3-22-2019

Evaluating Machine Learning Techniques for Smart Home Device Classification

Angelito E. Aragon Jr.

EVALUATING MACHINE LEARNING TECHNIQUES FOR SMART HOME DEVICE CLASSIFICATION

THESIS

Angelito E. Aragon Jr., Captain, USAF

AFIT-ENG-MS-19-M-006

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-19-M-006

EVALUATING MACHINE LEARNING TECHNIQUES FOR SMART HOME DEVICE CLASSIFICATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Angelito E. Aragon Jr., B.S.C.E.

Capt, USAF

March 2019

AFIT-ENG-MS-19-M-006

EVALUATING MACHINE LEARNING TECHNIQUES FOR SMART HOME DEVICE CLASSIFICATION

Angelito E. Aragon Jr., B.S.C.E.
Capt, USAF

Committee Membership:

Barry E. Mullins, Ph.D., P.E.
Chair

Brett J. Borghetti, Ph.D.
Member

Timothy H. Lacey, Ph.D., CISSP
Member

AFIT-ENG-MS-19-M-006

Abstract

Smart devices in the Internet of Things (IoT) have transformed the management of personal and industrial spaces. Leveraging inexpensive computing, smart devices enable remote sensing and automated control over a diverse range of processes. Even as IoT devices provide numerous benefits, it is vital that their emerging security implications are studied. IoT device design typically focuses on cost efficiency and time to market, leading to limited built-in encryption, questionable supply chains, and poor data security. In a 2017 report, the United States Government Accountability Office recommended that the Department of Defense investigate the risks IoT devices pose to operations security, information leakage, and endangerment of senior leaders [1].

Recent research has shown that it is possible to model a subject's pattern-of-life through data leakage from Bluetooth Low Energy (BLE) and Wi-Fi smart home devices [2]. A key step in establishing pattern-of-life is the identification of the device types within the smart home. Device type is defined as the functional purpose of the IoT device, e.g., camera, lock, and plug. This research hypothesizes that machine learning algorithms can be used to accurately perform classification of smart home devices.

To test this hypothesis, a Smart Home Environment (SHE) is built using a variety of commercially-available BLE and Wi-Fi devices. SHE produces actual smart device traffic that is used to create a dataset for machine learning classification. Six device types are included in SHE: door sensors, locks, and temperature sensors using BLE, and smart bulbs, cameras, and smart plugs using Wi-Fi. In addition, a device classification pipeline (DCP) is designed to collect and preprocess the wireless traffic, extract features, and produce tuned models for testing. *K*-nearest neighbors (KNN), linear discriminant analysis (LDA), and random forests (RF) classifiers are built and tuned for experimental testing.

During this experiment, the classifiers are tested on their ability to distinguish device types in a multiclass classification scheme. Classifier performance is evaluated using the Matthews correlation coefficient (MCC), mean recall, and mean precision metrics. Using all available features, the classifier with the best overall performance is the KNN classifier. The KNN classifier was able to identify BLE device types with an MCC of 0.55, a mean precision of 54%, and a mean recall of 64%, and Wi-Fi device types with an MCC of 0.71, a mean precision of 81%, and a mean recall of 81%. Experimental results provide support towards the hypothesis that machine learning can classify IoT device types to a high level of performance, but more work is necessary to build a more robust classifier.

Acknowledgements

To my family and friends: thank you for your unfailing support and encouragement throughout this journey. I would not be where I am today without you.

To my advisor, Dr. Mullins: thank you for your mentorship throughout my time here in AFIT. Your door was always open whenever I needed guidance.

To Dr. Borghetti, Dr. Lacey, and all AFIT faculty: thank you for sharing your time and knowledge with me. I am grateful that I had the opportunity to learn so much from your expertise and experience.

Bryan Aragon

Table of Contents

List of Figures

List of Tables

EVALUATING MACHINE LEARNING TECHNIQUES

FOR SMART HOME DEVICE CLASSIFICATION

I. Introduction

1.1 Background

Smart devices are increasingly being used in consumer and industrial applications. Once connected to the Internet, these smart devices allow for remote sensing and control of a wide variety of processes. The Internet of Things (IoT) is expected to have a network of over 31 billion devices by 2020 [3]. IoT devices have the potential to affect personal and commercial spaces, and therefore need to be studied for cybersecurity implications. IoT device design often focuses on minimizing power and cost [4]. Such design decisions can result in deficient security that cause information leakage. IoT devices regularly perform automatic functions upon a subject's arrival or departure. Traffic from these devices can be analyzed to figure out a subject's pattern-of-life [2]. By learning which type of devices are activating upon a subject's presence, a malicious actor can gain exploitable information. Therefore, it is important to study whether such IoT device classification is possible.

1.2 Problem Statement

Recent research has shown that it is possible to model a subject's pattern-of-life through data leakage from Bluetooth Low Energy (BLE) and Wi-Fi smart home devices [2]. BLE and Wi-Fi are two widely-used protocols in IoT devices that can leak sensitive information wirelessly, available for malicious attackers to collect and analyze without user awareness. A critical step in establishing pattern-of-life is the identification of the device types within the smart home. Device type is defined as the functional purpose of the IoT device, and extends across a broad spectrum including cameras, electrical plugs, light bulbs, door locks, temperature sensors, and motion sensors. Previous techniques in IoT device classification have been limited to manual packet analysis, a deliberate process that requires specific knowledge of target devices. This research seeks to leverage machine learning algorithms to produce a generalized and scalable method of IoT device classification. The problem statement this work answers is whether machine learning can be applied to successfully classify devices into their respective device types using collected wireless traffic.

1.3 Hypothesis and Research Goals

This work hypothesizes that if machine learning classifiers are trained using wireless traffic from a realistic smart home environment, then the classifiers can successfully identify the device type of IoT devices to a high degree of accuracy.

The goals that guide this research are:

1. Design and build a source of realistic smart home device traffic.
2. Develop procedures to collect and prepare the wireless traffic for machine learning classification.
3. Evaluate the performance of the linear discriminant analysis (LDA), *k*-nearest neighbors (KNN), and random forests (RF) machine learning classification algorithms in determining IoT device types.

4. Determine which features are most useful for classification purposes.
5. Assess the suitability of machine learning in IoT device type classification.

1.4 Approach

A smart home environment composed of commercially-available BLE and Wi-Fi devices is assembled to produce authentic wireless traffic. The wireless traffic is collected and preprocessed into a dataset suitable for machine learning. Classifiers are tuned and trained. The models are tested on a number of classification tasks to evaluate their performance. Results are synthesized to consider algorithm performance and device security implications. The machine learning approach uses a multiclass classification scheme, with three device types per wireless protocol used as response classes. *K*-nearest neighbors, random forests, and linear discriminant analysis are the classification algorithms used.

1.5 Assumptions/Limitations

The following assumptions and limitations are recognized throughout this experiment:

- The devices selected in the smart home environment are representative of an authentic smart home.
- All devices are compatible with an Apple iPhone.
- All algorithms are accurately implemented by third-party libraries.

1.6 Contributions

This research adds to the fields of IoT security and machine learning classification through two primary contributions:

1. **Smart Home Environment (SHE):** A smart home architecture using BLE and Wi-Fi smart devices is designed to provide realistic wireless traffic that can be used for analysis and classification.
2. **Device Classification Pipeline (DCP):** A system of machine learning techniques is applied to collect, process, and analyze the wireless traffic produced by the smart home devices.

1.7 Thesis Overview

This thesis is organized into six chapters. Chapter 2 presents an overview of relevant wireless protocols, machine learning techniques, classification algorithms, and other related research. Chapter 3 provides the design details of the SHE and DCP systems used to create, capture, prepare, and analyze the wireless traffic used in the experiment. Chapter 4 discusses the experiment methodology, while Chapter 5 presents the analysis of results. Lastly, Chapter 6 provides a summary of the work and considers possible avenues for future work in this research area.

II. Background and Related Research

2.1 Overview

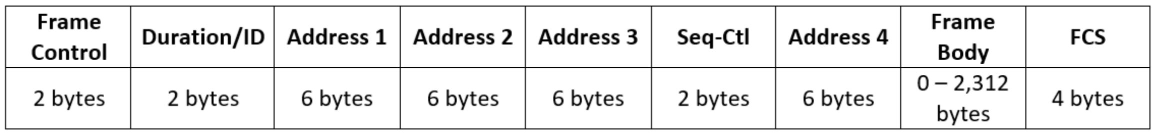
This chapter presents a technical review of the wireless protocols Wi-Fi and BLE in Sections 2.2 and 2.3 respectively to describe what features of their architecture and packet structure are applied in machine learning classification. Section 2.4 follows with a brief description of machine learning, and Section 2.5 provides a summary of traffic analysis research, the current state of IoT device classification, and a discussion of related research. Lastly, Section 2.6 offers a list of common terminology used throughout this research.

2.2 Wi-Fi

By far, the most commonly used technology for wireless local area networks (WLANs) is defined by the IEEE 802.11 standard, also known as Wi-Fi [1]. IEEE 802.11, hereafter referred to as 802.11, defines the medium access control (MAC) and Physical Layers (PHY). In wireless networks, a station (STA) is the addressable unit, and the basic service set (BSS) is the fundamental building block of a WLAN. The BSS is the effective area within which member STAs of the BSS can continue communication. In infrastructure mode, WLAN topology is centered on an access point (AP) that connects STAs from the WLAN to the wired network. A service set identifier (SSID) serves as the primary name associated with a WLAN and is typically used by STAs to find WLANs.

Association is the process through which a STA connects to an AP. 802.11 expects the AP to periodically send out beacon frames, containing the AP's SSID and media access control (MAC) address. The STA seeks out these beacon frames by continuously scanning the wireless channels defined in 802.11. Once an AP has been selected, the STA sends an association request frame to the AP, and the AP responds with an association response frame. After this process, the AP typically assigns an IP address to the STA through a Dynamic Host Configuration Protocol (DHCP) exchange. Once completed, the STA has joined the AP's subnet and is viewed as simply another device in that subnet.

The general frame used to transmit data in 802.11 is illustrated in Figure 1. The frame consists of various fields: frame control, duration/identification, address fields, sequence control, frame body, and frame check sequence (FCS). The frame control field contains the protocol version, frame type and subtype, and other control information. The duration/identification field specifies the transmission time required for the frame. The four address fields include the destination address, source address, receiver address, and, occasionally, the transmitter address. The sequence control field helps identify duplicate frames. The frame body, also known as the Data field, moves the higher-layer payload between stations. The FCS is a checksum appended to the frame to detect corruption. If the receiver calculates a different FCS than the FCS included in the frame, the frame is deemed corrupted and is discarded.



Frame Control	Duration/ID	Address 1	Address 2	Address 3	Seq-Ctl	Address 4	Frame Body	FCS
2 bytes	2 bytes	6 bytes	6 bytes	6 bytes	2 bytes	6 bytes	0 – 2,312 bytes	4 bytes

Figure 1. Wi-Fi Frame Fields [5]

2.3 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a separate technology from classic Bluetooth, with different design goals [6]. BLE, sometimes referred to as Bluetooth 4.0, was first introduced in 2010 by the Bluetooth Core Specification 4.0 [7]. While classic Bluetooth focuses on high data rates, BLE has been optimized for ultra-low power applications. Bluetooth Low Energy is not trying to improve on Bluetooth classic; instead, it targets new applications that have not previously used open wireless standards. These applications are those that require devices to send minimal octets of data from once a second to once every few days. By design, BLE is intended to minimize not only overall activity, but even the time required to do anything useful. If a device is operating, even if it is nothing more than checking whether it needs to send or receive something, it is using energy.

Certain key elements support low cost, including its industrial, scientific, and medical (ISM) band, intellectual property license and low power. The 2.4 GHz ISM band may have poor propagation, but is available worldwide with no license requirements. The Bluetooth Special Interest Group (SIG) only requires a very low cost intellectual property license. Finally, the best way to design a low-cost device is to reduce required materials such as batteries. BLE was designed to work with the smallest, cheapest, and most readily available battery option – button-cell batteries.

The BLE architecture is split into three parts: controller, host, and applications, as shown in Figure 2. The controller is a physical device that transmits and receives radio signals, and can convert these signals into packets with information. Within the controller are the physical and link layers, as well as the lower half of the Host Controller Interface. The controller can be identified as the Bluetooth chip or radio. The controller communicates to other devices using an antenna, and to the host using the Host Controller Interface.

The host is a software stack that directs how multiple devices communicate with one another, typically managing several services at the same time. The host controls the Logical Link Control and Adaptation Protocol (L2CAP), the Security Manager Protocol, Attribute Protocol, Generic Attribute Profile (GATT), and Generic Access Profile (GAP). The L2CAP handles the passing of data between host and the controller through channels. BLE uses three fixed channels: one each for connection management data, the Security Manager, and the Attribute Protocol. The Security Manager Protocol handles device pairing and key distribution. The Attribute Protocol defines the rules for accessing data by another device through the use of *attributes*. The GATT resides above the Attributes Protocol and defines the types of attributes and how they can be used. More detail on how attributes work is included in Section 2.3.5. Lastly, the GAP controls how devices discover, connect, and provide information to users in the application layer. It also outlines the procedures needed to discover, connect, and pair with other devices by controlling the link layer states. Finally, applications use the BLE architecture to provide various functions to users.

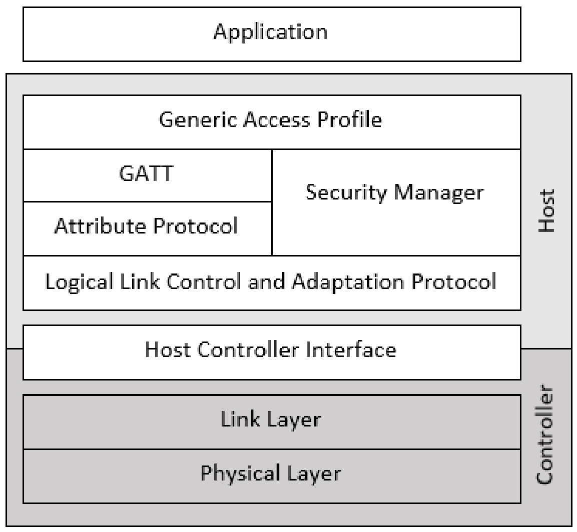


Figure 2. BLE Architecture [6]

The physical layer of the controller transmits and receives bits using the 2.4 GHz band. The frequency of the radio waves use a modulation scheme called Gaussian Frequency Shift Keying (GFSK) that shifts the frequency slight up and down over a Gaussian filter. Compared to classic Bluetooth's 79 1-MHz channels, BLE is split into 40 separate channels, with a 2 MHz separation between one another, as shown in Figure 3. Figure 3 also shows the advertising channels, indicated by the darkened channels. When transmitting data, BLE transmits at the rate of 1 million bits per second (Mbps), with a maximum transmit power of 10 mW.

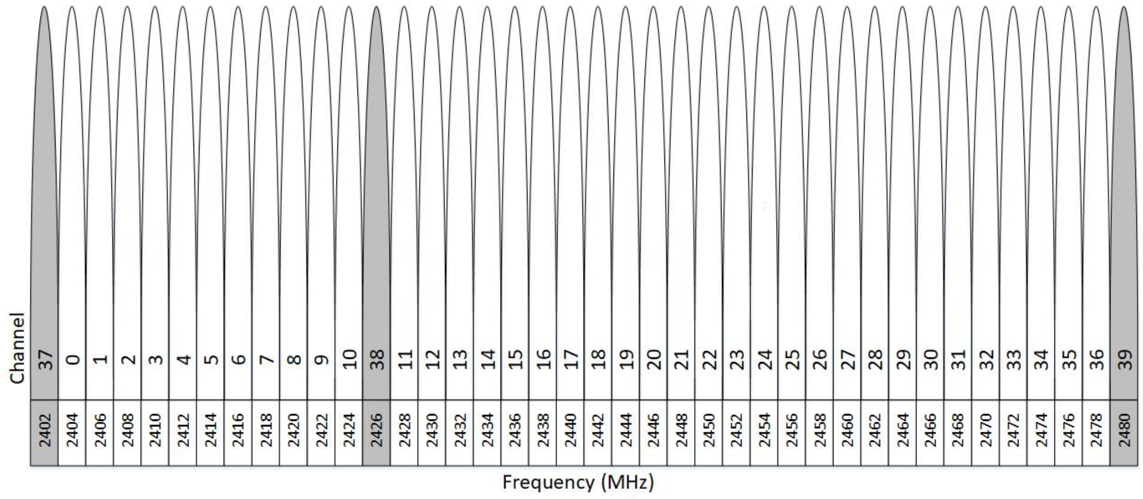


Figure 3. BLE Channel Map [6]

2.3.1 BLE Link-Layer States

The link layer describes packet details, advertising, and data channels. It also describes how device discovery, data broadcasting, and connections operate. As shown in Figure 4, the link layer defines five states: Standby, Advertising, Scanning, Initiating, and Connected.

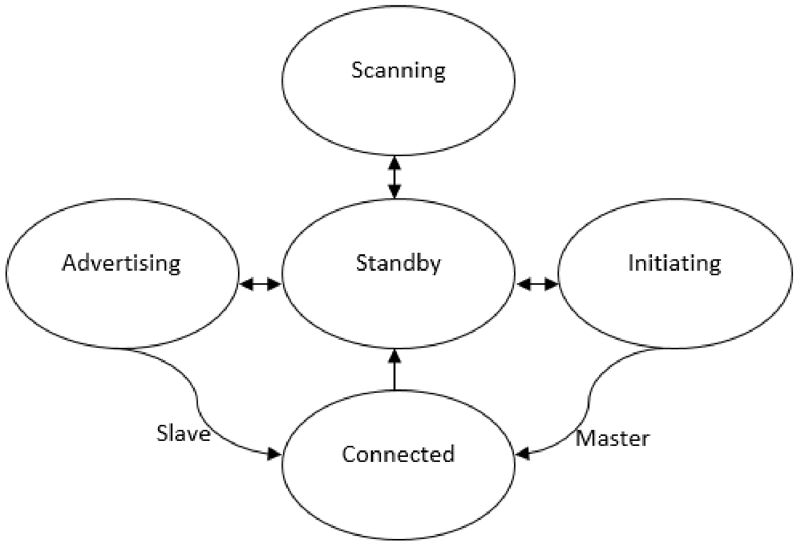


Figure 4. BLE States [6]

Upon powering on, devices start in the standby state and remain there until the host layers instruct them otherwise. Devices in the standby state can move into all other states. Once fully powered, the advertising state can be initiated by the application through the GAP. In the advertising state, the link layer can transmit advertising packets or respond to scan requests. Devices that want to be discoverable or connectable must be in the advertising state. Devices in the advertising state can only move to the connected or standby state.

The scanning state allows a device to receive advertising channel packets from other devices in the local area. There are two types of scanning: passive scanning and active scanning. Passive scanning only receives advertising packets; the device never transmits anything. In active scanning, the device additionally sends scan requests to all advertising devices. The advertising device then replies with a scan response. Both the scan requests and response packets are sent on the advertising channel.

To initiate a connection between devices, the link layer must go through the initiating state. In this state, the initiating device listens for advertising packets from the device with which it is trying to connect. Once an advertising packet is received, the link layer sends a connect request to the advertising device and moves into the connected state.

The last state of the link layer is the connected state. The connected state can only be entered through the advertising or initiating states. It is only in the connected state that data channel packets are sent and received. There are two substates: master or slave. Only the device that initiates the connection can become the master. A master device must regularly send packets to the slave device. The slave substate can only be entered from the advertising state. The device that becomes the slave must have been advertising to another device. A slave device can only transmit in response to the master device. Devices cannot be both master and slave simultaneously, nor can a device be a slave of two masters at the same time.

2.3.2 BLE Packet Structure

The packet is the standard data block of the link layer. There are two types of packets: advertising and data packets. Advertising packets are used to find and connect to other devices, while data packets are used once a connection is established. The packet type is determined by the channel on which the packet is transmitted. If a packet is transmitted on one of the three advertising channels, then it is an advertising packet; if it is transmitted on any of the 37 data channels, it is a data packet.

Link-layer packets follow the structure as displayed in Figure 5. These are divided into the preamble, access address, header, length, data and cyclic redundancy check (CRC) fields.

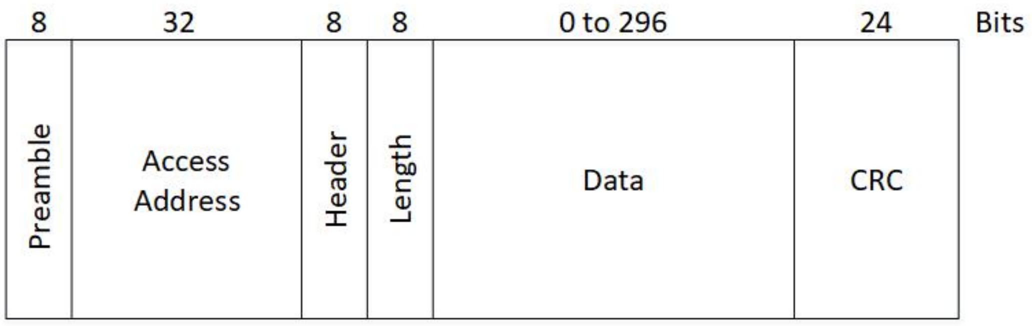


Figure 5. BLE Link-Layer Packet Structure [6]

The preamble, or the first 8 bits of a packet, is always either a 01010101 or 10101010 sequence, randomly selected. These simple sequences allow the radio to adjust gain and determine the frequencies used for zero and one bits. The access address is the next 32 bits and can be one of two types based on packet type: advertising access address or data access address. Advertising access addresses are set at a fixed value (0x8E89BED6) to help standardize the advertising process. Data channels use a different random access address on each and every connection, which is used when data must be reliably delivered to another device.

The header field varies based on the packet type. For advertising packets, the header contains the advertising protocol data unit (PDU) type. Table 1 provides a summary for each PDU type. ADV_IND indicates that the device is advertising as connectable (available to create a connection) and undirected (not looking to connect to a specific device); this is the advertising packet type most commonly used. ADV_DIRECT_IND indicates that the device is connected and directed (looking for a specific device with which to connect). ADV_NONCONN_IND indicates that the device is nonconnectable (refuses to connect) and undirected; this is used by devices seeking to only broadcast data. ADV_SCAN_IND, SCAN_REQ, and SCAN_RSP are used during active scanning. ADV_SCAN_IND indicates that the advertising device is open to active scanning, SCAN_REQ is a request made by the initiating device to receive a scan response, and SCAN_RSP is the scan response itself. Lastly, the CONNECT_REQ header type is sent by an initiating device to an advertising device when the initiating device wants to create a connection. CONNECT_REQ packets contain information needed to establish a connection.

Table 1. Advertising PDU Types

	PDU Type	**Purpose**
1	ADV_IND	General advertising indication
2	ADV_DIRECT_IND	Direct connection indication
3	ADV_NONCONN_IND	Nonconnectable advertising indication
4	ADV_SCAN_IND	Scannable indication
5	SCAN_REQ	Active scanning request
6	SCAN_RSP	Active scanning response
7	CONNECT_REQ	Connection request

Data packets have headers containing the logical link identifier (LLID), sequence number (SN), next expected sequence number (NESN), and more data, as shown in Figure 6. The LLID is used by the link layer to manage the channel for this connection. The one-bit sequence number for each new data packet toggles from the previous data packet's sequence number, with the first data packet in a connection having a sequence number of zero. The SN allows the receiving device to determine whether the received packet is a retransmission of a previous packet or a new packet. The NESN allows for acknowledgement of data packets. The last bit in the data packet header is the more data bit, where 1 signals that there is more data to transmit, and 0 signals the end of the data transmission.

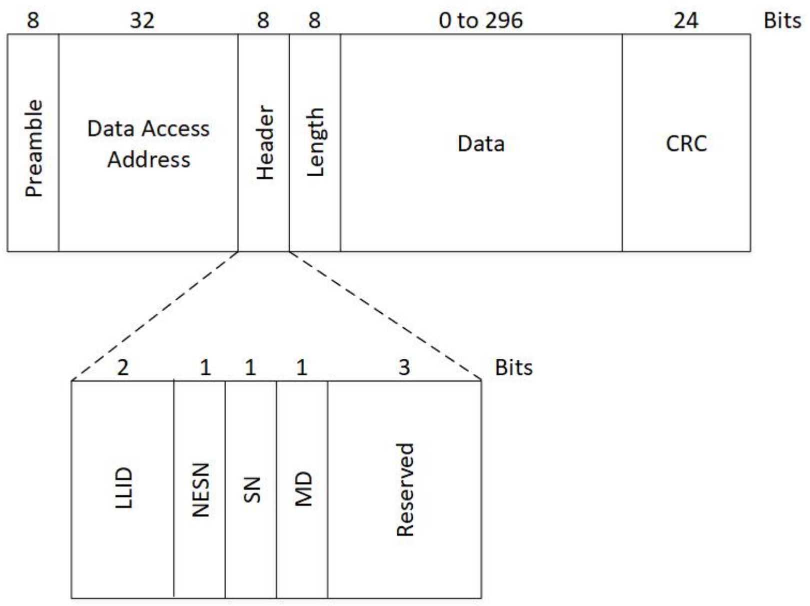


Figure 6. BLE Data Packet

The length field reports the size of the packet, with a range of valid values from 6 to 37 bytes for advertising packets, and 0 to 31 bytes for data packets. The payload is the actual data that is being transmitted for use by the application. The final part of the packet is a 3-byte CRC. The CRC is calcuated using the header, length and payload fields, and serves to detect accidental changes to raw data.

