two topologies. However, this trend does not hold for the complete dataset, as shown in Table 5.6. Instead, the star setup with four nodes achieves the best performance. Across all topologies, I observe a consistent decline in performance as the number of nodes increases, suggesting that decentralized federated networks do not scale well with an increasing number of nodes.

Scenario Description	Node counts	Accuracy	F1	Precision	Recall
Fully	15	0.9385	0.9340	0.9387	0.9386
Star	15	0.9164	0.9084	0.9166	0.9164
Ring	15	0.9058	0.8979	0.9080	0.9058

Table 5.5: Average Performance metrics for different Decentralized federated Networks on the subset of `Heqing_device2`.

5.5. EXPERIMENT 5: DECENTRALIZED FEDERATED NETWORK WITH UNBALANCED DATA DISTRIBUTIONS

Scenario	Nodes	Accuracy			F1	Precision		Recall	Steps	
Fully	4	0.8858	0.00	0.8665	0.00	0.8835	0.00	0.8858	0.00	11110
Fully	8	0.8318	0.04	0.8064	0.04	0.8342	0.04	0.8318	0.04	11517
Fully	16	0.7286	0.09	0.6792	0.10	0.6895	0.10	0.7286	0.09	6590
Ring	4	0.8948	0.03	0.8754	0.03	0.8921	0.03	0.8948	0.03	15183
Ring	8	0.8204	0.04	0.7901	0.04	0.8138	0.04	0.8204	0.04	6213
Star	4	0.9002	0.01	0.8840	0.01	0.8987	0.01	0.9002	0.01	6927
Star	8	0.8350	0.05	0.8063	0.05	0.8304	0.05	0.8350	0.05	5015

Table 5.6: Average Performance metrics for different training scenarios over the complete dataset.

5.5 Experiment 5: Decentralized Federated Network with Unbalanced Data Distributions

In this experiment, I train a fully connected Decentralized Federated Learning (DFL) network with an unbalanced data distribution. Specifically, I split the dataset across nodes according to a Dirichlet distribution, which results in nodes having access to only a subset of labels and learning partial representations. However, through the benefits of federated learning, nodes can learn from each other's shared weights and compensate for the lack of data.

The results of this experiment are presented in Table 5.7. The alpha value controls the distribution, with high values (e.g., 10) resulting in a balanced split and low values (approaching zero) leading to highly skewed and heterogeneous subsets.

My results show that the performance of the DFL network decreases as the alpha value decreases. This is expected, as some nodes may not have seen certain classes and therefore cannot learn from them. Nevertheless, the network still achieves an impressive performance of around 80% on the F1 score, even when multiple complete classes are not seen. This has significant implications for real-world scenarios, where a node can identify malware it was not trained on, which is essential for defending against cyber attacks. Furthermore, this experiment demonstrates the robustness of DFL in handling unbalanced data distributions, which is a common challenge in many real-world applications.

Alpha	Accuracy	F1	Precision	Recall	Steps
1	0.8397	0.7933	0.7822	0.8397	7768
10	0.9508	0.9436	0.9513	0.9508	9836

Table 5.7: Average Performance metrics for a DFL: fully connected, 8 nodes over the complete dataset.

Chapter 6

Conclusion

This report presents the results of a comprehensive study on the application of neural networks and federated learning to malware detection. My experiments demonstrate the effectiveness of feature selection methods in reducing the dimensionality of the dataset while maintaining performance. The results show that the selected features are predominantly from system calls of log files, highlighting their importance in malware detection.

The experiments also demonstrate the robustness of federated learning in various scenarios. In particular, the decentralized federated learning approach shows promising results, even with unbalanced data distributions. This has significant implications for real-world applications, where data is often heterogeneous and distributed across multiple nodes.

In conclusion, this study demonstrates the potential of neural networks and federated learning in malware detection. The results provide insights into the importance of feature selection, node count, and topology in decentralized federated learning. The ability of decentralized federated learning to handle unbalanced data distributions makes it a promising approach for real-world applications, where data is often heterogeneous and distributed across multiple nodes.

Chapter 7

Limitations

While my proposed approach has shown promising results in detecting and classifying malware in IoT devices, there are several limitations to my study that should be acknowledged.

One of the main limitations of my study is that I only experimented with two device models raspberry 3 and 4, representing a limited range of IoT devices. This may not be representative of the diverse range of devices that exist in real-world IoT ecosystems. Future studies should aim to include a more comprehensive range of devices to ensure the generalizability of the results.

My experiments were conducted mostly using 30-second windows of data, which may not be optimal for all scenarios. In fact, a limitation is the detection delay associated with using 30-second windows. This means that there is a 30-second delay between the time an attack occurs and the time it is detected. While this may be acceptable for some use cases, it may not be suitable for applications that require real-time detection and response. Future research should focus on developing approaches that can detect attacks in real-time or with minimal delay.