2.3.3 Creating Connections

A connection is required to reliably allow for two-way data transfer. Figure 7 shows how a connection is typically created. The first step is for one device to advertise using an advertising packet (commonly with ADV_IND) and for another device to initiate a connection to the advertising device with a CONNECT_REQ packet. Using the information in the CONNECT_REQ packet, a connection is created between the two devices, with the initiating device now the master device and the advertising device as the slave device.

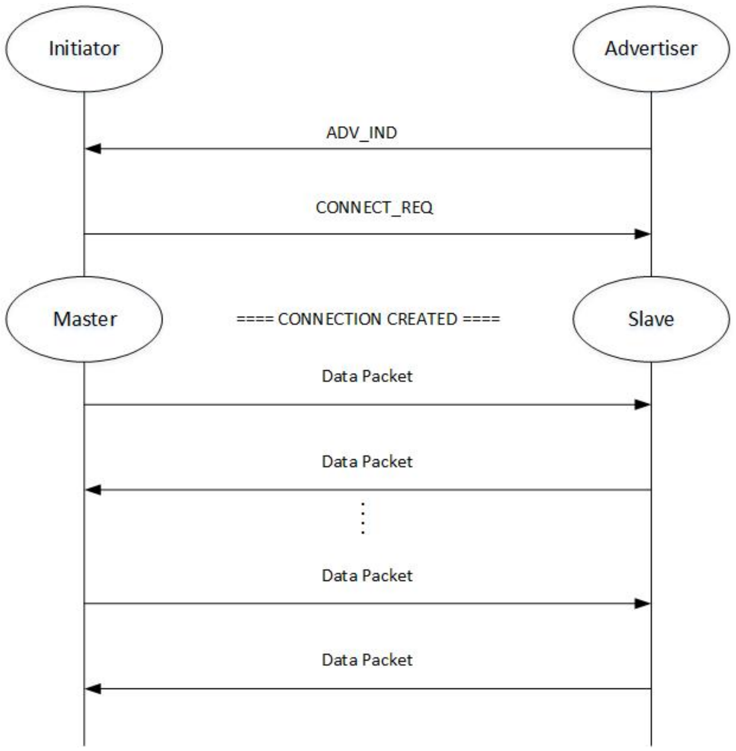


Figure 7. BLE Connection and Data Sending Diagram

All necessary information is contained within this CONNECT_REQ packet, including access address, connection interval, and channel map. The access address is randomly determined by the master. If a master has multiple slaves, it chooses a different access address for each slave. When in a connection, the master must transmit a packet to the slave once every connection event. The connection interval determines how frequently this happens; the connection interval can be any period between 7.5 milliseconds to 4 seconds. Lastly, the channel map is a bit mask of the data channels the connection uses, where if the bit is set to one, then the channel is deemed a good channel and can be used for data traffic, and if the bit is set to zero, the channel is deemed a bad channel, and is never used for data traffic.

The Generic Access Profile (GAP) defines the discovery and connection process between devices. The GAP provides two types of discoverability: limited and general. Limited-discoverable mode is used by devices that have just been made discoverable, and are meant to stand out from general-discoverable devices. As such, devices are not allowed to remain in the limited-discoverable mode for more than 30 seconds. The general-discoverable mode is used by devices that are discoverable but have been inactive for a period of time. This becomes the default mode for devices once they exceed the 30 seconds allowed for limited-discoverable mode.

2.3.4 Sending Data

Once in a connection, devices can send data to each other using data packets. Data packets have four fields in their header: logical link identifier, sequence number, next expected sequence number and more data. The logical link identifer (LLID) determines what kind of data the packet contains. The LLID can indicate that the packet is a link layer control packet, which is used by the link layer to manage connections. Otherwise, it is a data packet intended for the host, and can either be a start packet or continuation packet. Start packets signal the beginning of a series of data packets, and continuation packets make up the rest of the transmission. Interestingly, because the link layer does not need to know the entire length of the data, continuation packets can be continuously sent. Data packets have a single bit for the sequence number, beginning with zero for the first data packet. It then alternates between one and zero for each new data packet. To acknowledge a data packet, the next expected sequence number (NESN) is used. If the data packet received by a device has a sequence number of one, then the NESN is zero; otherwise, the data packet would be retransmitted. Lastly, the MD bit indicates that the transmitting device has more data ready to send. If one, then the receiving device maintains the connection. If zero, then the two devices can close the connection to save power.

Figure 8 provides an example of how connection events occur between two connected devices. A connection event is the start of a set of data packets sent from the master to the slave and back again. Connection events are always initiated by the master device. The master device initiates the transmission by sending a data packet with SN zero, NESN zero, and MD one. The slave device receives this packet and attempts to send its own packet, with SN zero, NESN one (acknowledging that the previous packet) and MD one. However this packet was not properly received by the master device. Without an acknowledgment from the slave device, the master device retransmits its first packet. The slave device detects that retransmission of the previous packet is required, and does so. This time, the packet is properly received. The master device no longer needs more data from the slave device and sends a packet with the MD bit zero. The slave device acknowledges this by sending its own packet with MD zero, and the connection event between the two devices end. A second connection event is initiated by the master, but the MD bit is set to zero. This type of connection event is typically performed to check on the slave's status, serving as a "ping". The slave receives the packet, and seeing that MD is zero, acknowledges the previous packet and ends the connection event.

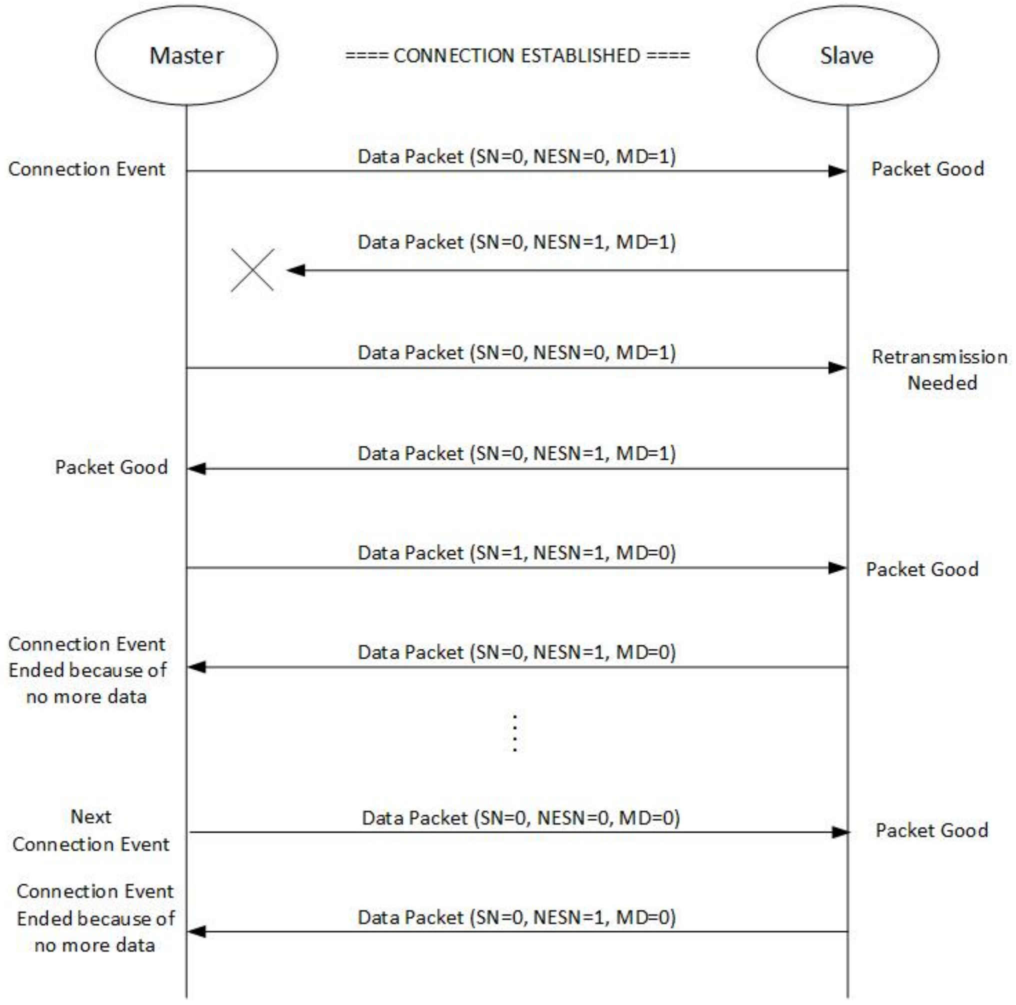

Master
==== CONNECTION ESTABLISHED ====
Slave
Connection Event
Data Packet (SN=0, NESN=0, MD=1)
Packet Good
Data Packet (SN=0, NESN=1, MD=1)
Data Packet (SN=0, NESN=0, MD=1)
Retransmission Needed
Packet Good
Data Packet (SN=0, NESN=1, MD=1)
Data Packet (SN=1, NESN=1, MD=0)
Packet Good
Connection Event Ended because of no more data
Data Packet (SN=0, NESN=1, MD=0)
Next Connection Event
Data Packet (SN=0, NESN=0, MD=0)
Packet Good
Connection Event Ended because of no more data
Data Packet (SN=0, NESN=1, MD=0)

Figure 8. Data Transmission [6]

2.3.5 Attributes

The Attribute Protocol, shown in Figure 2, is central to understanding Bluetooth Low Energy. BLE is designed as a client-server architecture, where a server is a device that has data, and a client is any device that is using data from another device. Figure 9 shows how the client-server architecture works. In practice, the master device acts as the client requesting data from its slave devices who act as servers.

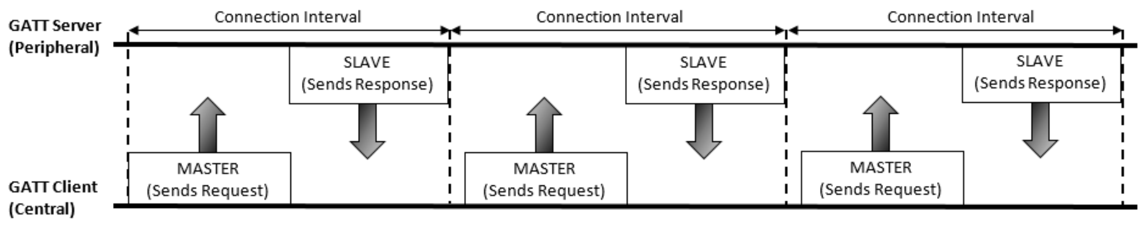


Figure 9. GATT Client-Server Interaction [8]

Attributes are the fundamental structure through which BLE achieves the client-server architecture. The Generic Attribute Profile (GAP) is a set of rules that define how to present, group, and transfer data using BLE. The GAP defines attributes as a piece of labeled, addressable data, and each attribute has three parts: a handle, a type, and a value. The attribute handle is the attribute's 16-bit address. The attribute type is comparable to a data type in programming languages, and is used to identify the nature of the attribute's information (e.g., temperature, pressure, time, etc.). Lastly, the attribute value is the actual value and has a size between 0 to 512 bytes. Attributes are stored in an attribute database, which is in turn contained within an attribute server. Clients communicate with the attribute server to obtain desired information. There can only be one attribute server per device, and every device must have both an attribute server and an attribute database.

Permissions must be set for every attribute in an attribute database, and these permissions come in three categories: access, authentication, and authorization. Access permissions must be set to readable, writable, or readable-writable. Authentication and authorization permissions are not required and can be left open. The difference between the two permission types is that authentication occurs at the client level, while authorization occurs at the server level. It is important to note that these permissions only relate to the attribute value; any device has permission to view the attribute handles and types on a given device.

The Attribute Protocol is the protocol through which clients find and access attributes on an attribute server. It is a simple protocol with only six basic operations: Request, Response, Command, Indication, Confirmation, and Notification.

A Request is sent by the client when the client wants the server to do an action and send back a Response. A client can only send one Request at a time, and must wait for a Response before sending another Request. A Command is similar to a Request, except no Response is needed. Indications are used by the server to inform a client about an update on a given attribute's value, and require a Confirmation from the client. Notifications are similar to Indications, except they need no Confirmation. Since Commands and Notifications do not require Responses nor Confirmations, they can be sent without any restrictions. If the receiving device cannot handle all the messages, the messages may be dropped. Therefore, Commands and Notifications are unreliable, while Requests and Indications are considered reliable.

Protocol messages are combinations of these basic operations used to perform common tasks using the Attribute protocol. Their role is comparable to library functions in programming languages. Most messages consist of both a Request and a Response operation. For example, the Read Request message uses a Request operation that has a handle of a desired attribute, and the Response returns the attribute value. Protocol messages have a wide range of functionality that enable efficient reading, writing, error handling, and notifications.

For connected devices, the Generic Attribute profile (GATT) defines two basic forms of grouping: characteristics and services. Figure 10 shows how the GATT structure organizes characteristics and services. Characteristics are defined attribute types that can only contain certain logical values. The BLE Specification provides over 200 predefined characteristics such as Alert Status, Language, Battery Level, and Time Zone that can only take on specific values based on their characteristic definition. A service is a collection of characteristics and relationships with other services to perform a given function [9]. Sometimes refered to as profiles, services expose certain device information and functionality in a standardized manner.

Predefined services include the Battery, Environmental Sensing, and Heart Rate services [7]. For example, consider a personal fitness monitoring device that uses the Battery, Environmental Sensing, and Heart Rate services. A user may connect the fitness monitor to a smartphone through BLE. Through the Battery service, the user can monitor the battery life of the device, ensuring that the device does not run out of power during workout sessions. Through the Environmental Sensing service, the user can monitor measurement data from the device's various sensors, such as air temperature, humidity and elevation. Finally, through the Heart Rate service, the user can track one's heart rate throughout the workout session. While predefined services accommodate common needs, custom services can also be created to suit developers' needs.

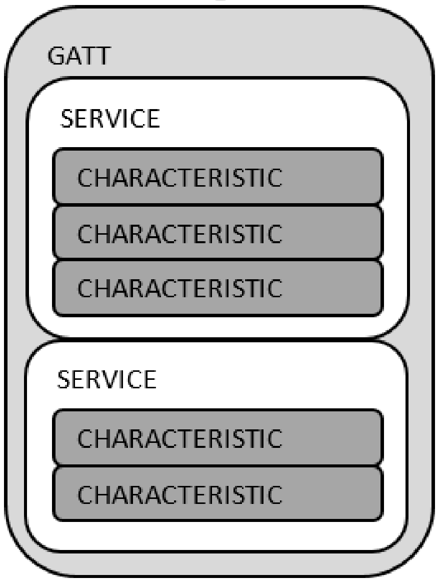


Figure 10. GATT Structure [8]

2.3.6 Security

The Security Manager (SM), shown in Figure 2, serves two fundamental functions: device pairing and message authentication. These two functions enable the rest of BLE's security features. Pairing allows two unfamiliar devices to authenticate each other's identity in preparation for activity that requires security. During pairing, each device first determines each other's input and output capabilities (e.g., no input-no output, display only, display yes/no, keyboard only, keyboard display) to determine what level of authentication is possible. For example, if two devices are both display only, they would not be able to authenticate via passkey entry, and the SM would default to simply letting the devices pair automatically without authentication. After determining input and outputs, the SM then proceeds to authenticate each other using a randomly-generated key, if possible. This is often implemented by the user typing in a six-digit key. Lastly, keys are distributed between the devices. Message authentication uses the CMAC (Cipher-based Message

Authentication Code) algorithm, with the keys distributed using pairing. An additional SignCounter value is used to prevent replay attacks.

2.4 Machine Learning

Coined by Arthur Samuel, the term machine learning (ML) refers to a field of computer science that applies statistical techniques to give computers the capability to learn from data without explicit programming [10]. ML plays a significant role in the fields of statistics, data mining, and artificial intelligence. Machine learning is increasingly being applied today to tasks too complex for traditional approaches or have no known algorithm [11].

To illustrate, consider the task of distinguishing between different types of fruit. Traditional approaches would need to first study the problem (for example, what is the difference between an apple and an avocado) then write rules to solve the problem (apples are red, avocados are green). However, this problem is more complex (some apples are also green). Therefore, additional rules must be included to further refine the solution to the program (apples have smooth skin, avocados have pebbled skin). But as more test cases and situations are added, this set of rules grows significantly (watermelons are green but have smooth skin), making the maintenance of these rules very difficult for a human programmer.

A typical ML scenario seeks to predict an outcome measurement, usually categorical (e.g., type of fruit) or quantitative (e.g., future house prices), using a set of *features* (e.g., fruit color, house location) from a dataset [12]. A set of *training data* is used to observe the outcome and feature measurements. With this data, a prediction *model* is built to predict the outcome for new cases, or *observations*.

Given a sufficiently large dataset, machine learning excels in the type of problem presented earlier. ML applies statistical techniques to reveal patterns within data. By analyzing the data, an ML approach can develop a model using the patterns in the data, and then produce a solution. Furthermore, the solution can reveal certain insights about the data that may have been missed.

There are two broad types of machine learning: supervised learning and unsupervised learning. Supervised learning requires the presence of the outcome measurement, or *labels*, to direct the learning process. A typical supervised learning task is classification, the task of assigning to which set of categories a given observation belongs. In the fruit classification problem, the fruit type would be an example of a label, and the task is assigning a fruit type to a given fruit.

As mentioned, classification is considered supervised learning because it requires the presence of labels. Another supervised learning task is regression. Regression is the task of predicting a numerical value for a given observation. An example is predicting future house prices, given the features of house location and age. Because of the use of labels, supervised learning methods can be evaluated on their performance. Chapter 3 provides the details on performance evaluation.

Unsupervised learning uses unlabeled data. Unsupervised learning aims to infer the structure of data. A common task for unsupervised learning is clustering. The goal of clustering is to detect groups of similar observations within the data. These groups may have not been clearly evident, and the detected groups may be used in applications such as pattern recognition, compression, and graphics. An example of a clustering problem would be when a website like Amazon or Netflix provides recommendations based on users' browsing history [13]. By clustering items that are similar to those users have previously looked at, online retailers or streaming services can suggest certain products or movies.

2.4.1 Classification Algorithms

The algorithms used in this experiment are among the most commonly used machine learning classification algorithms. All three algorithms are identified as supervised learning algorithms. Supervised learning requires the outcome measurement to direct the learning process. In classification, the objective is to assign a given observation to a particular class or label using a set of inputs or predictors. Because of their use of outcome measurements, supervised learning algorithms may be evaluated on their performance.

2.4.1.1 *K*-Nearest Neighbors

K-Nearest Neighbors (KNN) classifies an observation by finding the observation's k nearest neighbors, where k is an integer, and classifying the observation to the class with the highest estimated probability [14]. A commonly used distance metric is Euclidean distance, however other distance metrics can be used, such as Manhattan distance or Chebyshev distance. The hyperparameter k determines the number of neighbors KNN considers in its classification of an observation. Supposing $k = 3$, KNN considers the three closest neighbors of x_0. The observation is assigned the majority class of these k nearest neighbors. KNN does not make assumptions about how the data is distributed [15].

While this approach works well for data with unknown distributions, it leads to a higher susceptibility to local anomalies within the data. Additionally, if there are many dimensions in the data, several inputs may be “nearest” to the observation, leading to reduced effectiveness.

2.4.1.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a linear transformation technique first proposed by Ronald Fisher in 1936. LDA models the distribution of the predictors independently in each of the response classes, then applies Bayes’ theorem to find an estimate of the posterior probability. A Gaussian or normal distribution is typically used to model the distribution of the predictors. LDA assigns the observation to the response class with the highest probability. When the assumption about the predictors’ distribution does not hold, performance is reduced.

2.4.1.3 Random Forests

A random forest is an ensemble of decision tree-based algorithms [14]. Decision trees divide the feature space into different regions that can then be used to classify observations. By using a multitude (or ensemble) of decision trees, random forests assign an observation to the most commonly occurring class in the region to which it belongs. As with a decision tree, there is a danger of overfitting when applying random forests.

2.5 Related Work

Wired network traffic analysis has been extensively studied. Preliminary methods targeted port number and payload content analysis; these resulted in considerable success at the time [16]. However, these approaches have challenges that limit their effectiveness.

Port number analysis is accurate only if networks adhere to port standards, and payload analysis cannot cope with encrypted transmissions.

Transport-layer analysis has been applied to address the limitations of the previous methods [17]. In peer-to-peer (P2P) networks where arbitrary ports are most commonly used, transport-layer analysis allows the profiling of IP connection patterns. The observation of source-destination IP pairs and IP address-port pairs offered a method of studying P2P traffic without any examination of user payload. However, direct transport-layer analysis can be time-consuming, and analytical scripts are restricted by programmer knowledge.

Machine learning has emerged as a promising technique for traffic analysis [18]. Studies can be broadly categorized into unsupervised and supervised approaches. One of the earliest studies using unsupervised learning, McGregor et al. applied clustering techniques to group wired traffic flow between six common network protocols and found that the data rate was a key feature in this effort [19]. Bernaille et al. used a variation of the *k*-Means algorithm to classify Transmission Control Protocol (TCP) traffic to the application type (e.g., file transfer protocol (FTP), hypertext transfer protocol (HTTP), secure shell (SSH), etc.) using the packets at the start of traffic flow [20]. This approach focused on the packets used during the TCP handshake and was able to classify traffic flow with over 80% accuracy. However, this work assumed that the handshake can always be captured at the onset of traffic flow, which is not always achievable. It was with Erman et al. that web traffic was analyzed using the *k*-Means approach [21]. Instead of full bi-directional traffic flow, the work focused on uni-directional flows between server-to-client and client-to-server. Their results showed that server-to-client datasets produced the highest accuracy (95%), and that flow duration, number of bytes and number of packets were the features that provided the most value in classifying packets.

Supervised learning introduced algorithms such as the *k*-Nearest Neighbors (kNN), linear discriminant analysis (LDA), and quadratic discriminant analysis (QDA) to traffic analysis. A study by Roughan et al. used these techniques to classify different network applications to specific traffic classes [22]. The considered features were analyzed at the packet level, flow level, and connection level. Packet level features were derived from individual packets such as the size of the packet and the time the packet was sent. Flow level features were derived from sequences of packets that shared common field values, such as source IP address, destination IP address, and protocol type. Connection level features were derived from transport-layer protocol information such as those found in TCP connections. Of these features, average packet length and flow duration were found to be the most valuable. Moore and Zuev were able to further improve on traffic classification accuracy by using Naïve Bayes [23]. The dataset was manually classified beforehand, and over 240 features were used to train the classifier. Using Naïve Bayes alone, the study achieved approximately 65% accuracy. The classifier was then refined to reduce the number of features, and accuracy was improved to over 95%. Auld et al. extended this work by applying a Bayesian neural network (NN), further improving classification accuracy [24]. The training NN achieved a classification accuracy up to 99% for data trained and tested on the same day, and 95% for data trained and tested with an eight-month gap. The features were ranked for value, and the top three features were found to be: count of TCP PUSH packets, total number of bytes in the initial window (TCP handshake) from client to server, and total number of bytes in the initial window (TCP handshake) from server to client.

Wireless traffic adds a layer of complexity in traffic analysis. The most ubiquitous wireless network standard, IEEE 802.11 (commonly known as Wi-Fi), uses encryption in its wireless transmissions to protect against external eavesdropping. Nevertheless, Atkinson showed that Wi-Fi can still leak private user information using only side-channel information similar to those exploited in wired networks [25]. A Random Forests classifier was trained on a dataset of Skype activity with around 60,000 observations and 600 features, and achieved around 97% accuracy. Furthermore, it was shown that a classification accuracy of greater than 95% could be accomplished using only 200 variables and 20 trees. The most valuable features were discovered to be the amount of time between Sent frames, and the amount of time between a Received frame and the previous Sent frame.

As for Internet of Things (IoT) devices, there is a shortage of wireless traffic analysis in the current literature. This is likely due to the relative novelty of the IoT. A study by Copos et al. analyzed two IoT devices, the Nest thermostat and Nest Protect smoke detector, and succeeded in determining the devices' *Home* and *Away* modes with 88% and 67% accuracy respectively [26]. Beyer expanded on the use of wireless traffic analysis by showing that pattern-of-life profiling is possible using sniffed BLE and Wi-Fi traffic [2]. By analyzing both incoming and outgoing packet flows of three device types (outlet, sensor, and cameras), it was shown that 17 out of 18 IoT devices could be classified. Beyer used frame size, measured in bytes, as a primary feature. Meidan et al. is the first study found that applied machine learning to IoT device identification [27]. The goal of the study was to determine whether a network device was a personal computer, a smartphone or an IoT device. A dataset was produced using 802.11 wireless traffic from two PCs, two smartphones, and ten IoT devices, with device types including baby monitors, refrigerators, security cameras, thermostats, and smart outlets. A classifier was then trained using a combination of gradient boosting and Random Forests techniques, and was able to classify an IoT device with greater than 99% accuracy. A robust analysis on feature importance was not performed. Wang et al. is the latest study to apply machine learning techniques to traffic analysis of IoT devices [28]. A software-defined network (SDN) framework was developed capable of efficient network quality-of-service management. This was achieved through the use of deep-learning-based traffic analysis able to classify encrypted data traffic to various applications (e.g., email, Skype video calls, Spotify music streaming, etc.). Three deep-learning techniques were evaluated: multilayer perceptron, stacked autoencoder, and convolutional neural networks.