In conclusion, while my study has made significant contributions to the field of IoT malware detection, there are several limitations that should be acknowledged. Addressing these limitations will be crucial to developing more effective and practical solutions for detecting and mitigating malware attacks in IoT devices.

Bibliography

- [1] J. Han, H. Ren, X. Cheng, and Z. Zeng, "Creation of new datasets for decentralized federated learning", Master Project, University of Zurich, 2024.
- [2] E. T. Martinez Beltran, A. L. Perales Gomez, C. Feng, *et al.*, "Fedstellar: A platform for decentralized federated learning", *Expert Systems with Applications*, vol. 242, p. 122861, May 2024, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2023.122861. [Online]. Available: <http://dx.doi.org/10.1016/j.eswa.2023.122861>.
- [3] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechris, and H. Zhang, "Automated it system failure prediction: A deep learning approach", in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 1291-1300. DOI: 10.1109/BigData.2016.7840733.
- [4] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "Log t: Log anomaly detection using fine-tuned language models", *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1715-1723, 2024. DOI: 10.1109/TNSM.2024.3358730.
- [5] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.
- [6] I. Guyon and A. Elisseev, "An introduction to variable and feature selection", *J. Mach. Learn. Res.*, vol. 3, no. null, pp. 1157-1182, Mar. 2003, ISSN: 1532-4435.
- [7] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices", in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 29-35. DOI: 10.1109/SPW.2018.00013.

List of Figures

3.1	Dataset Files Structure	8
3.2	Data Axis	8
3.3	Number of Vectors per Device Family	11
3.4	Number of Vectors per class	11
3.5	Vector distribution over time by device	12
4.1	Distribution of assigned values to system calls for the device Heqing device2	16
4.2	Data Distribution of each feature colored for each label of the Block IO events	17
4.3	Scores for each feature of the Block data source averaged across all labels. .	19
4.4	Performance metrics for the <i>write_ops</i> feature of the Block data source. Note that the Chi-Squared values align perfectly with the f-classif values for the last two labels, rendering them indistinguishable and therefore not visible.	20
4.5	Heatmap of the <i>Mutual Information</i> Score across all labels and features of the block event subset	20
4.6	Top 5 features for each statistical test are plotted for label 1: normal device operation . The color of the feature title indicates the data source.	21
4.7	Distribution of Linear Layer weights for class 1 and class 9. The red line indicates the cut-off line.	22
5.1	Data sources of the Univariate Feature Selection.	25
5.2	Data sources of the Model-based Feature Selection.	26
A.1	Number of Vectors per Device	41
A.2	Vector distribution over time by label	42

A.3	Vector distribution over time by data source	42
B.1	Top 5 features for each statistical test are plotted for label 2: ransomware attack . The color of the feature title indicates the data source.	44
B.2	Top 5 features for each statistical test are plotted for label 3: thetick attack . The color of the feature title indicates the data source.	44
B.3	Top 5 features for each statistical test are plotted for label 4: bashlite attack . The color of the feature title indicates the data source.	45
B.4	Top 5 features for each statistical test are plotted for label 5: httpbackdoor attack . The color of the feature title indicates the data source.	45
B.5	Top 5 features for each statistical test are plotted for label 6: beurk attack . The color of the feature title indicates the data source.	46
B.6	Top 5 features for each statistical test are plotted for label 7: backdoor attack . The color of the feature title indicates the data source.	46
B.7	Top 5 features for each statistical test are plotted for label 8: bdv1 attack . The color of the feature title indicates the data source.	47
B.8	Top 5 features for each statistical test are plotted for label 9: xmrig coin-miner attack . The color of the feature title indicates the data source.	47
B.9	Weights Distribution of the Linear Layer per Class. The red line shows the cut-off line after the biggest 4 weights.	48

List of Tables

3.1	Count of files per Data Source	7
5.1	Performance metrics for different model architectures	26
5.2	Performance metrics for different sub-datasets	27
5.3	Average Performance metrics for different training scenarios on the subset of Heqing_device2	28
5.4	Average Performance metrics for different training scenarios over the complete dataset.	28
5.5	Average Performance metrics for different Decentralized federated Networks on the subset of Heqing_device2	28
5.6	Average Performance metrics for different training scenarios over the complete dataset.	29
5.7	Average Performance metrics for a DFL: fully connected, 8 nodes over the complete dataset.	29