2.6 Terminology

The following terms are frequently used throughout this thesis, and are defined here for the purpose of clarity:

- Classification Algorithm/Classifier: a classification technique used to distinguish between response classes. The classification algorithms used in this thesis are limited to *k*-nearest neighbors (KNN), linear discriminant analysis (LDA), and random forests (RF).

- Device Type: the broad functional purpose of an IoT device, e.g., cameras, smart plugs, locks. The device type is the response class the classifiers in this thesis are attempting to identify.
- Response Class: the output of supervised classification classifiers. The response class referred to in this thesis is the device type.
- Test dataset: a subset of the dataset reserved to evaluate the classifier's performance. The test dataset is never used to train the classifier.
- Training dataset: a subset of the dataset used to train the classifier.

2.7 Background Summary

This chapter provides a concise summary on the BLE and Wi-Fi protocols that IoT devices use. It explains the underlying architecture of these protocols and how their internal structure apply to machine learning classification. It offers a brief overview of machine learning. It gives a brief survey into the background research on wired traffic analysis and how machine learning has been applied in those efforts. While extensive research has been done in traditional wired traffic, there is a current shortage on IoT traffic analysis. This thesis contributes to the areas of IoT security and machine learning by showing how classification techniques may successfully identify IoT devices using information leaked in conventional wireless traffic.

III. System Design

3.1 Overview

This research offers two contributions in analyzing the classification of IoT devices using wireless traffic analysis: a Smart Home Environment (SHE) and a Device Classification Pipeline (DCP). SHE is a system of actual IoT devices that produces authentic smart home traffic by integrating various BLE and Wi-Fi commercial devices in one physical space. To study the wireless traffic produced by SHE, DCP is used to collect and preprocess the wireless traffic into workable data, extract features from the data, and produce tuned classifier models ready for testing. Lastly, data exploration is done to gain an initial understanding of the original dataset before any classification is performed. This chapter presents a detailed explanation of SHE, each component of DCP, and the products of data exploration on the datasets.

3.2 Smart Home Environment (SHE)

SHE is designed to produce actual wireless IoT traffic for analysis. SHE consists of a controller device, and a variety of BLE and Wi-Fi devices. Wireless traffic produced by SHE is collected using sniffing equipment.

3.2.1 Controller

An iPhone 6S, using iOS version 12.1, serves as the central controller for all devices in SHE. The iPhone controls a given device using the manufacturer-developed application. All applications can be obtained via the App Store. When the user is within range of SHE, the iPhone connects to the Internet via the Wi-Fi router and gains access to the devices in SHE. To interface with BLE devices, the iPhone's Bluetooth must be turned on. Once connected through Bluetooth, the user can control the BLE devices using the device’s application.

3.2.2 Wi-Fi Devices

To provide connectivity to Wi-Fi devices, a 2.4 GHz AP is set up with an SSID of “yosemite" and Wi-Fi Protected Access II (WPA2) security. The AP used is a Netgear Nighthawk X4S R7800 router, as shown in Figure 11. Figure 12 provides the list of AP settings.

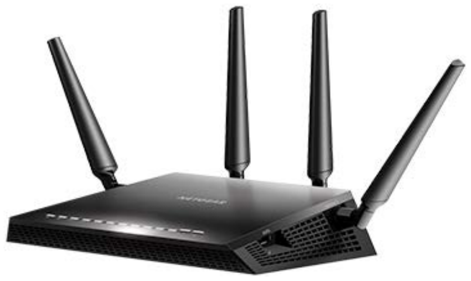
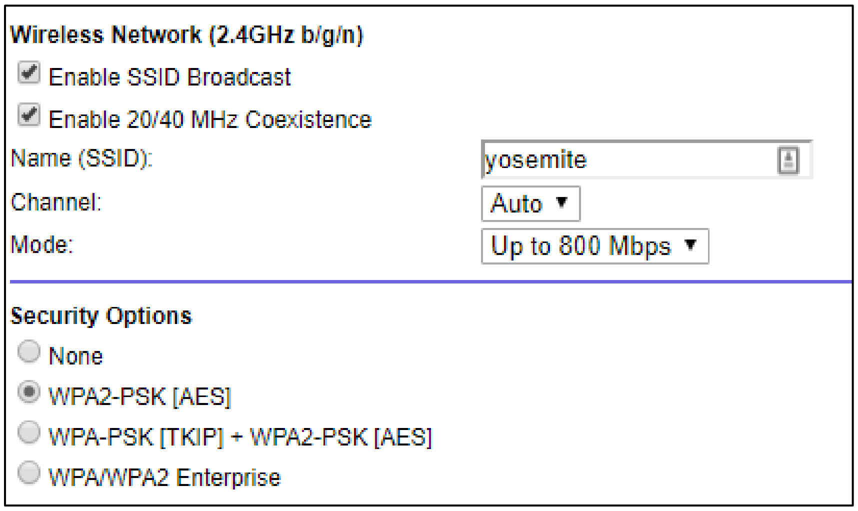


Figure 11. Netgear Nighthawk X4S R7800 router used as AP



Wireless Network (2.4GHz b/g/n)

✓ Enable SSID Broadcast

✓ Enable 20/40 MHz Coexistence

Name (SSID):	yosemite
Channel:	Auto
Mode:	Up to 800 Mbps

Security Options

- ○ None
- ● WPA2-PSK [AES]
- ○ WPA-PSK [TKIP] + WPA2-PSK [AES]
- ○ WPA/WPA2 Enterprise

Figure 12. Wi-Fi Access Point Settings

All Wi-Fi devices, identified as w_2 - w_{16}, are connected to the AP. SHE contains five cameras, seven smart plugs, and three light bulbs across four different manufacturers. Table 2 shows the list of Wi-Fi devices used in SHE, including details such as device type, device manufacturer, device name, and MAC address. Device MAC addresses are found by checking the labels on the physical device. Appendix A provides complete device details, including a device's model number and serial number.

Table 2. Wi-Fi Devices

ID	Manufacturer	Device Type	Device Name	MAC Address
w_1	Netgear	Router	Yosemite	78:D2:94:4D:AB:3E
w_2	Belkin	Camera	Netcam1	EC:1A:59:E4:FD:41
w_3	Belkin	Camera	Netcam2	EC:1A:59:E4:FA:09
w_4	Belkin	Camera	Netcam3	EC:1A:59:E5:02:0D
w_5	Dropcam	Camera	Dropcam	30:8C:FB:3A:1A:AD
w_6	TPLink	Camera	Kasa	AC:84:C6:97:7C:CC
w_7	Belkin	Plug	Insight	14:91:82:24:DD:34
w_8	Belkin	Plug	Mini	60:38:E0:EE:7C:E5
w_9	Belkin	Plug	Switch1	14:91:82:CD:DF:3D
w_{10}	Belkin	Plug	Switch2	B4:75:0E:0D:94:65
w_{11}	Belkin	Plug	Switch3	B4:75:0E:0D:33:D5
w_{12}	Belkin	Plug	Switch4	94:10:3E:2B:7A:55
w_{13}	TPLink	Plug	TpPlug	70:4F:57:F9:E1:B8
w_{14}	Lifx	Light Bulb	Lifx1	D0:73:D5:26:B8:4C
w_{15}	Lifx	Light Bulb	Lifx2	D0:73:D5:26:C9:27
w_{16}	TPLink	Light Bulb	TpBulb	B0:4E:26:C5:2A:41

3.2.3 BLE Devices

To control BLE devices b_1 - b_{11} , the iPhone is used as a Bluetooth master device. SHE contains three locks, four door sensors, and four temperature sensors across five different manufacturers. Table 3 shows the BLE devices used in the smart home environment, including details such as device type, device manufacturer, and device name. Appendix A provides complete device details, including a device's model number and serial number.

Table 3. BLE Devices

ID	Manufacturer	Device Type	Device Name
b_1	August	Lock	August1
b_2	August	Lock	August2
b_3	Kwikset	Lock	Kevo
b_4	BLE Home	Door Sensor	Home1
b_5	BLE Home	Door Sensor	Home2
b_6	Eve	Door Sensor	Door1
b_7	Eve	Door Sensor	Door2
b_8	Eve	Temperature Sensor	Room1
b_9	Eve	Temperature Sensor	Room2
b_{10}	Eve	Temperature Sensor	Weather
b_{11}	SensorPush	Temperature Sensor	Push

3.2.4 Device Actions

To produce sufficient traffic volume for machine learning, devices are set to perform actions expected in a smart home environment. Actions can be programmed or triggered. Programmed actions are set up by the user to occur on a scheduled time and interval, while triggered actions are performed by devices upon an event. Events can occur at any time during the experimentation. To reduce variability, devices of the same device type are programmed to perform similar actions at identical times or events. Table 4 lists the complete set of actions used in SHE.

Table 4. Device Actions

	Device Name	Device Type	Actions	Schedule/Event
1	Dropcam	Camera	Send email notification	Motion detected
2	Kasa	Camera	Send email notification	Motion detected
3	Netcam1	Camera	Send email notification	Motion detected
4	Netcam2	Camera	Send email notification	Motion detected
5	Netcam3	Camera	Send email notification	Motion detected
6	Lifx1	Light Bulb	Turn on, Turn off	Hourly, on the hour
7	Lifx2	Light Bulb	Turn on, Turn off	Hourly, on the hour
8	TpBulb	Light Bulb	Turn on, Turn off	Hourly, on the hour
9	TpPlug	Plug	Turn on, Turn off	Hourly, on the hour
10	Insight	Plug	Turn on, Turn off	Hourly, on the hour
11	Mini	Plug	Turn on, Turn off	Hourly, on the hour
12	Switch1	Plug	Turn on, Turn off	Hourly, on the hour
13	Switch2	Plug	Turn on, Turn off	Hourly, on the hour
14	Switch3	Plug	Turn on, Turn off	Hourly, on the hour
15	Switch4	Plug	Turn on, Turn off	Hourly, on the hour
16	Door1	Door Sensor	Report door state	Continuous
17	Door2	Door Sensor	Report door state	Continuous
18	Home1	Door Sensor	Report door state	Continuous
19	Home2	Door Sensor	Report door state	Continuous
20	August1	Lock	Report lock state	Continuous
21	August2	Lock	Report lock state	Continuous
22	Kevo	Lock	Report lock state	Continuous
23	Room1	Temp Sensor	Report temperature	Continuous
24	Room2	Temp Sensor	Report temperature	Continuous
25	Weather	Temp Sensor	Report temperature	Continuous
26	Push	Temp Sensor	Report temperature	Continuous

3.2.5 Device Location and Setup

All device locations are kept constant in SHE, except for the iPhone controller which typically is on the user and is therefore not at a fixed location. Figure 13 shows the location of each device in SHE. Wi-Fi devices are indicated by boxes with solid outlines, and BLE devices are indicated by boxes with dashed outlines. The area containing SHE is a one-bedroom apartment with three doors, one set of large windows, a large table, and a couch; these items are indicated by grey boxes. Devices locations are placed near power outlets to reduce the need for additional power cords.

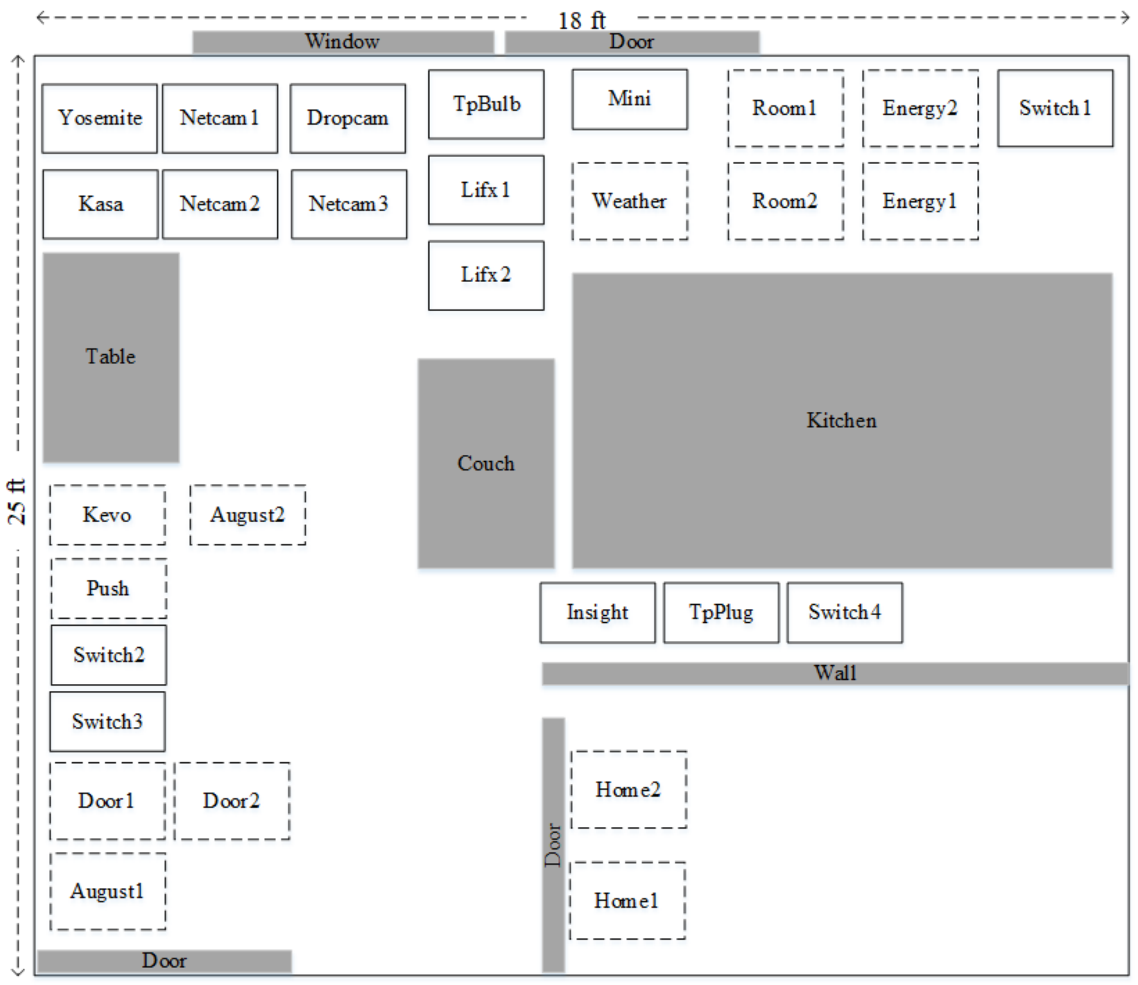


Figure 13. SHE Device Locations (not to scale)

Figures 13 to 17 show how various devices are set up. Not all devices are shown in the figures, but the overall configuration is similar across device types. Door sensors are placed along the edges of doors, with their door sensor magnets located across them. Light bulbs are installed into individual bulb sockets. Locks are installed into 3D-printed door lock holders with actual door lock parts included. Plugs are installed into either power strips or outlets, with no plugged-in device. Temperature sensors are placed in a flat surface with open space around them. Cameras are placed around a motion source to provide motion for their sensing capabilities. Figure 38 provides a photograph of the camera device setup.

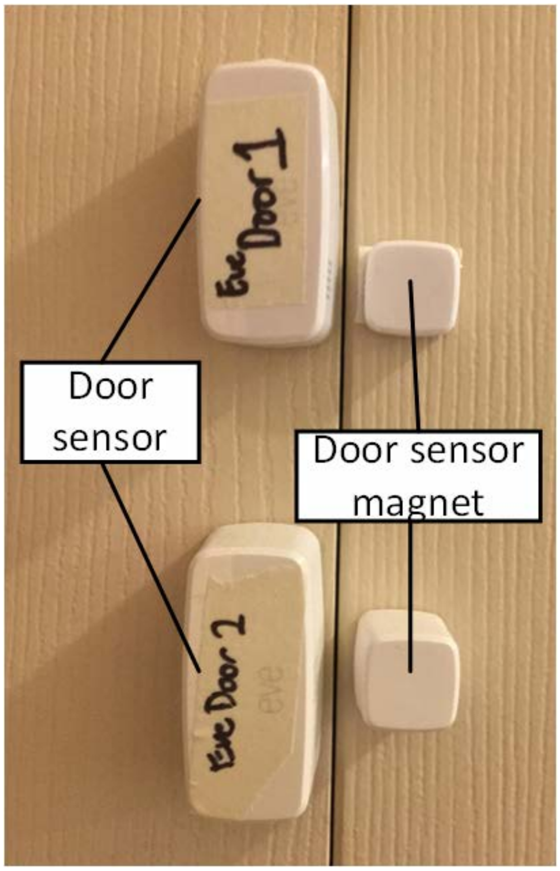


Figure 14. Door Sensor Setup

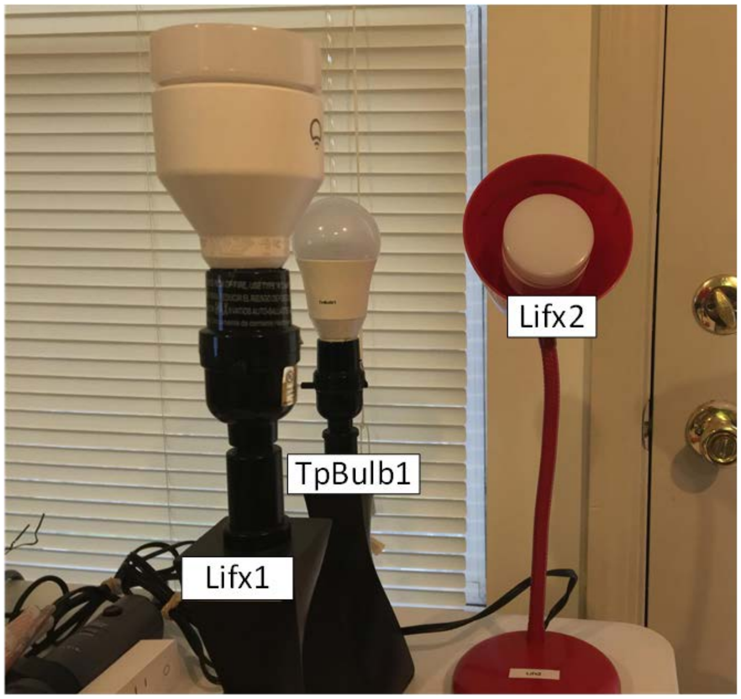


Figure 15. Light Bulb Setup

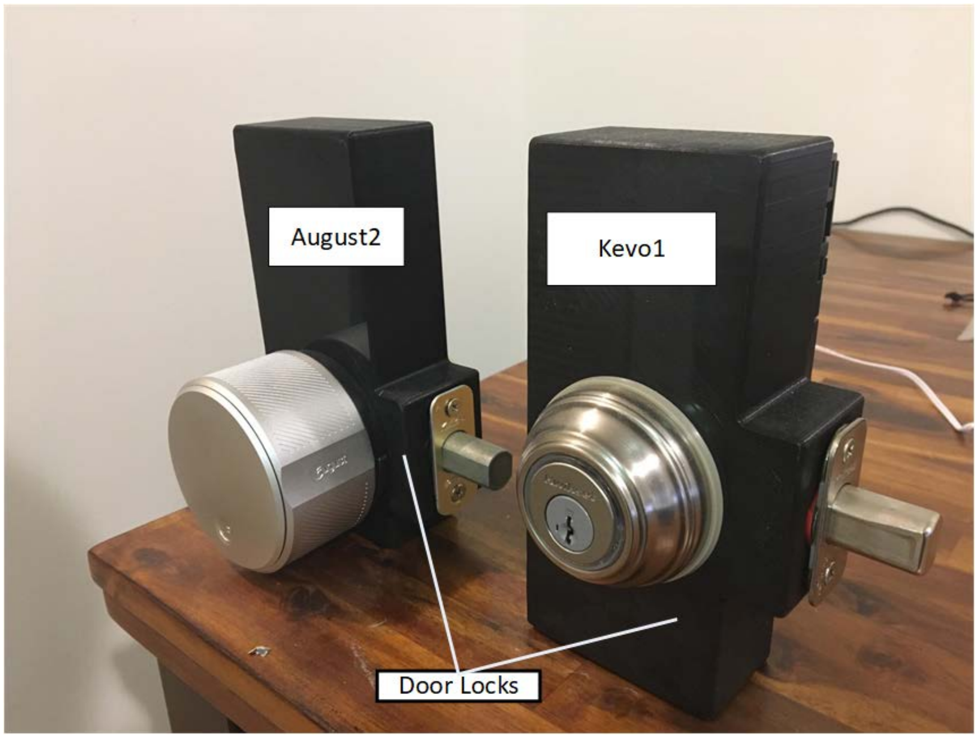


Figure 16. Lock Setup

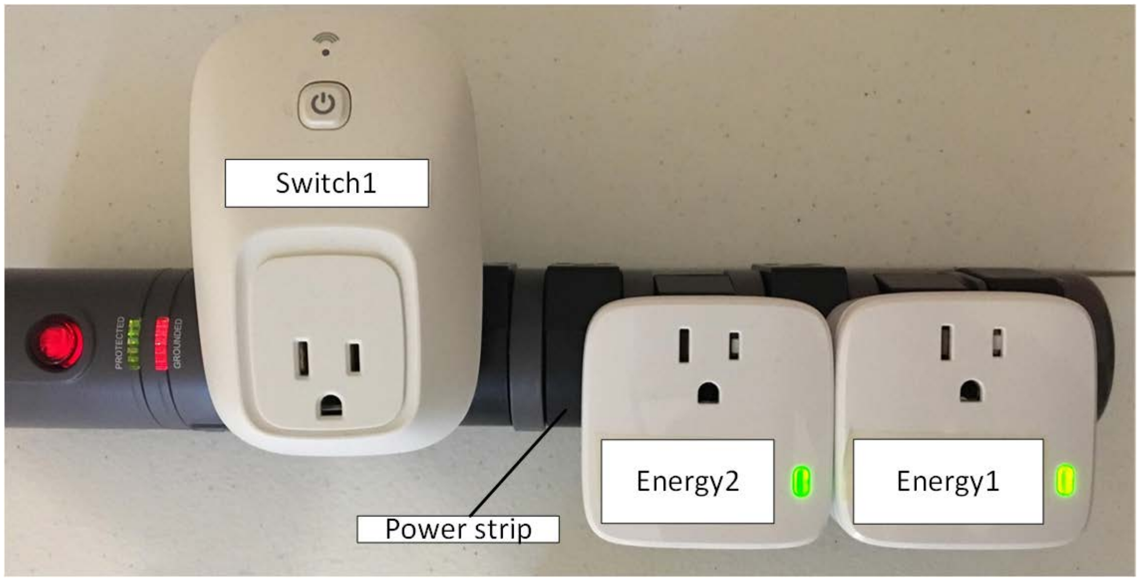


Figure 17. Plug Setup

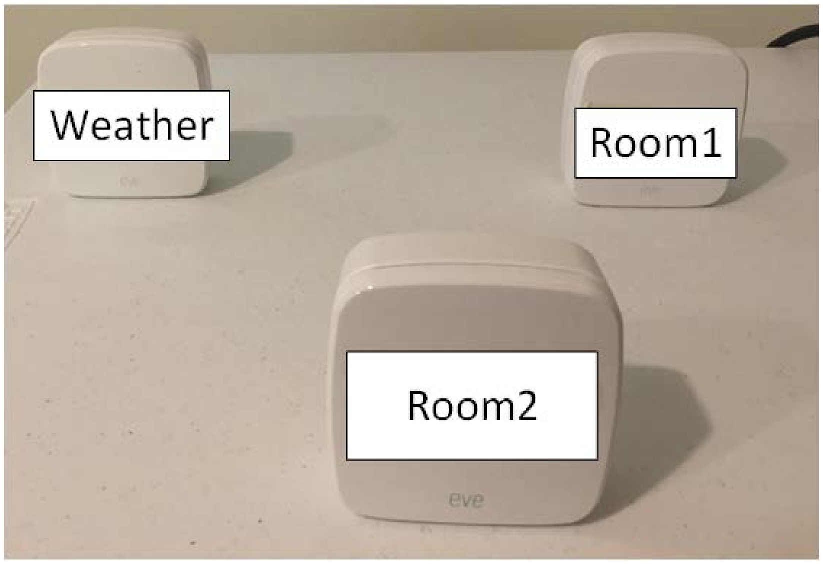


Figure 18. Temperature Sensor Setup

3.3 Device Classification Pipeline (DCP)

DCP is designed to collect and preprocess the wireless traffic into workable data, extract features from the data, and produce tuned classifier models ready for testing. Figure 19 shows the DCP system diagram and its three components: (i) data collection, (ii) data preprocessing, and (iii) model tuning. The outputs of DCP are tuned linear discriminant analysis (LDA), *k*-nearest neighbors (KNN), and random forests (RF) classifiers for model testing, as well as the test dataset. The following sections describe each component and their functions.