Appendix A

Additional Raw Dataset Figures

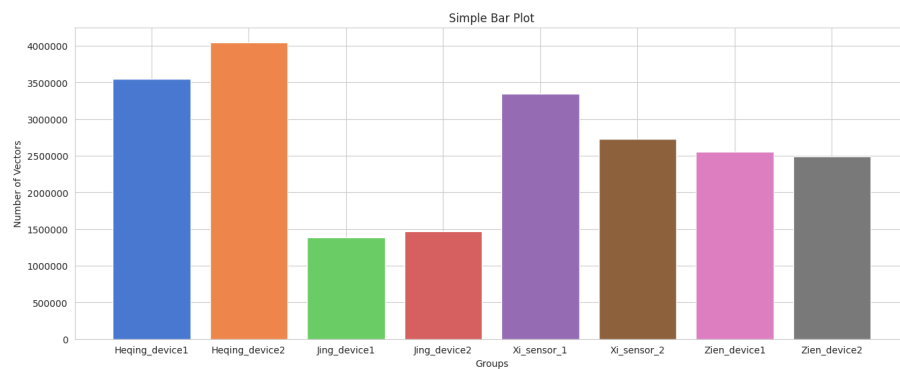


Figure A.1: Number of Vectors per Device

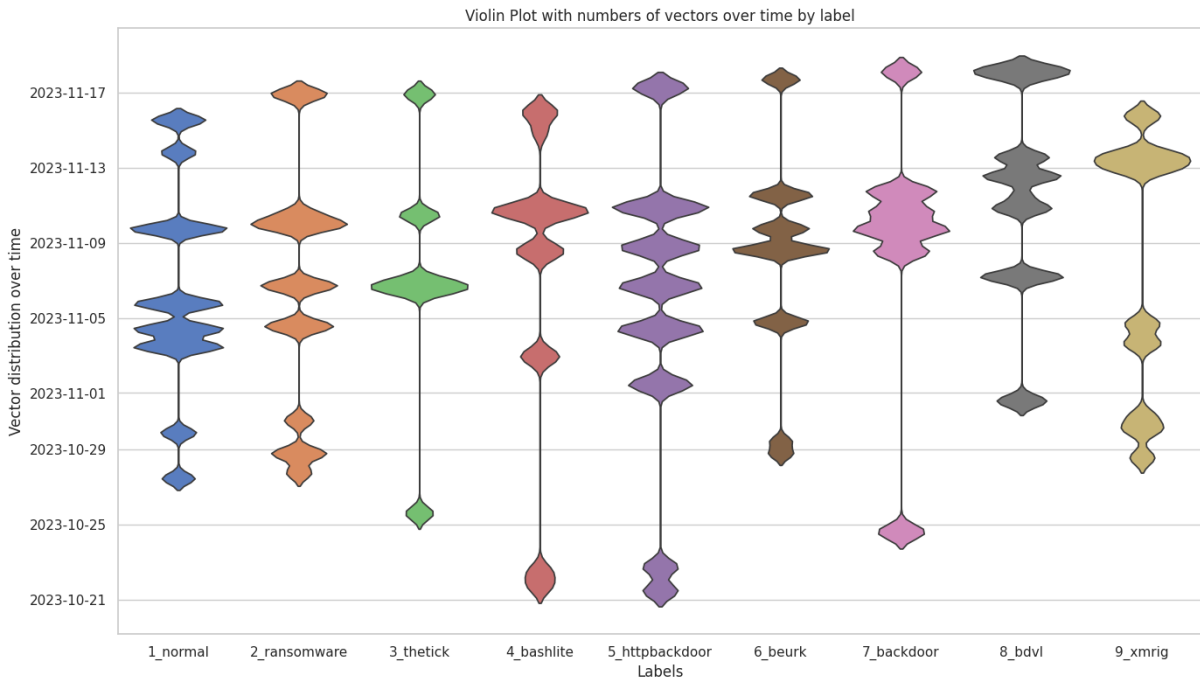


Figure A.2: Vector distribution over time by label

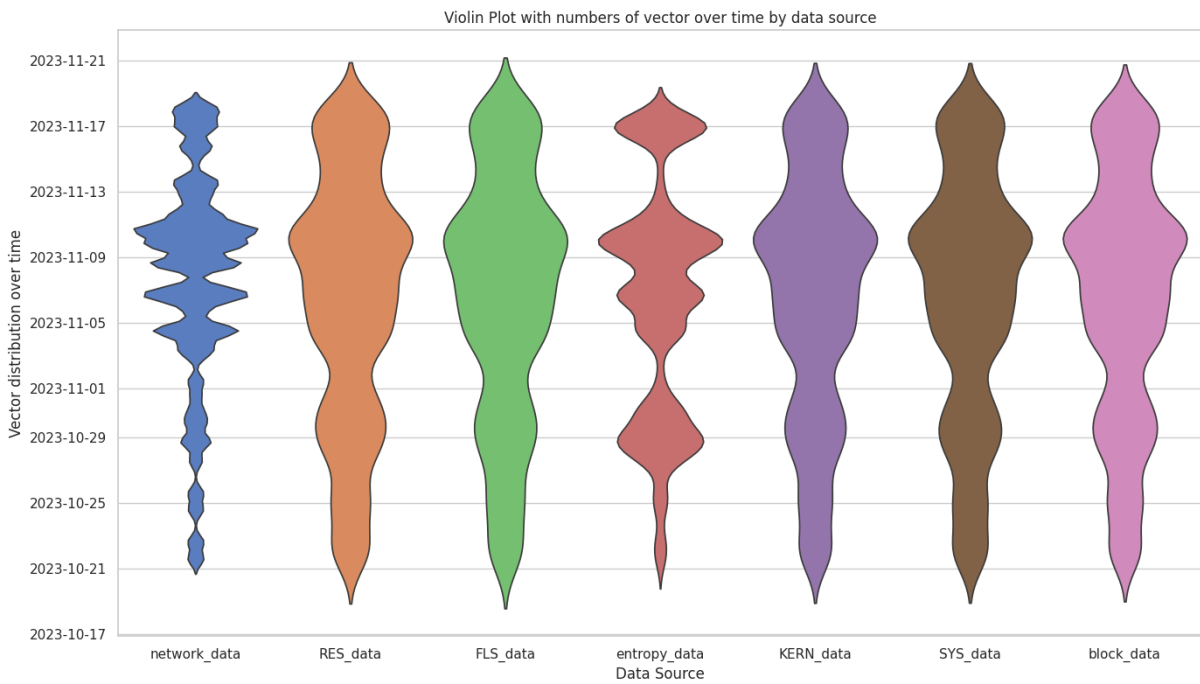


Figure A.3: Vector distribution over time by data source

Appendix B

Additional Feature Selection Figures

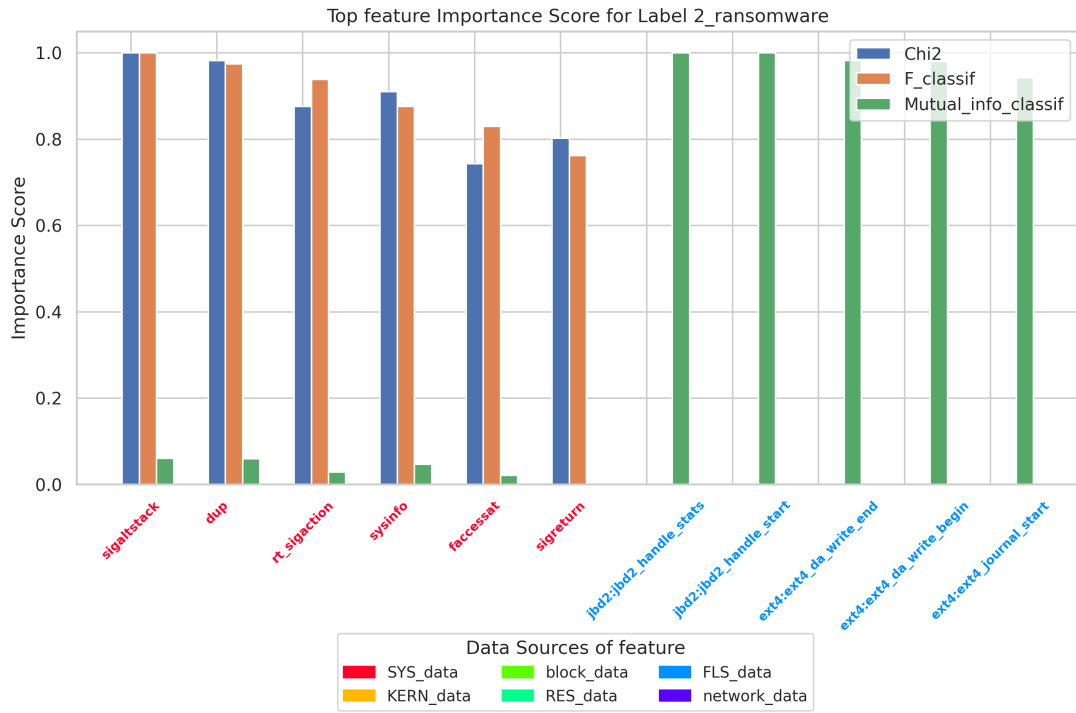


Figure B.1: Top 5 features for each statistical test are plotted for label 2: ransomware attack. The color of the feature title indicates the data source.