Data collection is accomplished using user-inputted commands and scripts, as described in Section 3.3.2, while data preprocessing and model tuning are accomplished using the `MulticlassDCP` class written in Python. The `MulticlassDCP` class contains two sub-classes, `BLEMulticlassDCP` and `WifiMulticlassDCP`, one for each protocol.

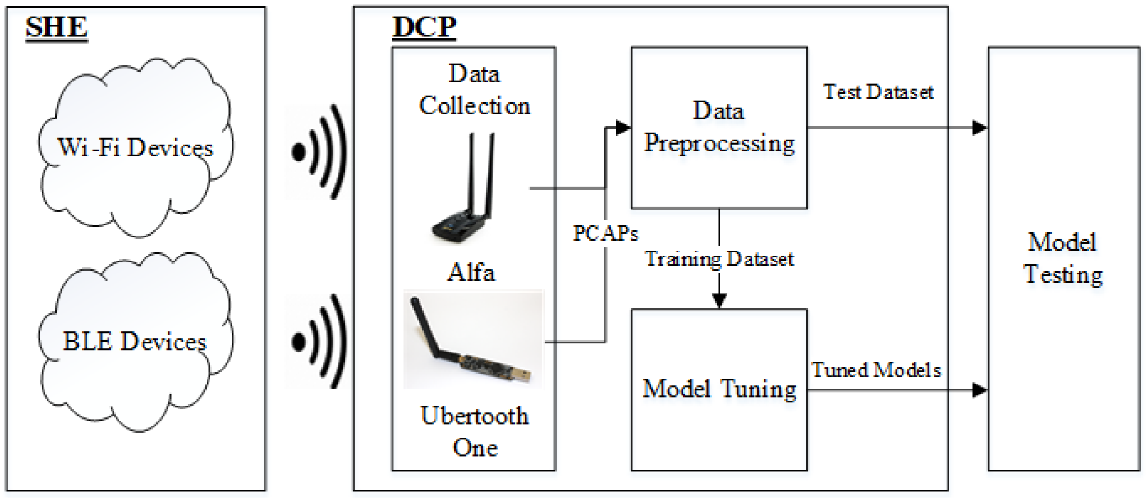


Figure 19. DCP System Diagram

3.3.1 Data Collection Hardware

A workstation is used to run all components of DCP. The workstation is an Acer Aspire E15 with a 64-bit Intel Core i5-6200U 2.3 GHz processor, 8 GB DD4 RAM, 256 GB solid-state hard drive, and runs Kali Linux 2018.4 as the operating system. The scanning and sniffing equipment consists of the Plugable Bluetooth adapter, three BLE sniffers (Ubertooth One with firmware 2018-08-R1), and a long-range dual-band Wi-Fi adapter (Alfa AWUS036ACH); all equipment is connected to the workstation using Universal Serial Bus (USB). Each Ubertooth One sniffer uses a 2.4 GHz 2.2 dBi antenna, while the Alfa card uses a 2.4 GHz and 5 GHz dual-band dipole antenna. Figure 20 provides images for these equipment.

Figure 20. Scanning and Sniffing Equipment. Plugable Bluetooth adapter (left), Alfa AWUS036ACH Wi-Fi adapter (center), and Ubertooth One BLE sniffer (right) [29]–[31]

3.3.1.1 Sniffer Distance Separation

All sniffers, Wi-Fi and BLE, operate in the 2.4 GHz band and must be horizontally isolated to prevent interference. The required distance between antennae, d, to ensure horizontal isolation is given by the Fraunhofer distance equation

$$\boldsymbol{d \geq 2\frac{D^2}{\lambda}} \quad (1)$$

where D is the antenna length in meters and λ is the wavelength of the device frequency band in Hz [32]. The Ubertooth One sniffers have 3.5 inch long antennae and operate at an average wavelength of 2441 MHz, and the Alfa card has 6.5 inch long antennae and operate at an average wavelength of 2412 MHz. Applying these values to (1) yields a separation distance of about 5 inches for the Ubertooth One sniffers and 17 inches for the Alfa card antenna. Separation distances are maintained by affixing the Ubertooth One sniffers on a wooden board and ensuring that the Alfa card is located at an appropriate distance, as shown in Figure 21.

3.3.2 Data Collection

Data collection supplies the raw data needed for data preprocessing. Data collection occurs in two steps: scanning and sniffing. Scanning collects necessary information required for sniffing and analysis, and sniffing gathers BLE and Wi-Fi wireless traffic and stores them into packet capture files (pcap). The scanning and sniffing procedures vary for BLE and Wi-Fi devices.

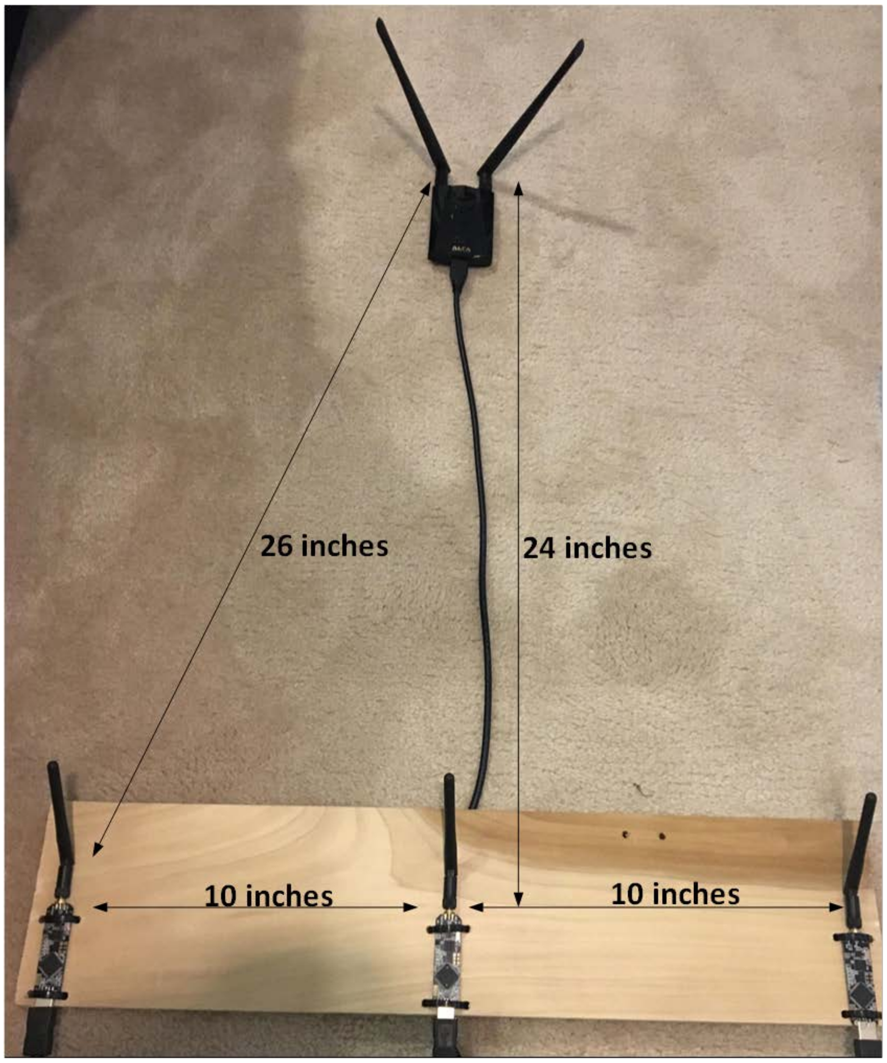


Figure 21. Sniffer Layout

3.3.2.1 Wi-Fi Devices

Wi-Fi scanning is used to find the AP MAC address, AP channel, and associated Wi-Fi device MAC addresses. Prior to scanning, the Alfa card is plugged into the workstation via USB. Figure 22 shows the commands used to prepare the Alfa card for scanning (note that the specific wireless interface "wlan1" may vary on other devices):

(i) `airmon-ng check kill` – end any processes that may affect operation,

(ii) `ifconfig wlan1 down` – turn off the wireless interface,

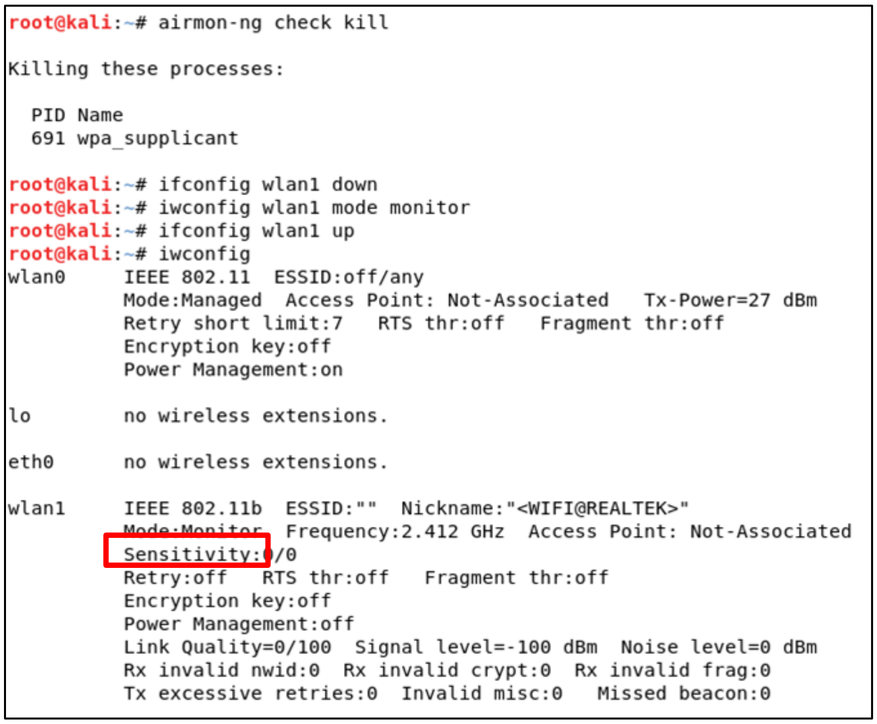
(iii) `iwconfig wlan1 mode monitor` – set wireless interface to monitor mode,

(iv) `ifconfig wlan1 down` – turn on the wireless interface, and

(v) `iwconfig` – verify that the changes occurred successfully. The wireless interface (in this case, wlan1) should be set to monitor mode, as indicated by the red box.

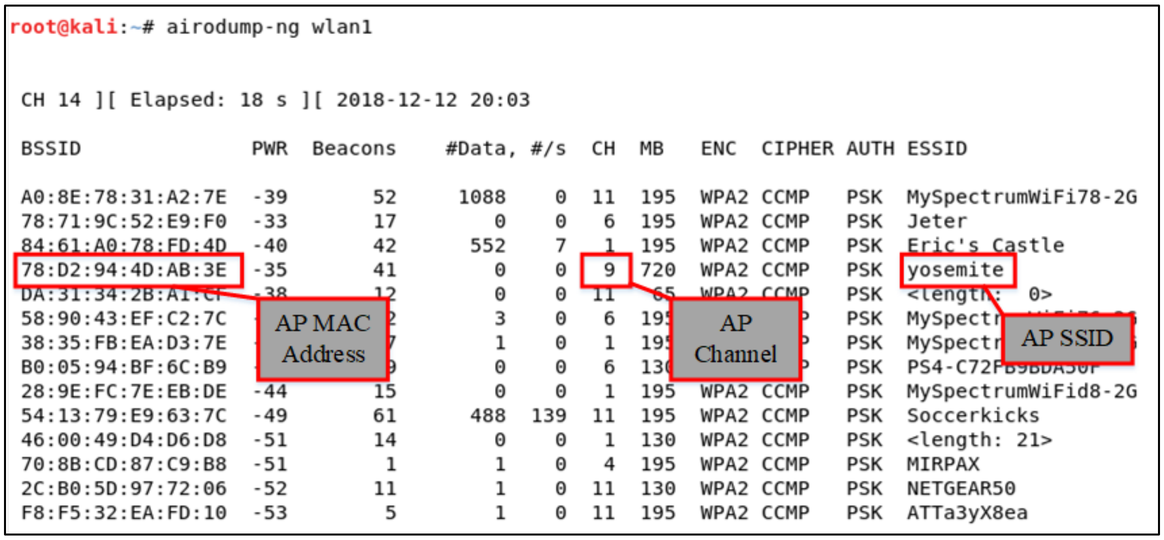
Figure 23 shows the command used to scan for APs. This scan discovers the following information: (1) the target AP's MAC address, (2) AP channel, and (3) AP SSID.

The next step is scanning for all Wi-Fi devices associated with the target AP. Figure 24 shows the command used to accomplish this. The resulting list of MAC addresses is compared against the list in Table 2 to ensure all Wi-Fi devices are detected.



```
root@kali:~# airmon-ng check kill

Killing these processes:

  PID Name
  691 wpa_supplicant

root@kali:~# ifconfig wlan1 down
root@kali:~# iwconfig wlan1 mode monitor
root@kali:~# ifconfig wlan1 up
root@kali:~# iwconfig
wlan0     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=27 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:on

lo        no wireless extensions.

eth0      no wireless extensions.

wlan1     IEEE 802.11b  ESSID:""  Nickname:"<WIFI@REALTEK>"
          Mode:Monitor  Frequency:2.412 GHz  Access Point: Not-Associated
          Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=0/100  Signal level=-100 dBm  Noise level=0 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Figure 22. Commands to prepare Alfa card for data collection

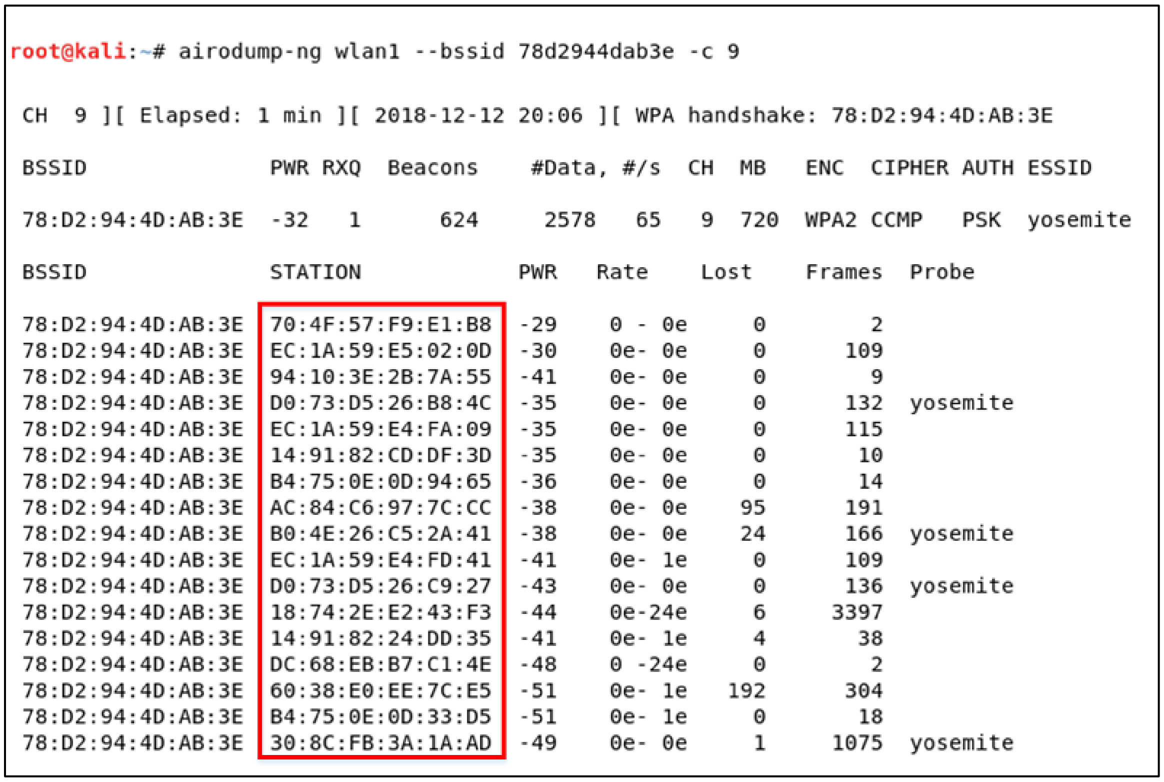


```
root@kali:~# airodump-ng wlan1

 CH 14 ][ Elapsed: 18 s ][ 2018-12-12 20:03

 BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 A0:8E:78:31:A2:7E  -39       52     1088    0  11  195  WPA2 CCMP   PSK  MySpectrumWiFi78-2G
 78:71:9C:52:E9:F0  -33       17        0    0   6  195  WPA2 CCMP   PSK  Jeter
 84:61:A0:78:FD:4D  -40       42      552    7   1  195  WPA2 CCMP   PSK  Eric's Castle
 78:D2:94:4D:AB:3E  -35       41        0    0   9  720  WPA2 CCMP   PSK  yosemite
 DA:31:34:2B:A1:CF  -38       12        0    0  11   65  WPA2 CCMP   PSK  <length:  0>
 58:90:43:EF:C2:7C             2        3    0   6  195          P   PSK  MySpectr
 38:35:FB:EA:D3:7E             7        1    0   1  195          P   PSK  MySpectr
 B0:05:94:BF:6C:B9             9        0    0   6  130          P   PSK  PS4-C72F
 28:9E:FC:7E:EB:DE  -44       15        0    0   1  195  WPA2 CCMP   PSK  MySpectrumWiFid8-2G
 54:13:79:E9:63:7C  -49       61      488  139  11  195  WPA2 CCMP   PSK  Soccerkicks
 46:00:49:D4:D6:D8  -51       14        0    0   1  130  WPA2 CCMP   PSK  <length: 21>
 70:8B:CD:87:C9:B8  -51        1        1    0   4  195  WPA2 CCMP   PSK  MIRPAX
 2C:B0:5D:97:72:06  -52       11        1    0  11  130  WPA2 CCMP   PSK  NETGEAR50
 F8:F5:32:EA:FD:10  -53        5        1    0  11  195  WPA2 CCMP   PSK  ATTa3yX8ea
```

Figure 23. Commands used to scan for Wi-Fi AP

Wi-Fi sniffing is used to collect wireless traffic from the Wi-Fi devices. Prior to scanning, ensure that the Alfa card is first set to monitor mode (see Figure 22. If monitor mode is not set, execute all the commands in Figure 22). The airodump-ng tool from the aircrack-ng suite is then used to capture raw Wi-Fi frames. To use airdump-ng, the wireless interface ("wlan1"), output file format ("pcap"), target AP MAC address ("78d2944dab3e"), and target AP channel ("9") must be set. The command is

```
airodump-ng wlan1 –o pcap –w wifi –bssid 78d2944dab3e –c 9
```



```
root@kali:~# airodump-ng wlan1 --bssid 78d2944dab3e -c 9

 CH  9 ][ Elapsed: 1 min ][ 2018-12-12 20:06 ][ WPA handshake: 78:D2:94:4D:AB:3E

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 78:D2:94:4D:AB:3E  -32   1      624     2578   65   9  720  WPA2 CCMP   PSK  yosemite

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 78:D2:94:4D:AB:3E  70:4F:57:F9:E1:B8  -29    0 - 0e      0        2
 78:D2:94:4D:AB:3E  EC:1A:59:E5:02:0D  -30    0e- 0e      0      109
 78:D2:94:4D:AB:3E  94:10:3E:2B:7A:55  -41    0e- 0e      0        9
 78:D2:94:4D:AB:3E  D0:73:D5:26:B8:4C  -35    0e- 0e      0      132  yosemite
 78:D2:94:4D:AB:3E  EC:1A:59:E4:FA:09  -35    0e- 0e      0      115
 78:D2:94:4D:AB:3E  14:91:82:CD:DF:3D  -35    0e- 0e      0       10
 78:D2:94:4D:AB:3E  B4:75:0E:0D:94:65  -36    0e- 0e      0       14
 78:D2:94:4D:AB:3E  AC:84:C6:97:7C:CC  -38    0e- 0e     95      191
 78:D2:94:4D:AB:3E  B0:4E:26:C5:2A:41  -38    0e- 0e     24      166  yosemite
 78:D2:94:4D:AB:3E  EC:1A:59:E4:FD:41  -41    0e- 1e      0      109
 78:D2:94:4D:AB:3E  D0:73:D5:26:C9:27  -43    0e- 0e      0      136  yosemite
 78:D2:94:4D:AB:3E  18:74:2E:E2:43:F3  -44    0e-24e      6     3397
 78:D2:94:4D:AB:3E  14:91:82:24:DD:35  -41    0e- 1e      4       38
 78:D2:94:4D:AB:3E  DC:68:EB:B7:C1:4E  -48    0 -24e      0        2
 78:D2:94:4D:AB:3E  60:38:E0:EE:7C:E5  -51    0e- 1e    192      304
 78:D2:94:4D:AB:3E  B4:75:0E:0D:33:D5  -51    0e- 1e      0       18
 78:D2:94:4D:AB:3E  30:8C:FB:3A:1A:AD  -49    0e- 0e      1     1075  yosemite
```

Figure 24. Command used to scan for Wi-Fi devices associated to the AP

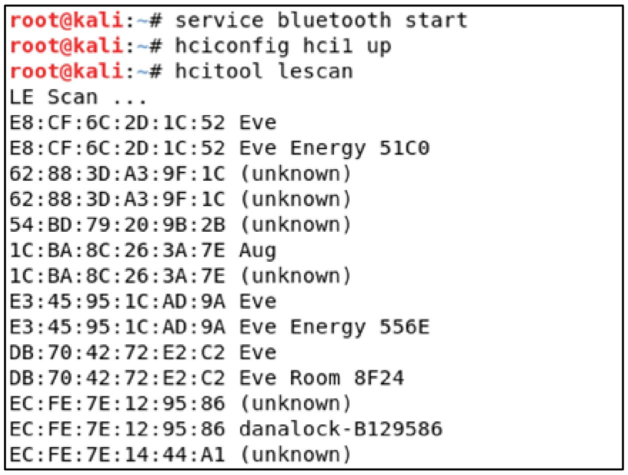
3.3.2.2 BLE Devices

BLE scanning is used to discover device names and device addresses from advertising devices. Prior to scanning, the Plugable adapter is connected via USB. Figure 25 shows the commands used to scan for advertising BLE devices:

(i) `service bluetooth start` – start the Bluetooth service,

(ii) `hciconfig hci1 up` – open and initialize the Bluetooth device, and

(iii) `hcitool lescan`– scan for BLE devices.