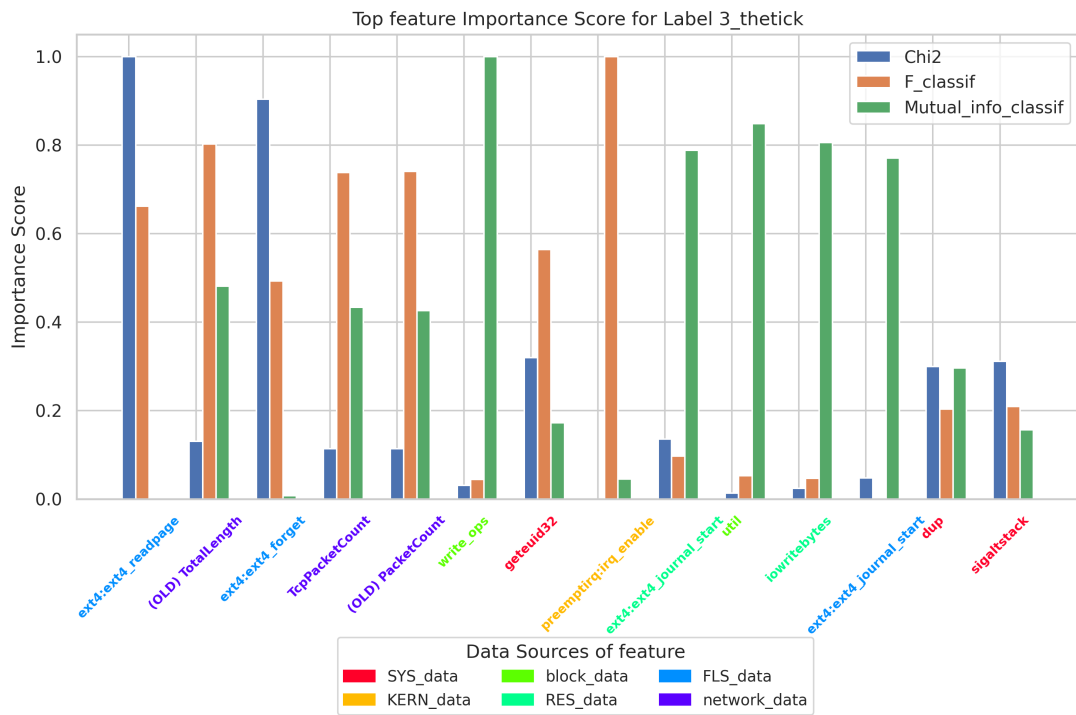


Figure B.2: Top 5 features for each statistical test are plotted for label 3: thetick attack. The color of the feature title indicates the data source.

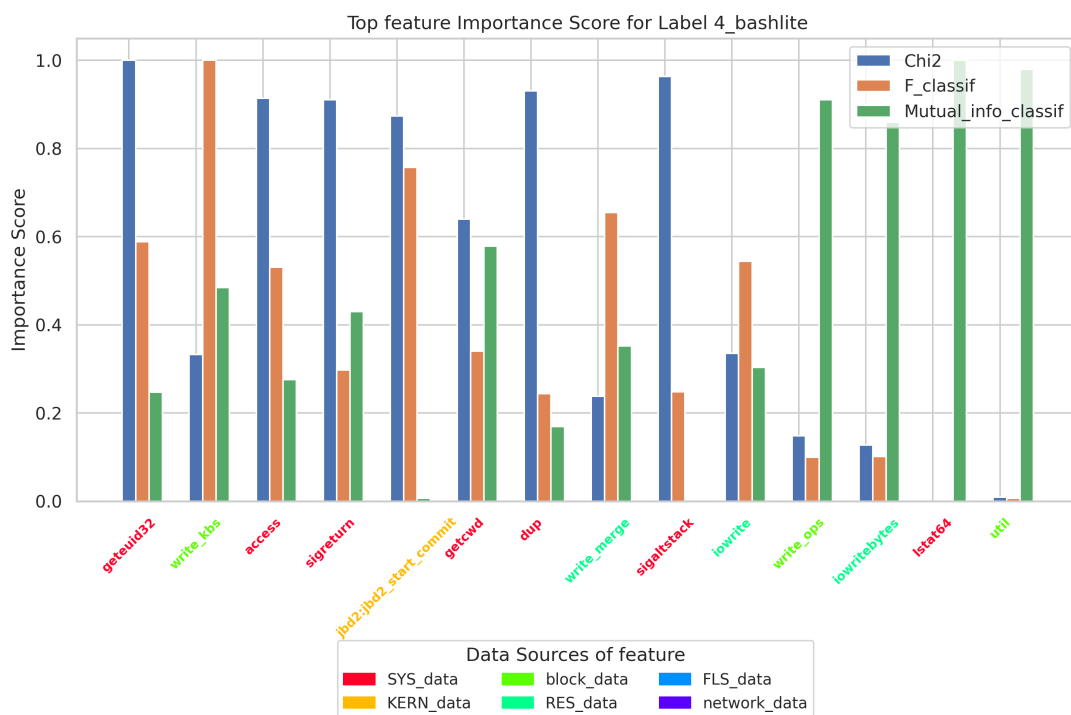


Figure B.3: Top 5 features for each statistical test are plotted for label 4: **bashlite** attack. The color of the feature title indicates the data source.

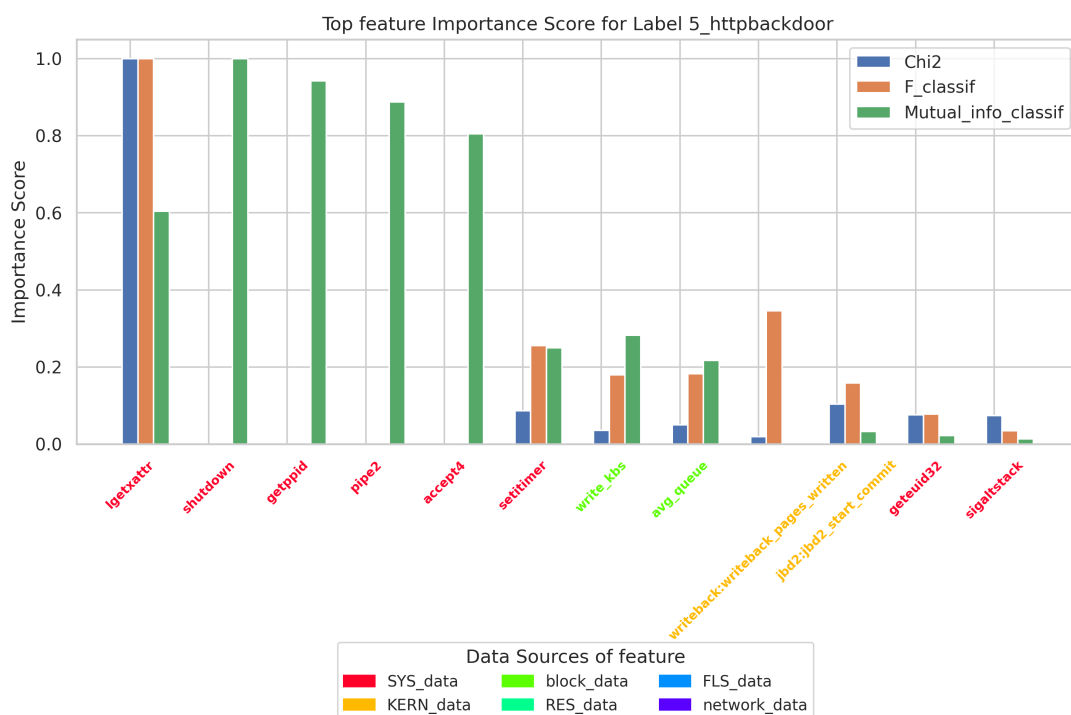


Figure B.4: Top 5 features for each statistical test are plotted for label 5: **httpbackdoor** attack. The color of the feature title indicates the data source.