```
root@kali:~# service bluetooth start
root@kali:~# hciconfig hci1 up
root@kali:~# hcitool lescan
LE Scan ...
E8:CF:6C:2D:1C:52 Eve
E8:CF:6C:2D:1C:52 Eve Energy 51C0
62:88:3D:A3:9F:1C (unknown)
62:88:3D:A3:9F:1C (unknown)
54:BD:79:20:9B:2B (unknown)
1C:BA:8C:26:3A:7E Aug
1C:BA:8C:26:3A:7E (unknown)
E3:45:95:1C:AD:9A Eve
E3:45:95:1C:AD:9A Eve Energy 556E
DB:70:42:72:E2:C2 Eve
DB:70:42:72:E2:C2 Eve Room 8F24
EC:FE:7E:12:95:86 (unknown)
EC:FE:7E:12:95:86 danalock-B129586
EC:FE:7E:14:44:A1 (unknown)
```

Figure 25. Commands used to scan for BLE devices

BLE sniffing is used to collect wireless traffic from the BLE devices. Prior to scanning, three Ubertooth One sniffers are connected to the workstation via USB. Each Ubertooth One device (“U0” – “U2”) is set to sniff on one of three advertisement channels (“A37”-“A39”), to follow connections (“f”), and to create a pcap output file (“q”). To operate a single Ubertooth One, the command is

```
ubertooth-btle -f -U0 -A37 -qble.pcap
```

3.3.3 Data Preprocessing

Data preprocessing changes the collected raw data into a dataset suitable for machine learning classifiers. Data preprocessing serves two objectives: feature extraction, and data transformation. Feature extraction is used to build numerical and categorical values (features) that represent information contained in the raw data. Data transformation is used to format features into a representation more suitable for the machine learning classifiers. Data preprocessing varies between Wi-Fi and BLE devices, and are discussed separately. Scaling is not performed for any numerical feature.

3.3.3.1 Wi-Fi Preprocessing

Wi-Fi preprocessing is used to create a dataframe containing the device type response class, associated packet count feature, packet length feature, packet subtype features, vendor features, device name, set, source address, and packet time.

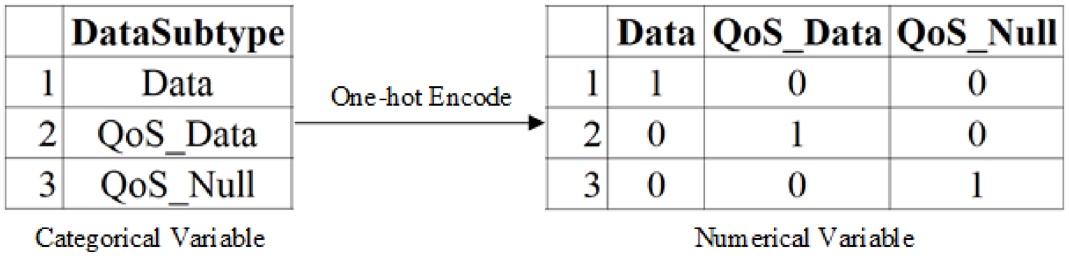
After data collection, the Wi-Fi pcap files are combined into one master pcap file, then parsed using `pyshark`, a Python wrapper for Wireshark packet dissectors. The list of device MAC addresses generated during data collection is then used to create a comma-separated values (csv) file for each known Wi-Fi device. The MAC addresses are collected to determine which devices are part of SHE, but are not used in the classification process. Information about packet time, packet length, and data packet subtype are extracted from each data packet. Time refers to the time the packet was transmitted and is measured in epoch time. Time is not used as a feature and is used for organizational purposes. Packet length is a numerical feature that refers to the size of the entire 802.11 packet and is measured in bytes. Data packet subtype is a categorical feature corresponding to the type of 802.11 data frame used. Frames of subtype 32 are data frames, and are the basic frame type used in data transmission. Frames of subtype 40 and 44 are quality-of-service (QoS) frames, which support latency-sensitive applications such as video and voice-over-IP [33]. QoS data frames (subtype 40) contain higher-protocol data, and function similarly to the standard data frame type. QoS null frames (subtype 44) are frames that transmit no data, but only frame information. QoS null frames are typically used by STAs to notify the AP that the STA is entering a power-save mode. Source MAC addresses are extracted but are not used as features in classification. These features are stored in the csv file of the source device. All other 802.11 packet types and packets with a source address not in the list of device MAC addresses are not used for classification and are not stored in the csv files.

Once all packets in the master pcap are parsed, the csv files are read into a dataframe created using `pandas`, an open-source library that provides high-performance structures for data analysis. Each row in the dataframe represents a single 802.11 packet, and each column in the dataframe represents a feature. The device type response class is added by mapping the source MAC address to a pre-built dictionary. Derived features are then produced using existing features. The vendor feature is produced by mapping the source MAC address to a vendor lookup application programming interface (API) from macvendors.co that returns the vendor name of the wireless chip as registered in the IEEE Standards Association [29][30]. Table 5 provides the complete list of Wi-Fi vendors. The wireless chip used by a certain device is not necessarily tied to the vendor, therefore differences between the device vendor and chip vendor are possible. For example, the Lifx light bulbs use wireless chips produced by Lifi.

Table 5. Wi-Fi Vendor List

	Vendors
1	Belkin
2	Dropcam
3	Lifi
4	TP-Link

The associated packets feature is then extracted. The associated packets feature is a numerical feature that refers to the number of packets of a given device sent within one second of each other, and is calculated using the transmission time feature. Categorical features are one-hot encoded to allow for classification by algebraic classifiers (i.e., LDA). One-hot encoding transforms a single categorical feature with k categories into k features where binary values are used to represent inclusion in a given category. Figure 26 provides an example using the data packet subtype feature. In the example, three packets each have a different data packet subtype stored in the categorical feature *DataSubtype*. Through one-hot encoding, the *DataSubtype* feature is transformed into three separate features, each corresponding to the three data subtypes. For convenience during preprocessing, the device names as listed in Table 2 are added as a dataframe variable to easily identify the source of a given packet, and is not used for classification.



	DataSubtype
1	Data
2	QoS_Data
3	QoS_Null

	Data	QoS_Data	QoS_Null
1	1	0	0
2	0	1	0
3	0	0	1

Figure 26. One-Hot Encoding

Packets are assigned to either the training set or test set, based on their source device. Packets belonging to devices in the training set are used to build the classifiers, and packets belonging to devices in the test set are used to evaluate the performance of the classifiers. Table 7 provides the complete list of Wi-Fi device set assignments. The camera device type has three devices in the training set, the light bulb device type has two, and the plug device type has five devices. The camera and plug device types each have two devices in the test set, while the light bulb device type has one. Device availability is the primary reason for the dissimilarity in device count.

Table 6 summarizes the dataframe columns produced in Wi-Fi preprocessing. The name of the dataframe column is provided, along with its use (categorical feature, numerical feature, dataframe variable which is not used for classification but used for organizational purposes only, or response class), definition, and unit or accepted values.

Table 6. Wi-Fi Dataframe Columns

	Attribute Name	**Definition**	**Value Type/ ML Use**	**Unit/Values**
1	Associated Packet Count	Number of packets of a device sent within one second	Numerical Feature	Packets per second
2	Device Name	Name given to device by user	Information	See Table 2
3	Device Type	Category of device	Response Class	See Table 2
4	Packet Length	Size of 802.11 packet	Numerical Feature	Bytes
5	Packet Subtype	802.11 data packet type	Categorical Feature	[Data, QoS data, QoS null]
6	Set	Assignment of device as training or test device	Information	See Table 7
7	Source Address	MAC address of source device	Information	See Table 2
8	Time	Time of packet transmission	Information	Epoch Time
9	Vendor	Vendor of wireless chip	Categorical Feature	See Table 5

Table 7. Wi-Fi Device Set Assignment

Device Type	Training Set	Test Set
Camera	Dropcam	Kasa
	Netcam1	Netcam3
	Netcam2	
Light Bulb	Lifx1	Lifx2
	TpBulb	
Plug	Insight	TpPlug
	Switch1	Switch4
	Switch2	
	Switch3	
	Mini	

3.3.3.2 BLE Preprocessing

BLE preprocessing is used to create a dataframe containing the device type response class, associated packet count feature, packet length feature, BLE link layer header length feature, protocol data unit (PDU) type feature, radio frequency (RF) channel number feature, device name, set, and packet time.

Similar to Wi-Fi preprocessing, the BLE pcap files are combined into one master pcap file, and parsed using `pyshark`. The link layer device names and advertising addresses are first extracted during parsing, but are not included as features. Instead, they are used to identify which packets belong to the known devices in SHE. A csv file is then created for each known BLE device. The following information is extracted from each BLE packet belonging to a known device: time, length, RF channel, link layer (LL) packet length, and PDU type. Time refers to the time the packet was transmitted, and is measured in epoch time. Time is not used as a feature and is used for organizational purposes. Length refers to the size of the entire BLE packet, and is measured in bytes. RF channel is a categorical feature with values 0, 12, and 39, each corresponding to the radio frequency channel from which the packet was sniffed. LL packet length refers to the length of the BLE link layer header, and is measured in bytes. PDU type is a categorical feature with values corresponding to each advertising PDU type (see Section 2.3.2). These features are then stored in the csv file of the source device. All other BLE packet types and packets from unknown devices are not used for classification and are not stored in the csv files. Once all packets in the master pcap are parsed, the csv files are read into a `pandas` dataframe, with each row in the dataframe representing a single BLE packet, and each column in the dataframe representing a feature. The device type response class is added by mapping either the LL device name or advertising address to a pre-built dictionary. The associated packets feature is then derived using the same method described in Section 3.3.3.1. All categorical features are then one-hot encoded. For convenience during preprocessing, the device names as listed in Table 3 are added in the dataframe to easily identify the source of a given packet, and are not used for classification.

Packets are then assigned to either the training set or test set, based on their source device. Packets belonging to devices in the training set are used to build the classifiers, and packets belonging to devices in the test set are used to evaluate the performance of the classifiers. Table 8 provides the complete list of BLE device set assignments. The door sensor and temperature sensor device type each have three devices in the training set, while the lock device types has two devices in the training set. All device types have one device in the test set. Device availability is the primary reason door sensors have an extra device.

Table 8. BLE Device Set Assignments

Device Type	**Training Set**	**Test Set**
Door Sensor	Home1	Door2
	Home2	
	Door1	
Temp Sensor	Room1	Room2
	Push	
	Weather	
Lock	August1	August2
	Kevo	

Table 9 summarizes the dataframe columns produced in BLE preprocessing. The name of the dataframe column is provided, along with its type (categorical feature, numerical feature, dataframe variable which is used for organizational purposes only, or response class), definition, and unit or accepted values.

Table 9. BLE Dataframe Columns

	Attribute Name	Definition	Value Type/ ML Use	Unit/Values
1	Associated Packet Count	Number of packets of a device sent within one second	Numerical Feature	Packets per second
2	Device Name	Name given to device by user	Information	See Table 3
3	Device Type	Category of device	Response Class	See Table 3
4	Link Layer Header Length	Length of BLE link layer	Numerical Feature	Bytes
5	Packet Length	Size of BLE packet	Numerical Feature	Byte
6	PDU Type	Advertising PDU Type	Categorical Feature	See Table 1
7	RF Channel	RF Channel on which packet was sent	Categorical Feature	[0, 12, 39]
8	Set	Assignment of device as training or test device	Information	See Table 8
9	Time	Time of packet transmission	Information	Epoch Time

3.3.4 Model Tuning

Model tuning uses the preprocessed datasets to find the classifier hyperparameters with the best performance during cross-validation. The optimal hyperparameters are then used to create classifier models for model testing. The classification algorithms used are implemented by `scikit-learn` version 0.20, an open-source machine learning package written in Python [36].

Models for the BLE and Wi-Fi datasets are tuned separately, however the tuning strategy for both protocols is identical. A range of possible hyperparameter values is selected and evaluated using 10-fold cross-validation grid search, an exhaustive search that evaluates each model using all hyperparameter values. The scoring metric used to evaluate the models is the Matthews correlation coefficient (MCC) (see Section 4.4). The MCC metric provides a measure of a classifier's overall classification performance. The MCC metric exists in the range $[-1, 1]$, where -1 represents perfect misclassification and 1 represents perfect classification. The MCC metric is chosen over the traditionally used accuracy metric because accuracy provides misleading information in imbalanced datasets, such as this one. MCC not only accounts for class imbalances, but also provides a convenient range of values to evaluate classifier performance.

Table 10 provides the complete list of the hyperparameters considered during tuning. The KNN hyperparameter, `n_neighbors`, determines the number of neighbors KNN considers in its classification of a given observation. The values used are the odd numbers in the range 1 to 19. Larger values are originally tested but required significant times to complete. The value range 1 to 19 provides a reasonable tuning range without increasing computation costs significantly. Two RF hyperparameters are tuned: `max_features` and `n_estimators`. The `max_features` hyperparameter determines the size of random subsets of features RF considers when splitting a node [36]. The values used are 2, 3, 5, 7, and 9 for Wi-Fi tuning and 2, 4, 7, 9, and 12 for BLE tuning. These values are chosen because they are evenly-spaced integers that do not exceed the total number of features used, which is 9 features for Wi-Fi tuning and 12 features for BLE tuning. The `n_estimators` hyperparameter determines the number of decision tree the RF model builds. The values used are 10, 15, 20, and 25 for both BLE and Wi-Fi tuning. While other hyperparameters are available, the tuning process is limited to these hyperparameters to allow for efficiency and speed, as increasing the number of hyperparameters compounds the amount of time needed for tuning.

No hyperparameter tuning is necessary for the LDA classifier. However, unlike the KNN and RF classifiers which are able to inherently handle imbalanced multi-classification tasks, the LDA classifier requires additional information. Because of the class imbalances in the test dataset, the class prior probabilities are required by the LDA classifier. The class prior probabilities for the Wi-Fi dataset are 61.68%, 37.82%, and 0.51% for the plug, camera, and bulb device types respectively. The class prior probabilities for the BLE dataset are 59.06%, 23.40%, and 17.54% for the door sensor, temperature sensor, and lock device types respectively.

Table 10. Hyperparameters used in Grid Search

	Hyperparameter Name	Grid Values
KNN	`N_neighbors`	[1 3 5 7 9 11 13 15 17]
RF (Wi-Fi)	`Max_features`	[2 3 5 7 9]
RF (BLE)	`Max_features`	[2 4 7 9 12]
RF (All)	`N_estimators`	[10 15 20 25]

Hyperparameter tuning is performed using a Jupyter notebook written in Python (see Appendix B) [37]. For both Wi-Fi and BLE datasets, the script preprocesses the data and tunes an untuned KNN and RF classifier using the hyperparameter values in Table 10. When finished, the script reports the hyperparameter values that produced the best performing classifiers. Table 11 provides the results of the cross-validation hyperparameter tuning process. The hyperparameters that performed best on the Wi-Fi dataset are presented in the left side of the table, while the hyperparameters for the BLE dataset are presented in the right. Both Wi-Fi classifiers achieved an MCC of over 0.75, showing that when tuned to these hyperparameters, the classifiers successfully attain a high level of performance. As for the BLE classifiers, both the KNN and RF classifiers also reach a high level of performance on the MCC metric. The hyperparameter values used show no signs of improvement past the range of values used, and so it is concluded that these hyperparameter values are used for experimental testing.

Table 11. Best-Performing Hyperparameters

	Wi-Fi		**BLE**	
	Hyperparameter Values	**MCC**	**Hyperparameter Values**	**MCC**
KNN	`N_neighbors` = 11	0.961	`N_neighbors` = 5	0.960
RF	`Max_features` = 2	0.975	`Max_features` = 7	0.961
	`N_estimators` = 20		`N_estimators` = 20	

3.4 Data Exploration

Data exploration is performed to gain an initial understanding of the original dataset before any classification is performed. Data exploration can provide valuable insights in classification. The datasets used in data exploration are the original preprocessed datasets. All graphs in this section are organized similarly. The *x*-axis is the device type, divided into separate bins. Bin range values are chosen to highlight significant properties in the data. The *y*-axis, expressed in powers of 10, is the count of packets that belong to each corresponding bin.

3.4.1 BLE Data Exploration

The BLE dataset is comprised of six features: packet length, BLE link layer header length, associated packet count, radio frequency (RF) channel number, and protocol data unit (PDU) type.

Figure 27 shows the packet length feature, measured in bytes, across the BLE device types. The figure is organized into four bins: 20 to 40 bytes, 40 to 60 bytes, 60 to 80 bytes, and 80 to 100 bytes. One immediate insight can be observed: if a BLE packet length is greater than 80 bytes, it must be a lock device because it is the only device with packet lengths greater than 80 bytes. No other significant insights are easily discernable.

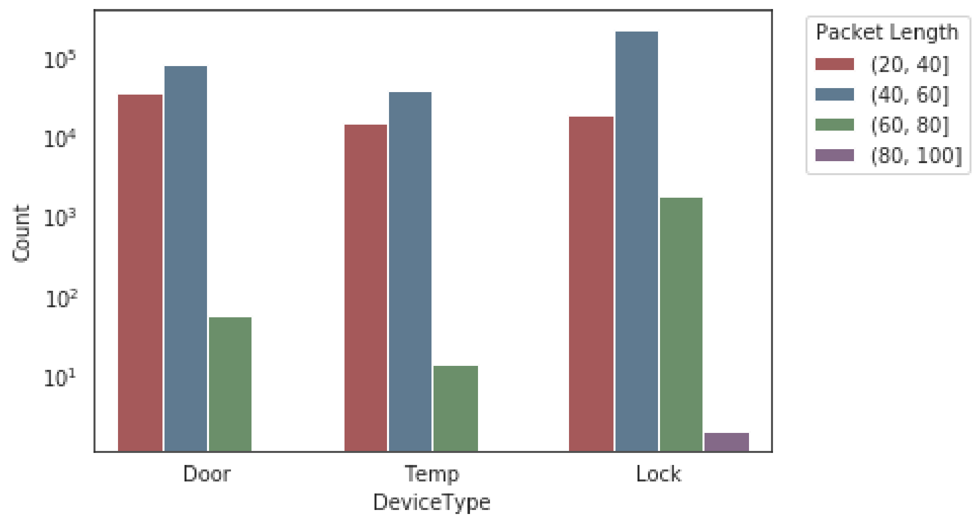


Figure 27. BLE Packet Length

Figure 28 shows the BLE link layer header length feature, measured in bytes, across the BLE device types. The figure is organized into three bins: 3 bytes to 100 bytes, 100 bytes to 150 bytes, and 150 bytes to 250 bytes. BLE link layer header length appears to be equally distributed across device types. No significant insights are easily discernable.

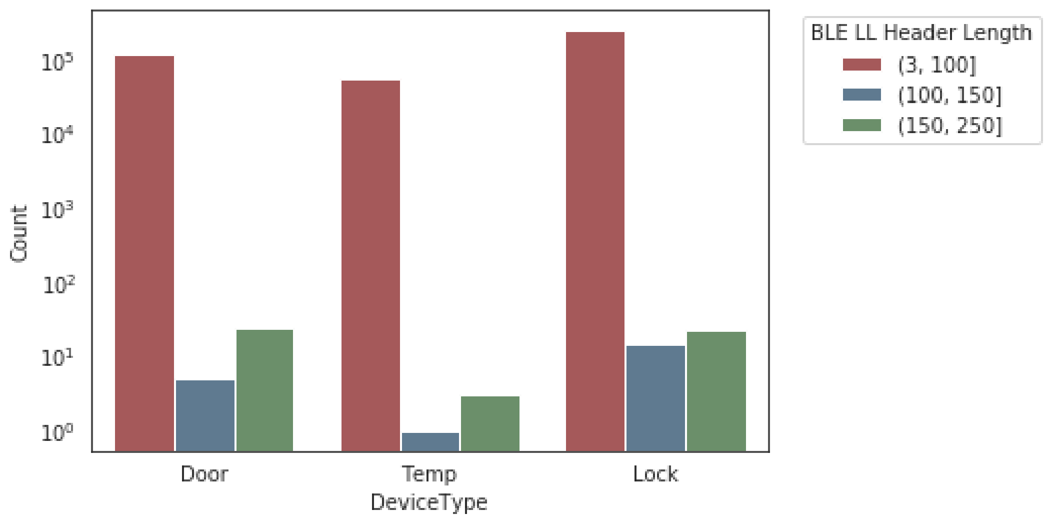


Figure 28. BLE Link Layer Header Length

Figure 29 shows the associated packet count feature, measured in packets per second, across the BLE device types, and is organized into three bins: 0 to 20, 20 to 40, and 40 to 60. BLE associated packet count appears to have some observable patterns. If a packet has an associated packet count of over 40 packets/sec, it must be a door sensor device. Furthermore, associated packet counts of 20 to 40 packets/sec are heavily associated with lock devices.

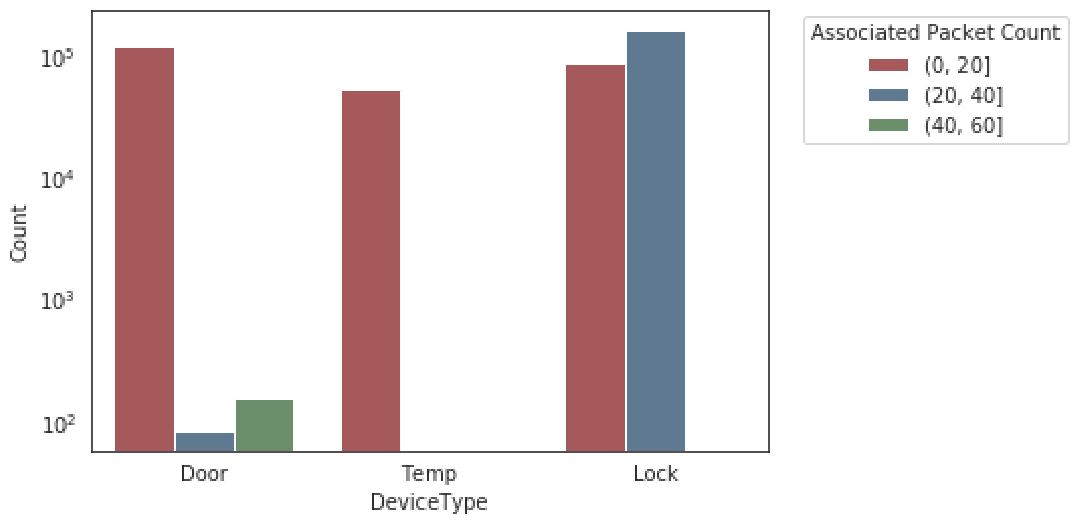


Figure 29. BLE Associated Packet Count

Figure 30 shows the RF channel number feature across the BLE device types, and is composed of three categories: channel 0, channel 12, and channel 39. The BLE RF channel feature appears to be equally distributed across device types. No significant insights are easily discernable.

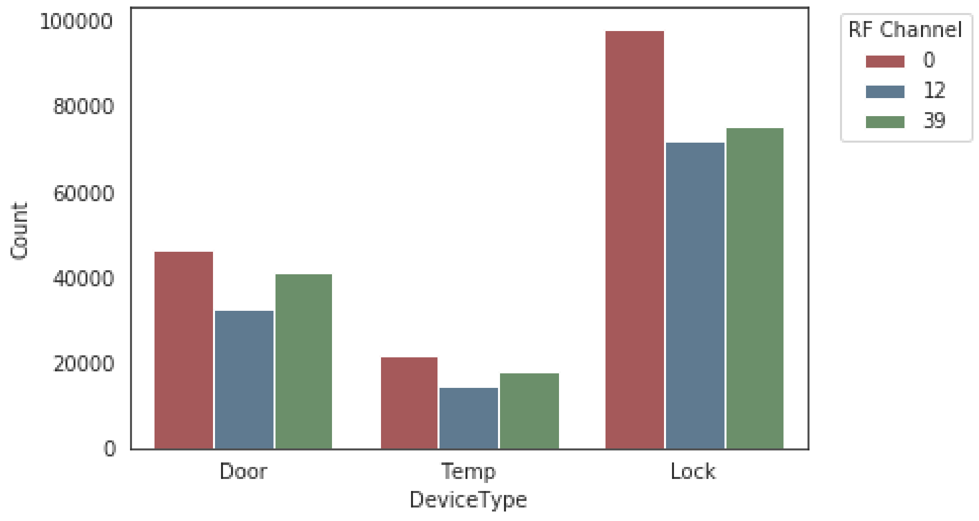


Figure 30. BLE RF Channels

Figure 31 shows the PDU type feature across the BLE device types and is composed of seven categories (see section 2.3.2). BLE PDU type appears to have one significant pattern. Three PDU types, scan requests (SCAN_REQ), advertising direct indications (ADV_DIRECT_IND), and connection requests (CONNECT_REQ) are only used by door devices. Temperature sensor and lock devices share PDU types and are similar in distribution.

To summarize BLE data exploration, the packet length, associated packet count, and PDU type features are observed to have clear classification value. If the packet length is greater than 80 bytes, the packet belongs to a lock device. If the associated packet count is greater than 40, the packet belongs to a door sensor device. If a packet uses one of three certain PDU types, the packet belongs to a door sensor. The BLE link layer header length and RF channel number features appear to be similarly distributed across the three device types.

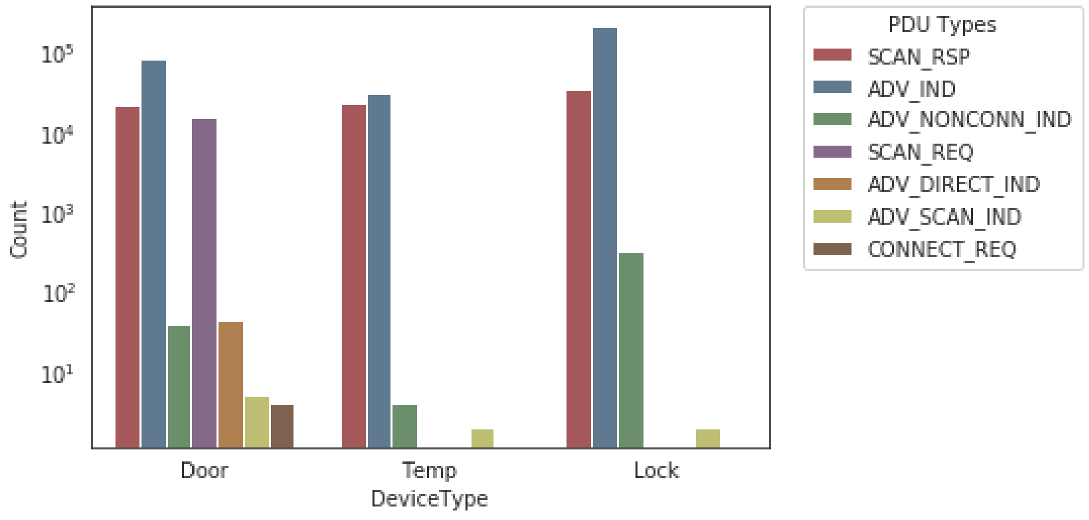


Figure 31. BLE PDU Types

3.4.2 Wi-Fi Data Exploration

The Wi-Fi dataset is comprised of four features: packet length, vendor, associated packet count, and packet subtype.

Figure 32 shows the packet length feature, measured in bytes, across the Wi-Fi device types. The figure is organized into three bins: 25 to 100 bytes, 100 to 500 bytes, and 500 to 1550 bytes. It can be observed that only camera devices have packets over 500 bytes. Additionally, plug and camera devices tends to use smaller packets, while bulb devices tend to use larger packets.

Figure 33 shows the vendor feature across the Wi-Fi device types, and is composed of six categories (see Table 5). The Wi-Fi vendor feature appears to have observable traits that can be used in classification. TP-link is the only vendor that produces all three device types. Belkin produces both plug and camera devices, but not bulbs. Lifi and Dropcam both only produce one device type each. The vendor feature may prove to be a powerful feature that can be leveraged by the classifiers.

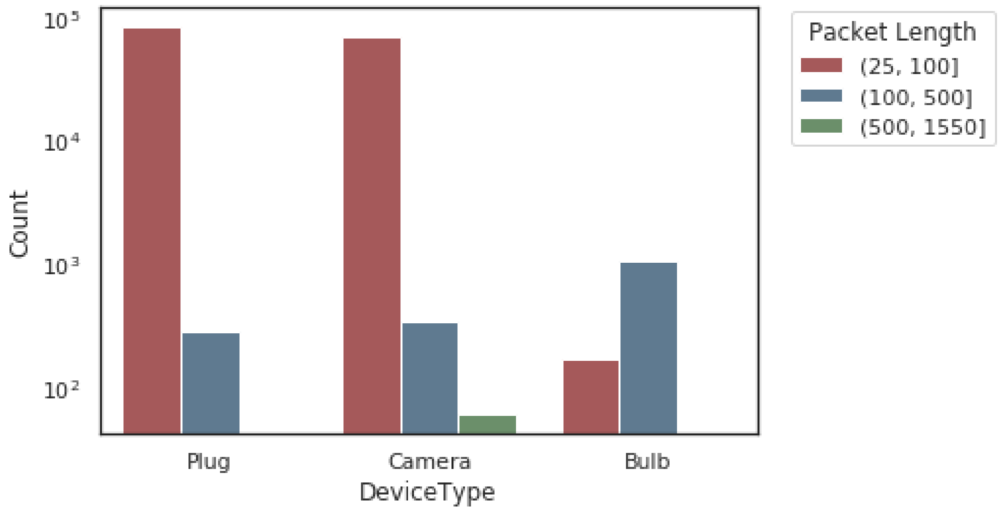


Figure 32. Wi-Fi Packet Length

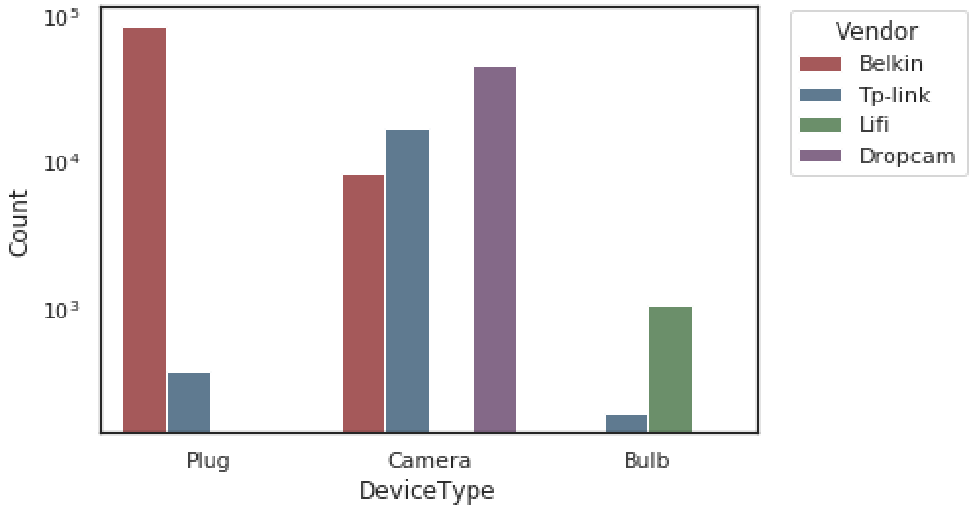


Figure 33. Wi-Fi Vendors

Figure 34 shows the associated packet count length feature, measured in packets per second, across the Wi-Fi device types, and is organized into three bins: 0 to 30 packets/sec, 30 to 100 packets/sec, and 100 to 200 packets/sec. Wi-Fi associated packet count appears to have some observable patterns. Only bulb devices have packets with an associated packet count over 100 packets/sec. Furthermore, between plug and camera devices, only camera devices have associated packet count of 30 to 100 packets/sec. Similar to the vendor feature, the Wi-Fi associated packet count appears to be a feature with classifying potential.

Figure 35 shows the packet subtype feature across the Wi-Fi device types, and is composed of three categories: data, quality-of-service (QoS) data, and QoS null. One clear observation can be made: bulbs do not use QoS null packets. Plug and camera devices appear to share similar packet subtype distributions.

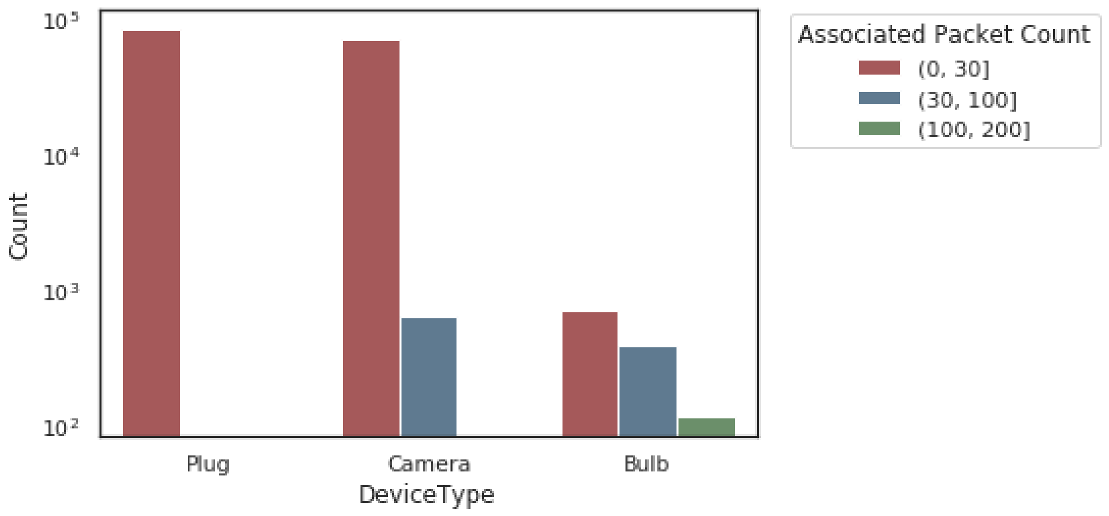


Figure 34. Wi-Fi Associated Packet Count

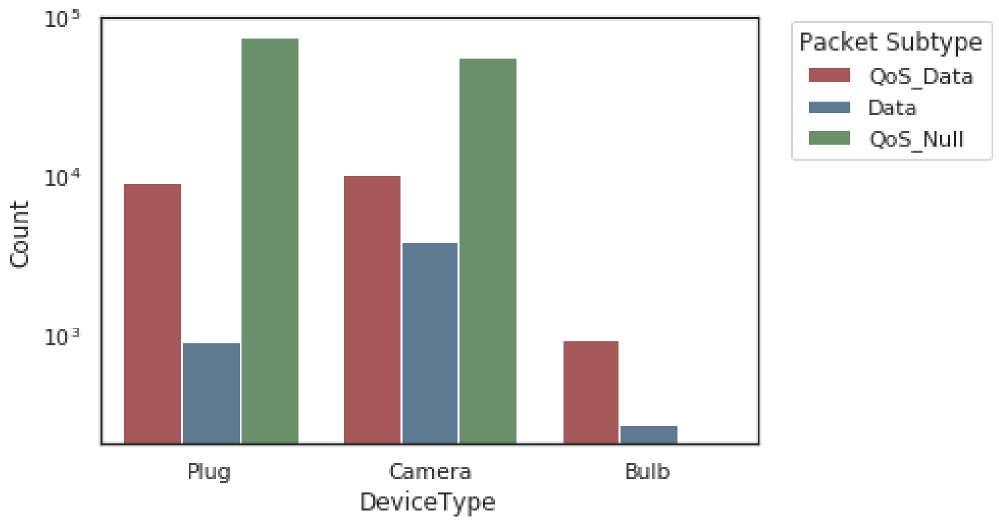


Figure 35. Wi-Fi Packet Subtype

To summarize Wi-Fi data exploration, all features are observed to have classification value. If the packet length is greater than 500 bytes, the packet belongs to a camera device. If the vendor is Belkin, the packet belongs to either a plug or camera device. If the vendor is Lifi, the packet belongs to a bulb device. If the vendor is Dropcam, the packet belongs to a camera device. If the associated packet count is greater than 100, the packet belongs to a bulb device. Lastly, if the packet is of the QoS null subtype, the packet does *not* belong to a bulb device.

3.5 Design Summary

This chapter describes each component of SHE and DCP. Their design enables the use of authentic smart device wireless traffic in the classification of IoT devices. This chapter also explores the features of the datasets, looking for patterns that may be useful in gaining an understanding of the classification task.

IV. Methodology

4.1 Problem/Objective

The goal of this experiment is to evaluate the effectiveness of machine learning classifiers at identifying IoT devices using wirelessly collected traffic. The experiment discussed in this section evaluates these classifiers using a set of performance metrics. The experiment attempts to complete three objectives:

1. Determine the ability of a classifier to classify a given packet to a device type.
2. Measure the performance of a classifier in identifying the device types of a smart home environment.
3. Determine which features are most useful for classification.

4.2 System under Test

The system under test (SUT) diagram is shown in Figure 36. The system under test are the tuned classifiers. Within the SUT are the three components under test: the KNN, LDA and RF classifiers. The response variables and metrics used to evaluate the classifiers' performance include the confusion matrix, feature importance score, Matthews correlation coefficient, mean precision, and mean recall, and are described in Sections 4.3 and 4.4. Controlled variables are discussed in Section 4.5. The collected BLE and Wi-Fi traffic are considered uncontrolled and are analyzed in Section 4.6. The parameters are variables that remain unchanged throughout the experiment and are studied in Section 4.7. The experimental factors are the variables that change between experimental trials, and are discussed in Section 4.8.

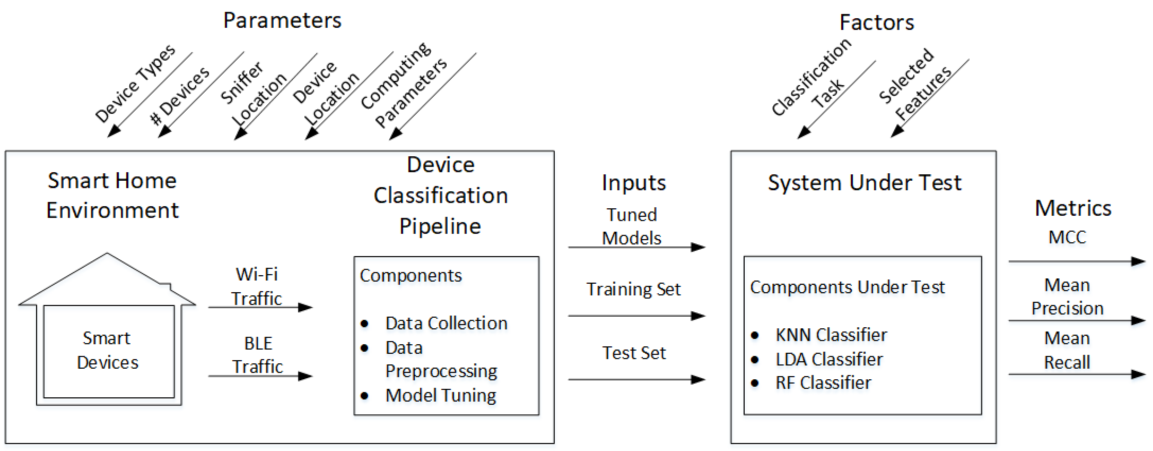


Figure 36. System under test diagram

4.2.1 Assumptions

The following assumptions are made throughout the design and execution of the experiments of the SUT:

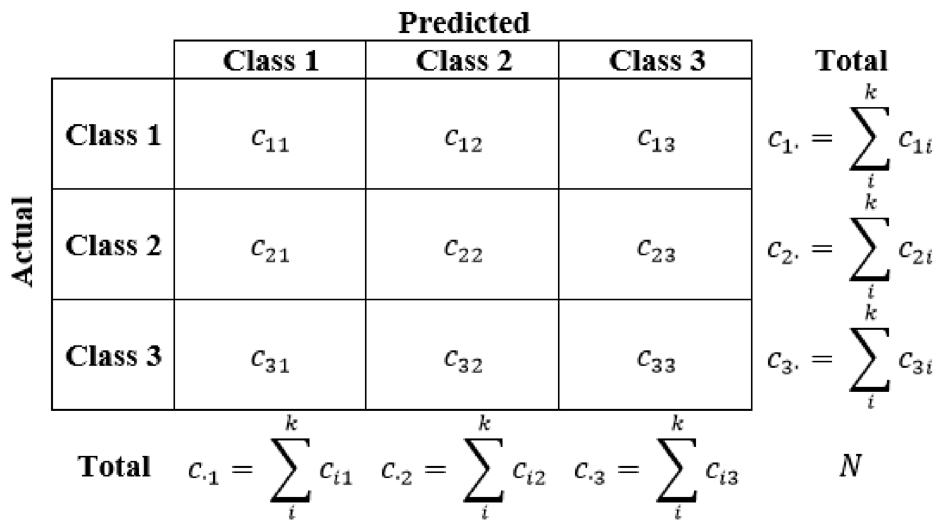
1. The activities done by the devices in the smart home characterize a real-life smart home environment.
2. Each device in the test setup is unique, and the sniffer is not misrepresenting any collected data.
3. The sniffer has necessary network knowledge to collect wireless traffic, to include access point MAC address, access point channel, BLE device MAC addresses, and BLE device names.
4. Devices in a test setup are not interfering with each other in any significant manner.
5. Outside noise is negligible.
6. Devices do not apply additional security mechanisms.

4.3 Response Variables

Response variables are the direct outputs of the experiment, and are used to calculate the metrics. The objectives of the experiment motivate the response variables selected to assess the performance of the classifiers. The confusion matrix response variables are produced by each combination of device type and algorithm, for a total of nine configurations per wireless protocol. The feature importance score is produced by the random forest classifier, and is reported once per trial. All response variables are numerical. Table 12 provides a summary of each response variable.

- **Objective 1**: Determine the ability of each classifier to classify a given packet to a device type. The variables listed below are collectively referred to as the confusion matrix response variables because they are derived from the confusion matrix.
    - **True Positives (TP)**: The TP response variable measures the number of true positives, or packets that are correctly classified to a device type.
    - **False Positives (FP)**: The FP response variable measures the number of false positives, or packets that are incorrectly classified as a device type.
    - **False Negatives (FN):** The FN response variable measures the number of false negatives, or packets that are incorrectly *not* classified as a device type.

The TP, FP, and FN variables are presented using the confusion matrix. The confusion matrix is a square matrix with an equal number of row and columns, where a row of the matrix represents the instances in an actual class, and a column represents the instance in a predicted class. Figure 37 shows a $k\ x\ k$ confusion matrix, where k is the number of classes (in this case, three), N is the total number of instances, and c_{ij} is the number of instances with a true label of i classified into class j. The total predicted count for class x is given by $c_{\cdot x}$, while the total actual count for class x is given by summing the values along the column $c_{x\cdot}$. The TP count for class x, TP_x, is given by the value at c_{xx}. It can be observed that all class TP counts are found along the diagonal of the confusion matrix. The FP count for class x, FP_x, is calculated by $c_{\cdot x} - TP_x$. The FN count for class x, FN_x, is calculated by $c_{x\cdot} - TP_x$.



		Predicted			
		Class 1	Class 2	Class 3	Total
Actual	Class 1	c_{11}	c_{12}	c_{13}	$c_{1\cdot} = \sum_i^k c_{1i}$
	Class 2	c_{21}	c_{22}	c_{23}	$c_{2\cdot} = \sum_i^k c_{2i}$
	Class 3	c_{31}	c_{32}	c_{33}	$c_{3\cdot} = \sum_i^k c_{3i}$
	Total	$c_{\cdot 1} = \sum_i^k c_{i1}$	$c_{\cdot 2} = \sum_i^k c_{i2}$	$c_{\cdot 3} = \sum_i^k c_{i3}$	N

Figure 37. A k x k Confusion Matrix ($k = 3$)

- **Objective 2:** Measure the importance of features in classification using the random forests classifier.
    - **Feature Importance Score:** The feature importance score measures how significant a feature is in the classification model. The random forest classifier, as implemented by `scikit-learn`, reports the feature importance with values ranging from 0 to 1, where higher values correspond to higher feature importance. The sum of feature importance scores is 1.

Table 12. Response Variables

Name	Source/Formula	Definition
TP	Count for a class found at the diagonal of the confusion matrix	Correctly classified packet
FP	$\sum Class\ Predicted - TP$	Packets that were incorrectly classified as a particular class
FN	$\sum Class\ Actual - TP$	Packets that were incorrectly *not* classified as a particular class
Feature Importance	Derived from random forests classifiers	Measure of a given feature's usefulness in classification

4.4 Performance Metrics

The performance of the classifiers is measured using three metrics: the Matthews correlation coefficient, mean recall, and mean precision. These performance metrics are calculated using functions from the `scikit-learn metrics` module.

4.4.1 Matthews Correlation Coefficient (MCC)

The MCC metric provides a measure of a classifier's overall classification performance using the confusion matrix. Traditionally used as a binary classification metric, the MCC has been successfully extended to multi-classification tasks [38] [39]. Functionally, the MCC metric is chosen over the popularly used accuracy metric because accuracy misrepresents classifier performance in imbalanced datasets. If instances from the majority class significantly outnumber minority classes, a classifier can potentially report high accuracy scores simply by selecting the majority class for all instances. MCC does not suffer from this type of misrepresentation. MCC not only accounts for class imbalances, but also provides a convenient range of values to evaluate classifier performance. The MCC metric exists in the range $[-1, 1]$, where -1 represents perfect misclassification and 1 represents perfect classification. A MCC value of 0 is calculated for confusion matrices that performed random classification. The MCC metric is calculated in this research using the `sklearn.metrics matthews_corrcoef` function. The MCC can be calculated formally using

$$MCC = \frac{N\,Tr(C) - \sum_{kl} C_k C_l}{\sqrt{N^2 - \sum_{kl} C_k (C^T)_l}\sqrt{N^2 - \sum_{kl} (C^T)_k C_l}} \qquad (2)$$

where N is the total number of instances in the confusion matrix C, $Tr(C)$ is the trace or sum of the confusion matrix diagonal, C_k is the kth row of C, C_l is the lth column of C, and C^T is C transposed [39].

4.4.2 Mean Precision

The mean precision metric provides a measure of the overall positive predictive power of a classifier by calculating the mean precision metric over all classes. Precision is calculated using

$$Precision = \frac{TP}{TP+FP} \times 100 \qquad (3)$$

where TP represents the true positive count and FP represents the false positive count. A classifier with high precision provides high confidence that a positive prediction is a correct prediction. The mean precision metric is calculated in this research using the `sklearn.metrics precision_score` function with macro-averaging, while individual class precision metrics are calculated using the `precision_score` function without averaging.

4.4.3 Mean Recall

The mean recall metric provides a measure of the overall success of a classifier as tested on an imbalanced dataset by calculating the mean recall over all classes. The recall metric measures the rate at which a classifier correctly identifies a given device type. Recall is calculated using

$$Recall = \frac{TP}{TP+FN} \times 100 \tag{4}$$

where TP represents the true positive count and FN represents the false negative count. A classifier with high recall rarely overlooks an actual positive, giving high confidence that the classifier can identify a complete set of actual positives. Mean recall is also known as balanced accuracy, and recall is also known as sensitivity or true positive rate. The mean recall metric is calculated in this research using the `sklearn.metrics` `balanced_accuracy_score` function, while individual class recall metrics are calculated using the `recall_score` function.

4.4.4 High Performance

Classifiers are evaluated on their performance using the MCC, mean recall, and mean precision metrics. Table 13 provides a summary of the performance metrics, including the units, accepted range, and performance threshold values of each metric. Performance threshold values specify the values necessary for a metric to be considered high or low performance, and are derived from hypothesized classifier performance. A classifier that performs worse than random chance is considered a low performance classifier. As such, the low performance threshold values reflect those expected from a random chance classifier. The chosen high performance threshold values indicate performance that is significantly better than random chance, and would reflect a classifier with high classification performance.

Table 13. Performance Metrics

Metric	Units	Range	High Performance Threshold	Low Performance Threshold
MCC	--	-1 to 1	≥ 0.50	< 0.0
Mean Recall	%	0 to 100	≥ 75%	< 50%
Mean Precision	%	0 to 100	≥ 75%	< 50%

4.5 Control Variables

The following variables are controlled in each trial to provide the classifiers sufficient information.

- **Classifier Hyperparameters**: Each classifier has certain hyperparameters that may be adjusted to improve performance. A range of tuning hyperparameters are evaluated by DCP to produce the best-performing classifiers. Tuned classifiers are used for the multiclass full-featured classification task, but not used for all other trials.
- **Source of Motion:** Various devices in the smart home environment rely on motion detection to initiate actions. A controlled source of motion is needed to consistently initiate device actions. An Arduino Uno microcontroller is programmed to activate a stepper motor every fifteen minutes. The stepper motor rotates an eight-inch rod with a two-inch-wide paper flap attached to it. The entire apparatus resembles a one-armed windmill, and its appearance and location in SHE is shown in Figure 38.

4.6 Uncontrolled Variables

The experiment assumes a realistic smart home environment. Therefore, the collected BLE and Wi-Fi traffic cannot be controlled. The use of commercial devices and an open wireless environment presents potential issues with outside noise and accompanying unintended effects. Nevertheless, no significant noise is experienced throughout data collection and experimentation.

4.7 Parameters

The experiment is performed under a number of parameters to replicate a realistic smart home environment. Minimizing external factors is critical in achieving this. The parameters in this experiment include:

- **Location of Devices**: Each device is placed in the same location in the testing environment at the time of sniffing.
- **Location of Sniffers**: Each sniffer is placed in the same location in the testing environment at the time of sniffing.
- **Number of Devices**: The number of devices does not change.
- **Type of Devices**: The types of devices present in each test setup does not change.
- **Sniffing Equipment**: The same sniffing equipment is used to collect wireless packets. This includes antennas, wireless dongles, and software tools. This limits the amount of instrumentation noise that could be introduced by the equipment.
- **Computing Environment**: The computing environment that performs the classification is kept constant. This includes operating system, system resources, programming languages, and hardware.