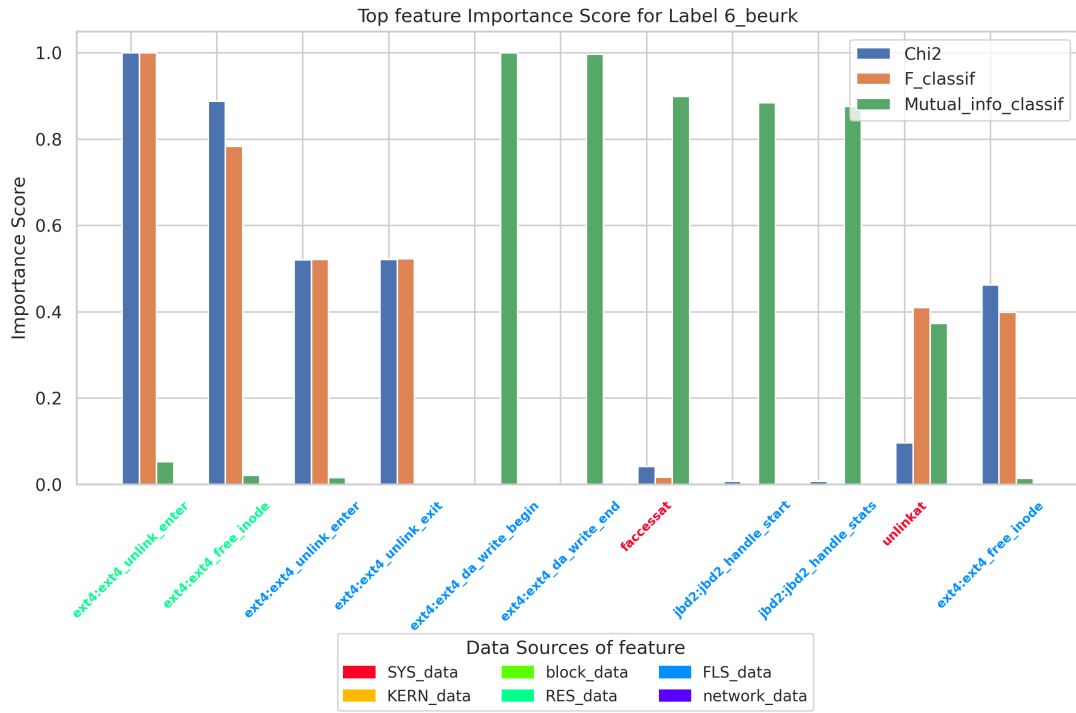


Figure B.5: Top 5 features for each statistical test are plotted for label 6: **beurk** attack. The color of the feature title indicates the data source.

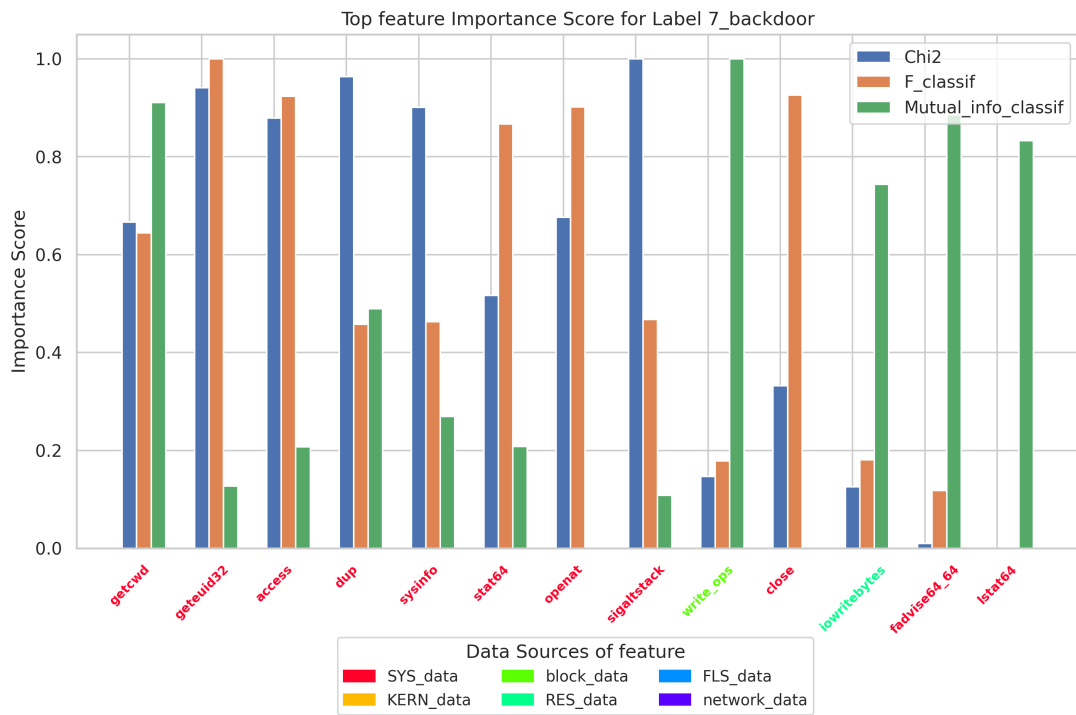


Figure B.6: Top 5 features for each statistical test are plotted for label 7: **backdoor** attack. The color of the feature title indicates the data source.