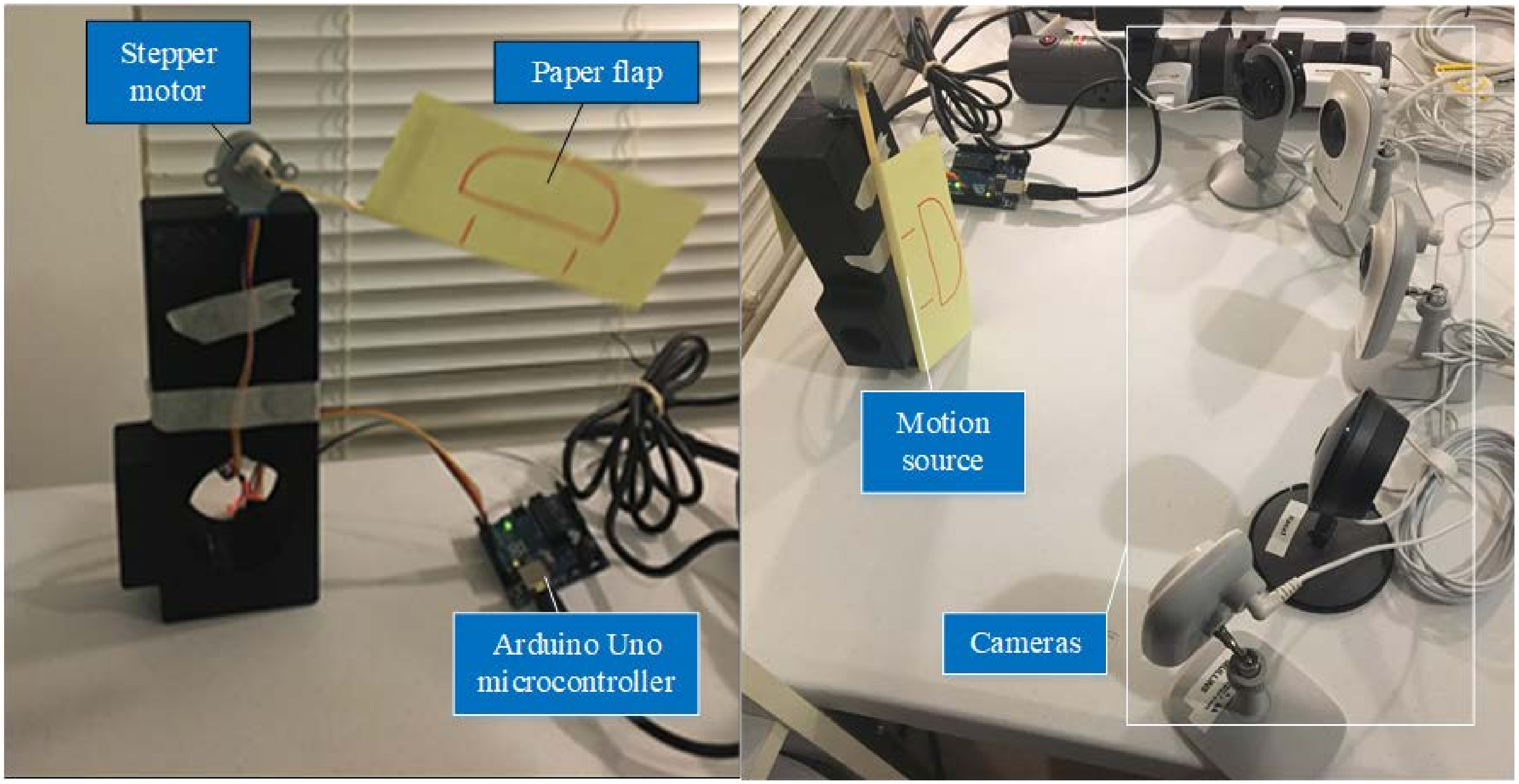


Figure 38. Motion Source Appearance (left) and Location in SHE (right)

4.8 Factors

The experiment factors are the changes done between experimental trials. The experiment uses two factors in evaluating the classifiers' performance: classification task and selected features.

The classification task factor refers to the number of response classes (i.e., device types) being classified. Two classification tasks are used in the experiment: multiclass and binary. Multiclass classification is performed to classify all given device types in the wireless protocol, while binary classification only classifies between two device types. Multiclass classification is the primary focus of the experiment as it provides the most realistic classification value. Binary classification is completed for classification analysis purposes, especially when classifiers are having difficulties distinguishing between two given device types.

The selected features factor refers to the number of features employed in the classification process (see Table 6 and Table 9 for the full list of Wi-Fi and BLE features, respectively). Two feature configurations are used in this experiment: full-featured and best-features. Full-featured multiclass classification uses all available features for the given protocol to classify devices into their respective device types. The tuned models with optimal hyperparameters are used in this task because the tuning process applies all features. The best-features multiclass classification uses only the *k* best-performing features from full-featured multiclass classification, where *k* is an integer no greater than the total of all available features. The use of a small subset of relevant features frequently results in improved performance because of the removal of noisy and redundant features that confuse the classifiers [40]. In this experiment, *k* is set to 3 in order to standardize the evaluation of both BLE and Wi-Fi classifiers.

4.9 Experimental Design

The experiment proceeds in three stages: data collection, model tuning, and model testing. The complete list of steps needed to perform the experiment is provided in Appendix C. SHE devices are set up and allowed to reach a steady state for one day. Data collection for the BLE and Wi-Fi devices then occurs over a period of three days. Wireless packet sniffing is performed for eight hours for each data collection day. Once data collection is complete, two Jupyter notebooks running DCP, one each for BLE and Wi-Fi, are used to clean and process data, extract features, and perform model tuning. Once model tuning is complete, the classifiers are updated to use the best-performing hyperparameter values. The test dataset is adjusted using down-sampling to maintain a uniform distribution of packets across all device types. Random down-sampling selects data points at random and removes them from the dataset, ensuring that all device types have an equal number of packets in the test dataset.

Model testing is done using a Jupyter notebook written in Python, one each for BLE and Wi-Fi (see Appendix D and E). The models are evaluated on two classification tasks: full-featured multiclass classification and best-features classification. The full set of features is used to evaluate the overall classification performance of the machine learning classifiers, while best-features classification is used to check if performance can be improved by removing irrelevant features. Using the random forests feature importance scores, the three most important features are selected and used to retrain and retest the

KNN, RF, and LDA classifiers. By reducing the number of features, feature confusion can be minimized. Response variables and performance metrics are extracted, and are stored in csv files. Evaluating the performance of the classifiers on the test set is the primary focus of Chapter 5.

4.10 Methodology Summary

This chapter provides the experimentation methodology used to evaluate the performance of the classifiers through the confusion matrix and feature importance response variables, and the Matthews correlation coefficient, mean recall, and mean precision metrics.

V. Results and Analysis

5.1 Overview

This chapter provides the results obtained from the experimentation described in Chapter 4. Classifier performance is evaluated using MCC, mean recall, and mean precision performance metrics. When needed, further analysis is completed using the confusion matrix results and feature importance scores. Classifiers are evaluated on their performance in the full-featured multiclass classification task where all available features are used in classification, and then in the best-features multiclass classification task where only the most relevant features are used, as determined by feature importance scores. Lastly, classification analysis is completed to discuss notable behavior by the classifiers. Sections 5.2 and 5.3 examine the classifiers' performance in the BLE and Wi-Fi datasets respectively.

5.2 BLE Classifier Performance

The classification task for BLE classifiers is categorizing a BLE packets into one of three device types: door sensor, lock, or temperature sensor. Results are calculated using the Jupyter notebook in Appendix D.

5.2.1 BLE Full-featured Classification

This section examines the performance of the classifiers when the full set of available features are used, otherwise known as full-featured classification. Figure 39 and Table 14 provide the confusion matrices and overall performance metrics of the BLE classifiers for full-featured classification. The MCC metric provides a measure of a classifier's overall classification performance using the confusion matrix, where -1 indicates perfect misclassification and 1 indicates perfect classification. The KNN classifier achieves the best overall performance with an MCC of 0.55, a mean precision of 54%, and a mean recall of 64%. The RF classifier attains an MCC of 0.00, indicating that its performance is comparable to random chance, while the LDA classifier achieves a negative MCC value, revealing its poor value as a classifier. Out of all classifiers, only the KNN classifier's MCC metric succeeds in exceeding the high performance threshold.

Table 14. Classifier Performance in BLE Full-Featured Classification

	MCC	**Mean Precision**	**Mean Recall**
KNN	0.55	54.1%	64.0%
RF	0.00	17.1%	33.5%
LDA	-0.61	0.0%	0.0%

The precision metric measures the positive predictive power of the classifiers. A classifier with high precision provides high confidence that a positive prediction is a correct prediction. Table 15 provides the precision scores for all BLE device types on the full-featured classification task. While KNN achieves a perfect precision score on lock devices, it performs poorly on the door sensor and temperature sensor devices. This mixed performance explains KNN's low mean precision score. As expected from their low mean precision scores, the RF and LDA classifiers achieve poor individual precision scores, with the LDA classifier managing to score 0.0% precision for all device types.

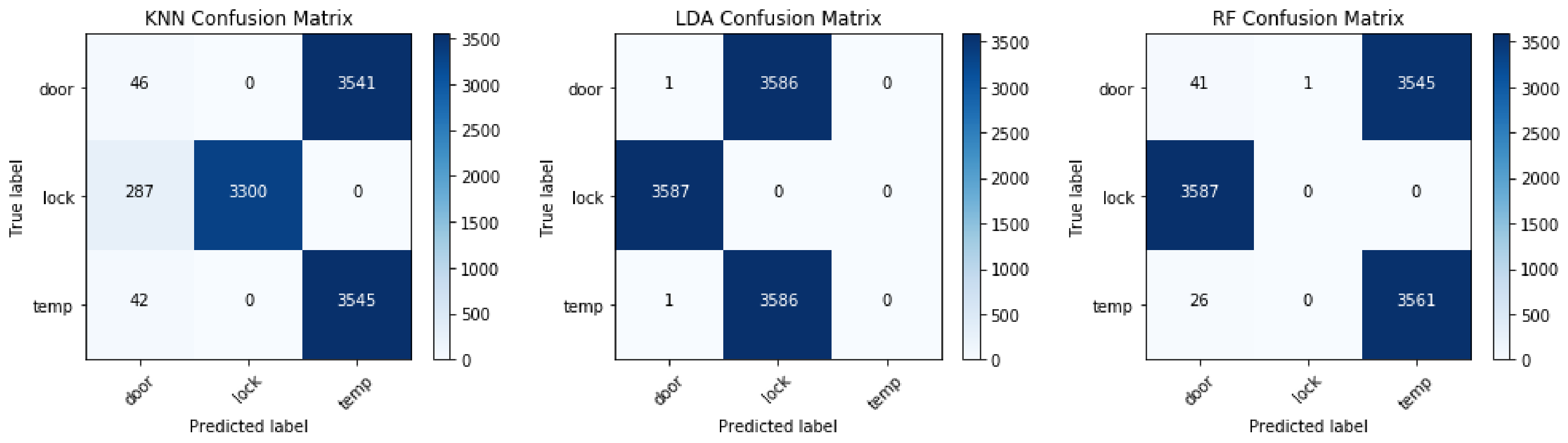


Figure 39. Confusion Matrices from BLE Full-Featured Classification

Table 15. Device Type Precision in BLE Full-Featured Classification

	Door Sensor	Lock	Temp Sensor
KNN	12.3%	100.0%	50.0%
RF	1.1%	0.0%	50.1%
LDA	0.0%	0.0%	0.0%
Mean	4.5%	33.3%	33.4%

The recall performance metric measures the completeness of the classifiers. A classifier with high recall provides high confidence that a complete set of actual positives is found. Table 16 provides the recall scores for all BLE device types on the full-featured classification task. KNN again achieves the best performance, with excellent scores on two device types. Given their recall scores, high confidence can be placed that KNN can successfully identify the majority of lock and temperature sensor devices. However, KNN's performance is severely diminished by the poor recall score of 1.3% on the door devices. The RF classifier manages an outstanding 99.3% recall score on the temperature sensor devices, but scored poorly on the other two device types. The LDA classifier continues its low performance with a 0.0% recall score for all device types.

Table 16. Classifier Recall in BLE Full-Featured Classification

	Door Sensor	Lock	Temp Sensor
KNN	1.3%	92.0%	98.8%
RF	1.1%	0.0%	99.3%
LDA	0.0%	0.0%	0.0%
Mean	0.8%	30.7%	66.0%

The feature importance score measures the significance of a feature in classification. Feature importance scores range from 0 to 1.0, where higher values indicate more important features. Feature importance is reported from the random forest classifier. It is important to note that the random forests classifier is only performing at chance, with an MCC value of 0.00. Therefore it cannot be assumed that the reported feature importance scores are meaningful. Nevertheless, their scores are reported here for completeness. Table 17 provides the feature importance scores obtained from full-featured classification. Out of the twelve available features, seven report nonzero feature importance scores. Packet length, BLE link layer header length, and associated packet count are observed as the three most important features, and combined account for over two-thirds of the feature importance. This indicates that a majority of the random forest classifier's decision-making rely on these three features. Five features report a feature importance score of 0.000, signifying that the random forest classifier does not benefit from these features. It is interesting to note that the top three features are numerical features and the features with scores of 0.000 are categorical features. A possible explanation is that because categorical features are one-hot encoded, the value of their information is spread across multiple individual features. For example, the PDU type feature is one-hot encoded to six binary features. Therefore, the feature importance of the PDU type feature is divided into six separate feature importance scores. As such, numerical features tend to report high importance scores because categorical scores are spread out.

Table 17. Feature Importance in BLE Full-Featured Classification

	Feature	Score
1	Packet Length	0.327
2	BLE LL Length	0.212
3	Associated Packet Count	0.193
4	SCAN_RSP PDU Type	0.135
5	ADV_IND PDU Type	0.106
6	SCAN_REQ PDU Type	0.026
7	Channel 39	0.001
8	ADV_DIRECT_IND PDU Type	0.000
9	ADV_NONCONN_IND PDU Type	0.000
10	CONNECT_REQ PDU Type	0.000
11	Channel 0	0.000
12	Channel 12	0.000

5.2.2 BLE Best-Features Classification

Best-features classification uses a small subset of features from the original feature set. The use of a small subset of relevant features frequently results in improved performance because of the removal of noisy and redundant features that confuse the classifiers [40]. Best-features classification is performed after full-featured classification to take advantage of the feature importance scores obtained from full-featured classification.

The three best features reported by BLE full-featured classification are the packet length, BLE link layer header length, and associated packet count features (see Table 17). The classifiers are retrained and retested using a dataset containing only these three features to execute best-features classification. Table 18 provides the overall performance metrics of the BLE classifiers in best-features classification. As hypothesized, the performance of the classifiers improves in best-features classification. The KNN classifier experiences minor improvement across all metrics, with a 0.02 points added to MCC, 3.7% added to mean precision, and 1.2% added to mean recall. The RF classifier and LDA classifier both experienced significant improvements, with the RF classifier adding 0.57 points to its MCC, 36.2% added to mean precision, and 31.6% added to mean recall, and the LDA classifier adding 0.61 points to its MCC, 17.3% points to its mean precision, and 33.4% points to its mean recall.

Table 18. Classifier Performance in BLE Best-3 Feature Classification

	MCC	Mean Precision	Mean Recall
KNN	0.57	57.8%	65.2%
RF	0.57	53.3%	65.1%
LDA	0.00	17.3%	33.4%

5.2.3 BLE Classification Analysis

Classification analysis is performed to examine notable observations and offer explanations that caused them. The key classification observation in the BLE dataset is the misclassification of door sensors as temperature sensors. Figure 39 provides the confusion matrix results from BLE full-featured classification. As described in Section 4.3, the x-axis shows the predicted labels and the y-axis shows the true labels. Cells with darker colors indicate a higher count of instances. It is observed that both KNN and RF classifiers heavily misclassify door sensors as temperature sensors. This misclassification directly contributes to the classifiers' poor recall performances. To understand this misclassification, a classification trial is prepared with only door sensors and temperature sensors. The classification task is adjusted to a binary classification task between the two device types. Table 19 and Figure 40 provide the respective overall performance metrics and confusion matrix for this binary classification. The overall performance metrics confirm that the classifiers cannot reliably distinguish between door sensors and temperature sensors. All classifiers achieve MCC scores close to zero, indicating that their value as classifiers resembles that of random guessing. Similarly, the mean recall scores are approximately 50%, signifying that the classifiers are only able to consistently discern a complete set of device types half the time. The confusion matrix results imply that because the classifiers are not capable of finding a meaningful difference between the device types, the classifiers are reduced to a naïve strategy of categorizing the vast majority of instances into a single device type. Considering the top three features, it becomes clear that there are no clear differences between door sensors and temperature sensors. Door sensors and temperature sensors have nearly identical distributions for the packet length, BLE LL length, and associated packet count features, explaining why the classifiers experienced difficulty in separating the two device types (see Appendix E).

Table 19. Classifier Performance in BLE Door Sensors vs Temperature Sensors

	MCC	**Mean Precision**	**Mean Recall**
KNN	-0.01	25.0%	50.0%
RF	0.02	55.2%	50.2%
LDA	0.03	59.4%	50.2%

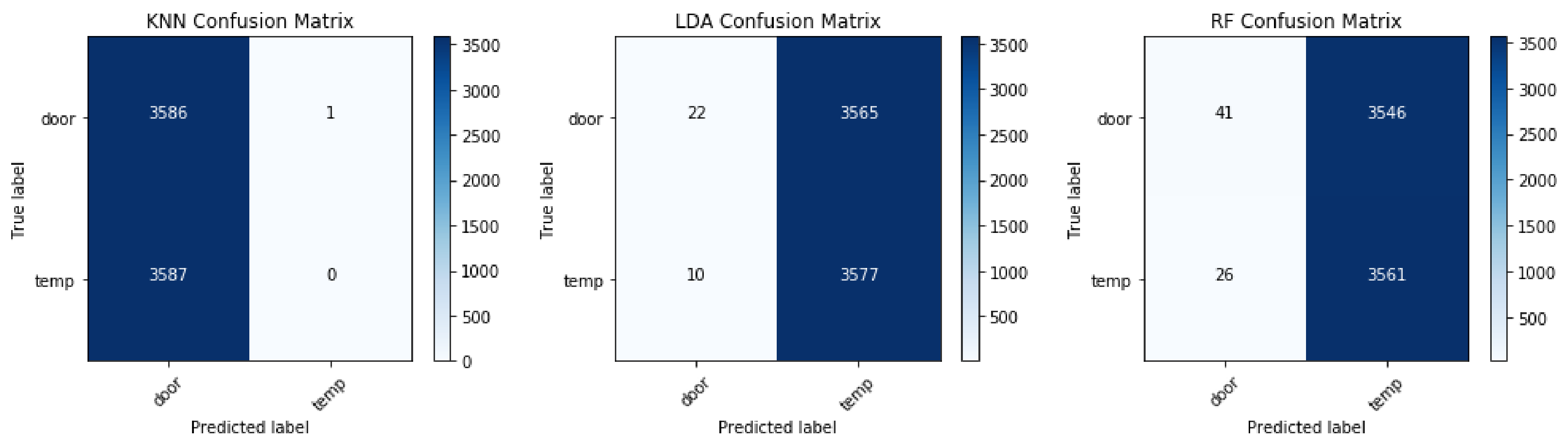


Figure 40. Confusion Matrices from BLE Door Sensors vs Temperature Sensors

5.3 Wi-Fi Classifier Performance

The classification task for Wi-Fi classifiers is categorizing a given Wi-Fi packet into one of three device types: bulb, camera, or smart plug. Results are calculated using the Jupyter notebook in Appendix E.

5.3.1 Wi-Fi Full-featured Classification

Figure 41 and Table 20 provide the confusion matrices and the overall performance metrics of the Wi-Fi classifiers in full-featured classification. Out of the three classifiers, two achieve MCC values above the high-performance threshold, indicating they have noteworthy classification value. The KNN classifier attains the best overall performance with all metrics exceeding the high-performance threshold as mentioned in Section 4.4.4. The LDA classifier succeeds with a high-performance MCC, but fails to achieve high performance in its mean precision and mean recall scores. The RF classifier fails to achieve any high performance metrics, but nevertheless manages respectable performance levels.

Table 20. Classifier Performance in Wi-Fi Full-Featured Classification

	MCC	**Mean Precision**	**Mean Recall**
KNN	0.71	81.1%	80.4%
RF	0.74	85.4%	80.8%
LDA	0.84	89.7%	89.2%

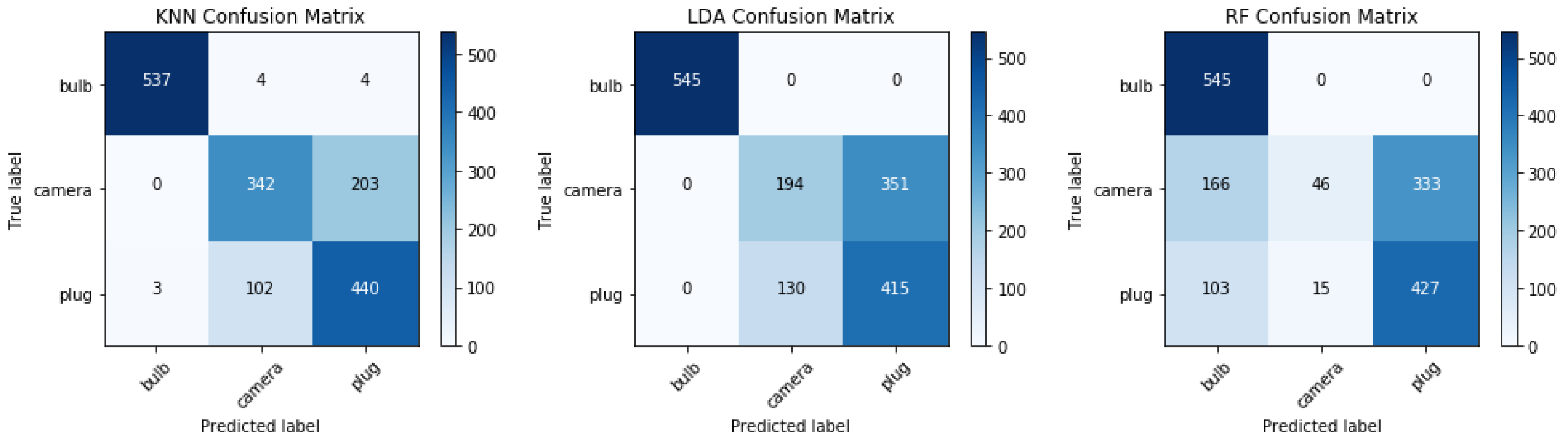


Figure 41. Confusion Matrices from Wi-Fi Full-Featured Classification

Table 21 and Table 22 provide the precision and recall scores for all Wi-Fi device types respectively. Out of the three device types, the KNN and RF classifiers achieve high precision scores on two device types, while the LDA classifier achieves high precision scores on all three device types. Comparing between device types, the classifiers achieves the highest mean precision on the bulb devices, suggesting the classifiers are able to positively predict the bulb devices at a higher rate than the camera and plug devices. The recall scores confirm this idea, as all three classifiers achieve high recall scores on the bulb device type. Recall scores for all three classifiers on the plug device type also meet the high performance threshold, implying that Wi-Fi classifiers are able to successfully identify two out of the three device types.

Table 21. Device Type Precision in Wi-Fi Full-Featured Classification

	Bulb	**Camera**	**Plug**
KNN	100.0%	74.9%	68.3%
RF	67.0%	95.9%	93.4%
LDA	100.0%	79.3%	89.8%
Mean	89.0%	83.4%	83.8%

Table 22. Device Type Recall in Wi-Fi Full-Featured Classification

	Bulb	**Camera**	**Plug**
KNN	96.7%	63.1%	81.3%
RF	100.0%	64.0%	78.3%
LDA	100.0%	91.4%	76.1%
Mean	98.9%	72.8%	78.6%

Table 23 provides the feature importance scores obtained from Wi-Fi full-featured classification. Out of the ten available features, eight report nonzero feature importance scores. The Belkin vendor, associated packet count, and Dropcam vendor features are observed as the three most important features, and together account for over 90% of the feature importance. This indicates that a majority of the random forest classifier's decision-making depended on these three features. One feature (Tp-Link) reports a feature importance score of 0.000, signifying that the random forest classifier does not benefit from this feature.

Table 23. Feature Importance in Wi-Fi Full-Featured Classification

	Feature	**Score**
1	Belkin Vendor	0.531
2	Associated Packet Count	0.283
3	Dropcam Vendor	0.103
4	Packet Length	0.035
5	QoS_Null Packet Subtype	0.021
6	QoS_Data Packet Subtype	0.015
7	Data Packet Subtype	0.008
8	Lifi Vendor	0.005
9	Tp-link Vendor	0.000

5.3.2 Wi-Fi Best-features Classification

Similar to BLE classification, best-features classification is performed after full-featured classification to benefit from the feature importance scores from full-featured classification. It is hypothesized that the use of a reduced subset of relevant features results in improved classification performance.

The three best features reported by Wi-Fi full-featured classification are the Dropcam vendor, Belkin vendor, and associated packet count features (see Table 23). The classifiers are retrained and retested using a dataset containing only these three features to execute best-features classification. Table 24 provides the overall performance metrics of the Wi-Fi classifiers in best-features classification. Best-features classification results in significant decline of performance for all classifiers. The KNN classifier experiences a decline in performance across all metrics, with a 0.38 point reduction in MCC, 38.3% subtracted from mean precision, and 27.8% subtracted from mean recall. The RF classifier also receives substantial decreases across all metrics, with a 0.24 point reduction in MCC, 37.0% subtracted from mean precision, and 20.9% subtracted from mean recall. Lastly, the LDA classifier receives significant losses in performance, with a 0.28 point reduction in MCC, 11.0% subtracted from mean precision, and 21.4% subtracted from mean recall.

Table 24. Classifier Performance in Wi-Fi Best-3 Feature Classification

	MCC	**Mean Precision**	**Mean Recall**
KNN	0.32	42.8%	52.5%
RF	0.49	48.4%	59.9%
LDA	0.56	78.7%	67.8%

5.3.3 Wi-Fi Classification Analysis

Wi-Fi classification analysis is performed to understand two notable observations: the misclassification of camera devices as plug devices and the reliance of the classifiers on vendor features.

The first notable observation is the misclassification of cameras as plug devices. Figure 41 provides the confusion matrix results from Wi-Fi full-featured classification. It is observed that all three classifiers misclassify cameras as plugs to varying degrees. To understand this misclassification, a classification trial is prepared with only camera and plug device types in the training and test sets. The classification task is adjusted to a binary classification between the two device types. Table 25 and Figure 42 provide the respective overall performance metrics and confusion matrix for the binary classification of cameras and plugs. The overall performance metrics report a loss of performance across all classifiers. Notably, the KNN classifier sees a 0.79 drop in its MCC score. By contrast, the RF classifier's performance experiences a lesser yet still significant decline, with the MCC dropping by 0.22 points. The confusion matrix results suggest that the KNN and LDA classifiers classify the majority of packets as plug devices, while the RF classifier successfully separates the two device types.

The random forest feature importance scores are then analyzed to understand which features the RF classifier uses to achieve this. Table 26 provides the feature importance scores in the cameras versus plugs classification. The best features are observed as the Belkin vendor, associated packet count, and Dropcam vendor features. Interestingly, these are the same best features found by full-featured classification, except their importance order are switched around. Looking into the feature distributions between cameras and plugs, it can be observed that while both device types have devices manufactured by Belkin, there are more Belkin plug devices than there are Belkin camera devices (see Appendix F). Additionally, the camera device type has more instances of associated packet counts 1 to 4. At this point, it is hypothesized that using the full set of available features adds noise to the classification, and that using only the best features would improve the performance of the KNN classifier, maintain the performance of the RF classifier, and diminish the performance of the LDA classifier.

Table 25. Classifier Performance in Wi-Fi Cameras vs Plugs (Full-Featured Classification)

	MCC	Mean Precision	Mean Recall
KNN	-0.08	45.1%	46.8%
RF	0.51	75.6%	75.5%
LDA	0.13	57.0%	55.9%

Table 26. Feature Importance in Wi-Fi Cameras vs Plugs (Full-Featured Classification)

	Feature	Score
1	Belkin Vendor	0.455
2	Dropcam Vendor	0.273
3	Associated Packet Count	0.192
4	QoS_Null Packet Subtype	0.027
5	Packet Length	0.025
6	Data Packet Subtype	0.024
7	QoS_Null Packet Subtype	0.012
8	Tp-Link Vendor	0.000
9	Lifi Vendor	0.000

The classification task is adjusted to use only the features with the best importance scores: Belkin vendor, Dropcam vendor, and associated packet count. Table 27 provides the overall performance of the classifiers on the binary classification with best features. The classifier performances change as predicted. The KNN classifier's performance metrics are restored to decent values, with the MCC returning to a positive value, and the mean precision and mean recall scores returning to above 50%. The RF classifier maintains its previous performance, while the LDA classifier experiences a slight drop in performance.

Table 27. Classifier Performance in Wi-Fi Cameras vs Plugs (Best Features Classification)

	MCC	Mean Precision	Mean Recall
KNN	0.46	73.1%	72.7%
RF	0.66	83.2%	83.0%
LDA	0.05	61.5%	50.5%

The second notable observation made from Wi-Fi classification is the reliance of the full-featured classification on certain vendor features (e.g., Dropcam, Belkin) but complete independence from the other vendor features (e.g., Lifi, Tp-Link). A classification trial was performed to analyze how classifier performance is affected if no vendor features are used in the classification. It was hypothesized that classifier performance would decline because of the absence of all vendor features, and that the associated packet count feature would be an important feature. Table 28 and Figure 43 provides the respective performance metrics and confusion matrix of the Wi-Fi classification with no vendor features. As hypothesized, all classifiers experienced a decline in performance, as compared to the full-featured classification. However, the degree to which the performance declined was unexpected. The KNN and LDA classifiers suffered significant reductions in performance, with the KNN classifier experiencing a 38% decrease in MCC, 31% decrease in mean precision, and 25% decrease in mean recall, and the LDA classifier experiencing an 82% decrease in MCC, 32% decrease in mean precision, and 44% decrease in mean recall. The RF classifier experienced a minor drop in performance, with a 10% decrease in MCC, 13% decrease in mean precision, and 4% decrease in mean recall.

Table 28. Classifier Performance in Wi-Fi Classification (No Vendor Features)

	MCC	Mean Precision	Mean Recall
KNN	0.44	55.8%	60.3%
RF	0.43	57.4%	60.0%
LDA	0.10	48.4%	39.6%

The random forest feature importance scores were analyzed to understand which features the RF classifier found most significant. Table 29 provides the feature importance scores in the cameras vs. plugs classification. The associated packets count is observed as by far the most important feature, garnering over 90% of the feature importance. This result suggests that the vendor features provide significant information, the classifiers are able to extract sufficient information from the associated packets count feature to achieve meaningful results.

Table 29. Feature Importance in Wi-Fi Classification (No Vendor Features)

	Feature	Score
1	Associated Packet Count	0.918
2	Packet Length	0.046
3	QoS_Null Packet Subtype	0.014
4	Data Packet Subtype	0.013
5	QoS_Data Packet Subtype	0.009

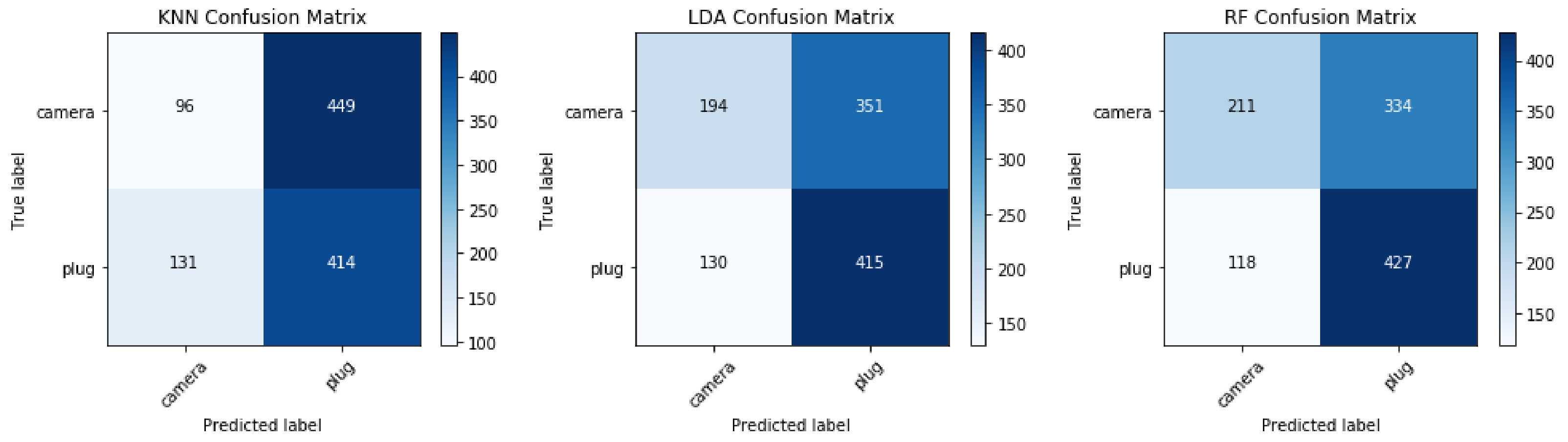


Figure 42. Confusion Matrices from Wi-Fi Cameras vs Plugs

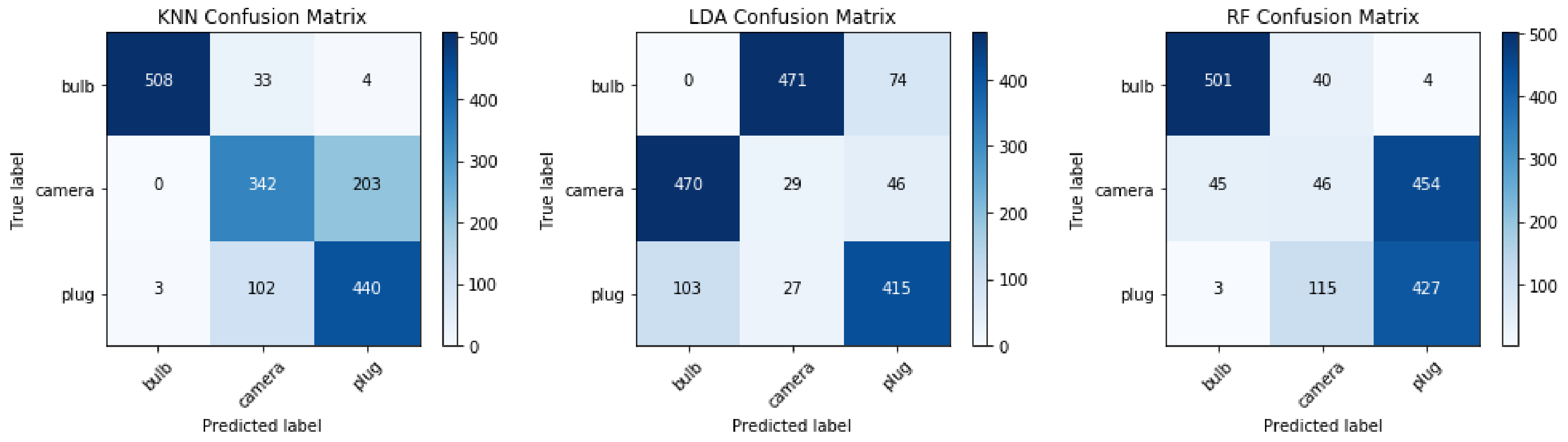


Figure 43. Confusion Matrices from Wi-Fi Classification (No Vendor Features)

5.4 Results Summary

This section reviews the results of classification trials and analyzes the classifier performances on the BLE and Wi-Fi device type classification tasks. Table 30 and Table 31 summarize the classifiers performance on the classification tasks, showing which classifiers met the criteria for high performance and low performance. Out of the BLE classifiers, only the KNN classifier managed to achieve a high performance in both the full-featured and best-features classification tasks, getting excellent MCC scores in both. Out of the Wi-Fi classifiers, KNN succeeded in achieved high performance across all metrics for the full-featured and no vendor features classification tasks. The LDA classifier attained two high performance metrics in the best-feature classification task.

Table 30. High and Low Performance BLE Classifiers

	Classifiers	MCC	Mean Precision	Mean Recall
Full-Featured	**KNN**	✓		
	RF		✗	✗
	LDA	✗	✗	✗
Best-Features	**KNN**	✓		
	RF	✓		
	LDA		✗	✗

Table 31. High and Low Performance Wi-Fi Classifiers

	Classifiers	MCC	Mean Precision	Mean Recall
Full-Featured	**KNN**	✓	✓	✓
	RF	✓	✓	✓
	LDA	✓	✓	✓
Best-Features	**KNN**		✗	
	RF		✗	
	LDA	✓	✓	
No Vendor Features	**KNN**			
	RF			
	LDA		✗	✗

VI. Conclusion

6.1 Overview

This chapter provides a summary of the research and results found during experimentation. Section 6.2 reviews the conclusions taken from the experiment and results, while Section 6.3 offers a review of the research's significance. Finally, Section 6.4 presents opportunities for future work in this research area.

6.2 Research Conclusions

The research goals that guide this thesis are successfully met through five contributions:

1. **Design and build a source of realistic smart home device traffic**: designing and building SHE to produce real-life smart home wireless traffic
2. **Develop procedures to collect and prepare the wireless traffic for machine learning classification**: developing DCP to collect and prepare wireless data for machine learning
3. **Evaluate the performance of the linear discriminant analysis (LDA), *k*-nearest neighbors (KNN), and random forests (RF) machine learning classification algorithms in determining IoT device types**: implementing and evaluating LDA, KNN, and RF classifier performances using experimental trials
4. **Determine which features are most useful for classification purposes**: reporting feature importance scores used by RF classifiers in the experiment

5. **Assess the suitability of machine learning towards the task of IoT device type classification**: discussed below, with an assessment of the machine learning approach used in this research towards the task of IoT device type classification.

The hypothesis presented in this research is if machine learning classifiers are trained using wireless traffic from a realistic smart home environment, then the classifiers can successfully identify the device type of IoT devices to a high degree of performance. This research provides mixed results towards answering this hypothesis. A smart home environment was successfully created and used towards training machine learning classifiers. However, the classifiers achieved moderate levels of performance on the BLE dataset and high levels of performances on the Wi-Fi dataset. On average, the classifiers were able to identify BLE device types with an MCC of -0.02, a mean precision of 23.7%, and a mean recall of 32.5%, and Wi-Fi device types with an MCC of 0.76, a mean precision of 85.4%, and a mean recall of 83.4%. Therefore, when viewed as a whole, the research results provide moderate support for the hypothesis. However, individual classifiers managed to achieve higher levels of success. In the BLE dataset, the KNN classifier achieved an MCC of 0.55, a mean precision of 54.1%, and a mean recall of 64%. In the Wi-Fi dataset, the LDA classifier achieved an MCC of 0.84, a mean precision of 89.7%, and a mean recall of 83.4%. While not shared by the RF and LDA classifiers, these individual moments of high performance suggest that machine learning can indeed be applied toward the task of IoT device classification, and therefore provides support for the research hypothesis.

6.3 Research Significance

The research completed in this thesis offers relevant insights for machine learning and its applications in IoT cybersecurity. This research presents the first, and at the time of this work, the only application of machine learning towards the task of IoT device type classification. This research uses three different classification methods: linear transformation (LDA), decision trees (RF), and non-parametric methods (KNN). The data exploration revealed certain wireless traffic patterns that may guide new research attempts in this area. Lastly, while the research ultimately yielded mixed results, the methodology applied a straightforward approach that serves as a necessary stepping-stone for future efforts.

6.4 Future Work

There are several opportunities in extending this research area as there is currently a lack of research in the intersection of IoT device security and machine learning classification. Five future work possibilities are offered below:

1. **Development of a sequential pattern-of-life tracking tool:** Smart device usage may indicate a subject's location within an area, providing information on the subject's pattern-of-life. By applying the techniques used in this research, it may be possible to develop a classifier that can sequentially track a subject's actions and movements and create a log of the subject's activities.
2. **Expansion of devices in the smart home environment.** The number, device type, wireless protocol, model, and manufacturer of devices in the

smart home environment can be expanded to produce a training set that includes a more robust selection of IoT devices currently in use in the market today. The inclusion of more devices with a single manufacturer but with varying device types is a particularly interesting idea as only a limited number of these devices were used in this research.

3. **Scalability to multiple smart home environments**. Attempts to deploy this research in a scalable matter could involve expanding the smart home environment to several environments. The classifiers were trained using a dataset limited to a single home environment with an individual user. The dataset can be expanded to include multiple home environments with multiple users. Doing so would introduce a larger variety of smart device usage patterns that may reduce bias in the dataset due to only having a single user. Care must be taken in first developing a big data system that can handle the substantial volume of wireless traffic data produced by multiple smart home environments.
4. **Feature extraction.** Features can be obtained across different levels in the data. BLE and Wi-Fi header information from the individual packet level is the primary source of features used by DCP. Features from the flow level and connection level can be extracted and derived. Applying a more complex approach that factors device interactions with time-based features may produce more promising results.

5. **Classification algorithms.** More classification algorithms and techniques can be applied. A myriad of sophisticated techniques, including support vector machines and deep learning, exist that may yield better classification performances.

Appendix A. Device Details

Name	Brand	Model	Model Number	Serial Number	Device Type	Protocol	Device Setup	MAC Address
August1	August	Smart Lock	ASL3B	L4FWQ02EL4	Lock	BLE	Training	
August2	August	Smart Lock	ASL01	L1GHX005D6	Lock	BLE	Test	
Door1	Eve	Door & Window	2ED309901000	CU49F1A03655	Door Sensor	BLE	Training	
Door2	Eve	Door & Window	20EAL9901	DV13H1A00054	Door Sensor	BLE	Test	
Dropcam	Dropcam	WiFi Video Monitoring	DROPCAM3HDB	308CFB3A1AAD	Camera	WiFi	Training	308CFB3A1AAD
Home1	BLE Home	Door Sensor		1444BE	Door Sensor	BLE	Training	
Home2	BLE Home	Door Sensor		1444A1	Door Sensor	BLE	Training	
Insight	Belkin	Wemo Insight Switch	F7C029V2	231618K12013ED	Plug	WiFi	Training	14918224DD35
Kasa	TPLink	Kasa Cam	KC120	2184339000783	Camera	WiFi	Test	AC84C6977CCC
Kevo	Kwikset	Kevo	925GED1500MK2	3022AMK2	Lock	BLE	Training	
Lifx1	Lifx	Lightbulb	LHA19E26UC10	D073D526B84C	Light Bulb	WiFi	Training	D073D526B84C
Lifx2	Lifx	Lightbulb	LHA19E26UC10	D073D526C927	Light Bulb	WiFi	Test	D073D526C927
Mini	Belkin	Wemo Mini	F7C063	221708K0100DEA	Plug	WiFi	Test	6038E0EE7CE5

Netcam1	Belkin	NetCam HD+	F7D7602V2	35418VB2200526	Camera	WiFi	Training	EC1A59E4FD41
Netcam2	Belkin	NetCam HD+	F7D7602V2	35418VB2200320	Camera	WiFi	Training	EC1A59E4FA09
Netcam3	Belkin	NetCam HD+	F7D7602V2	35418VB2200833	Camera	WiFi	Test	EC1A59E5020D
Push	SensorPush	Smart Sensor	HT1	2AL9XHT1	Temp Sensor	BLE	Training	
Room1	Eve	Room	2ER309901000	BU45F1A03216	Temp Sensor	BLE	Training	
Room2	Eve	Room	2ER309901000	BU35E1A02542	Temp Sensor	BLE	Test	
Yosemite	Netgear	Nighthawk X4S AC2600	R7800	4H4E855K01D4E	Router	WiFi	NA	78D2944DAB3F
Switch1	Belkin	Wemo Switch	F7C027	221621K01027F9	Plug	WiFi	Training	149182CDDF3D
Switch2	Belkin	Wemo Switch	F7C027	221343K010034E	Plug	WiFi	Training	B4750E0D9465
Switch3	Belkin	Wemo Switch	F7C027	221342K0101C51	Plug	WiFi	Test	B4750E0D33D5
Switch4	Belkin	Wemo Switch	F7C027	221417K01007F1	Plug	WiFi	Test	94103E2B7A55
TpBulb	TPLink	Smart WiFi LED Bulb	LB100E26	217C581015895	Light Bulb	WiFi	Training	B04E26C52A41
TpPlug	TPLink	Smart WiFi Plug	HS100	2179815005849	Plug	WiFi	Test	704F57F9E1B8

Weather	Eve	Weather	2EW309901000	AU40F1A04650	Temp Sensor	BLE	Training	

Appendix B. Hyperparameter Tuning Script