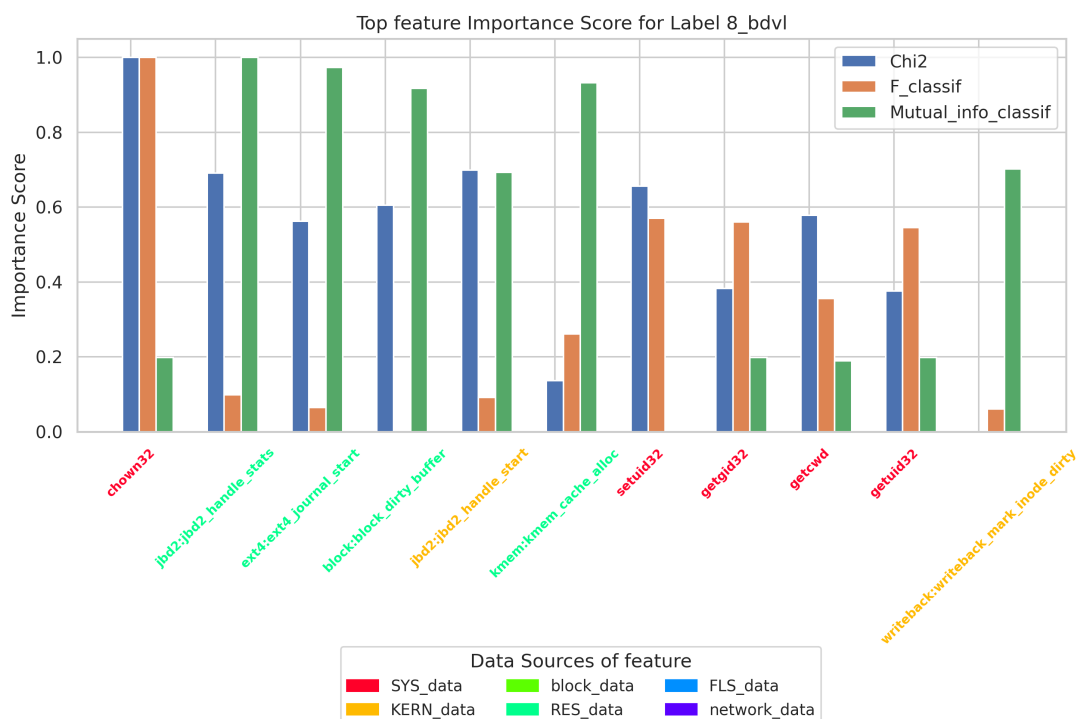


Figure B.7: Top 5 features for each statistical test are plotted for label 8: **bdvl** attack. The color of the feature title indicates the data source.

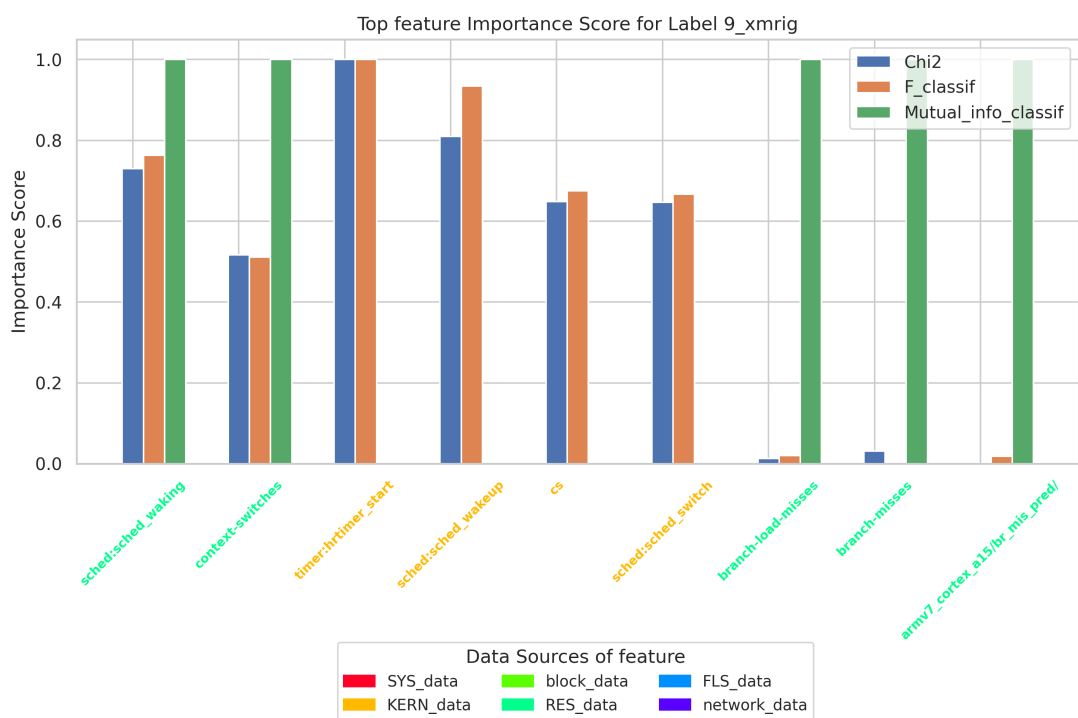


Figure B.8: Top 5 features for each statistical test are plotted for label 9: **xmrig coinminer** attack. The color of the feature title indicates the data source.