```
# coding: utf-8

# In[12]:


# from Pipeline import BLEPipeline, WifiPipeline
from MulticlassDCP import BLEMulticlassDCP, WifiMulticlassDCP

# General data processing
import numpy as np
import pandas as pd

# Plotting
import matplotlib.pyplot as plt
import seaborn as sns
import scikitplot as skplt

# ML libraries
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import balanced_accuracy_score, precision_score, recall_score, roc_auc_score, confusion_matrix
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score

# System libraries
import itertools
import random, time

# Warning filtering
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
plt.rcParams.update({'figure.max_open_warning': 0})


# # Wi-Fi Tuning

# In[13]:


start_time = time.time()


# In[14]:


w = WifiMulticlassDCP()


# In[15]:


```

```python
df = w.make_dataframe()

# Take out packets from router
df = df[df["DeviceType"]!="router"]


# In[16]:


# Divide training and test sets
df_train = df[df['Set']=='train']
df_test = df[df['Set']=='test']


# In[17]:


# Wifi: Define which features to use
features_list = [
        # Packet info
        "PacketLength",

        # Vendor
         "Belkin", "Dropcam", "Lifi", "Tp-link",

        # 802.11 Data subtype
        "Data", "QoS_Data", "QoS_Null",

        # Associated Packets
        "Assoc_Packets"]

# Define what the response classes are
y_list = ["bulb", "camera", "plug"]


# In[18]:


# Define grid values
knn_param_grid = dict(n_neighbors=np.arange(1,19,2))
rf_param_grid = dict(max_features=np.linspace(2, len(features_list), num=5, dtype=int))
lda_param_grid = dict(n_components=np.arange(1,5))

# Time wifi gridsearch
wifi_start = time.time()

      # Run gridsearch
      w_knn = w.tune_gridsearch(KNeighborsClassifier(), knn_param_grid, df_train,
                              features_list, y_list)
      w_rf = w.tune_gridsearch(RandomForestClassifier(), rf_param_grid, df_train,
                             features_list, y_list)
      w_lda = w.tune_gridsearch(LinearDiscriminantAnalysis(priors=[0.61678342, 0.37815795, 0.00505862]), lda_param_grid, df_train,
                              features_list, y_list)

```

```python
wifi_end = time.time() - wifi_start


# In[19]:


print wifi_end, "sec"


# In[20]:


print w_knn['grid_result'].best_score_, w_knn['grid_result'].best_params_
print w_lda['grid_result'].best_score_, w_lda['grid_result'].best_params_
print w_rf['grid_result'].best_score_, w_rf['grid_result'].best_params_


# In[21]:


w.plot_all_vcs([w_knn, w_lda, w_rf])


# # BLE Tuning

# In[22]:


b = BLEMulticlassDCP()


# In[23]:


bdf = b.make_dataframe()


# In[24]:


# Divide training and test sets
bdf_train = bdf[bdf['Set']=='train']
bdf_test = bdf[bdf['Set']=='test']


# In[25]:


# BLE: Define which features to use
features_list = [
    # Packet info
    "PacketLength", "BLE_LL_Length",

    # Associate Packets
    "Assoc_Packets",

    # Channel number
    "Channel_0", "Channel_12", "Channel_39",

    # PDU Type
    "SCAN_RSP", "ADV_IND", "SCAN_REQ",
    "CONNECT_REQ", "ADV_NONCONN_IND", "ADV_DIRECT_IND"]

y_list = ["door", "lock", "temp"]


# In[26]:


# Define grid values
knn_param_grid = dict(n_neighbors=np.arange(1,19,2))
rf_param_grid = dict(max_features=np.linspace(2, len(features_list), num=5, dtype=int))
lda_param_grid = dict(n_components=np.arange(1,5))

# Time BLE gridsearch
ble_start = time.time()

# Run gridsearch
b_knn = b.tune_gridsearch(KNeighborsClassifier(), knn_param_grid, bdf_train,
                        features_list, y_list)
b_rf = b.tune_gridsearch(RandomForestClassifier(), rf_param_grid, bdf_train,
                       features_list, y_list)
b_lda = b.tune_gridsearch(LinearDiscriminantAnalysis(priors=[0.59063441, 0.23399223, 0.17537336]), lda_param_grid, bdf_train,
                        features_list, y_list)

ble_end = ble_start - time.time()


# In[27]:


print ble_end


# In[28]:


print b_knn['grid_result'].best_score_, b_knn['grid_result'].best_params_
print b_lda['grid_result'].best_score_, b_lda['grid_result'].best_params_
print b_rf['grid_result'].best_score_, b_rf['grid_result'].best_params_


# In[29]:


b.plot_all_vcs([b_knn, b_lda, b_rf])
```

```


# In[30]:


end_time =  time.time() - start_time
total_gridsearch_time = ble_end + wifi_end
print total_gridsearch_time, "sec"
print end_time, "sec"
```

Appendix C. Experimental Procedure

1. Set up devices in designated locations as described in Section 3.2.5. Configure device actions as described on Table 4.
2. Set up motion source. Ensure Arduino microcontroller is powered on and operational.
3. Let devices reach steady state over the course of one day.
4. Perform scans using Plugable Bluetooth adapter and Alfa card. Use the commands as described in Figure 22 and 15 for Wi-Fi scanning, and Figure 25 for BLE scanning.
5. Perform sniffing using Ubertooth One sniffers and Alfa card. Use the commands as described in Section 3.3.2. Let run for 8 hours. Perform for a total of three days.
6. Perform model tuning by executing the Jupyter notebook in Appendix B. Update MulticlassDCP.py with the best-performing hyperparameter values. Find class prior probability values and update LDA classifier in MulticlassDCP.py with these values.
7. Perform data preprocessing and model testing. Use the Jupyter notebooks in Appendix D and Appendix E for BLE testing and Wi-Fi testing, respectively.

Appendix D. Multiclass BLE Classification

```
# coding: utf-8

# In[1]:


from MulticlassDCP import BLEMulticlassDCP

# General data processing
import numpy as np
import pandas as pd

# Plotting
import matplotlib.pyplot as plt
import seaborn as sns

# ML libraries
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import balanced_accuracy_score, precision_score, recall_sco
re, confusion_matrix, matthews_corrcoef
from sklearn.model_selection import GridSearchCV, cross_val_score
from imblearn.under_sampling import RandomUnderSampler

# System libraries
import itertools
import random, time

# Warning filtering
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
plt.rcParams.update({'figure.max_open_warning': 0})


# # Create, process dataframe

# In[2]:


start_time = time.time()


# In[3]:


b = BLEMulticlassDCP()


# In[4]:


df = b.make_dataframe()
```

```python


# # Prep dataset

# In[5]:


# BLE: Define which features to use
features_list = [
    # Packet info
    "PacketLength", "BLE_LL_Length",

    # Associate Packets
    "Assoc_Packets",

    # Channel number
    "Channel_0", "Channel_12", "Channel_39",

    # PDU Type
    "SCAN_RSP", "ADV_IND", "SCAN_REQ",
    "CONNECT_REQ", "ADV_NONCONN_IND", "ADV_DIRECT_IND"]

y_list = ["door", "lock", "temp"]


# In[6]:


# Prep training set
df_train = df[df['Set']=='train']
print df_train['DeviceType'].value_counts()


# In[7]:


df_test = df[df['Set']=='test']

# Show initial test set imbalance
print "Initial test set distribution:"
print df_test['DeviceType'].value_counts()
df_test['DeviceType'].value_counts().sort_index().plot(kind='bar', title="Test Packet Counts Before Resampling",logy=True);


# In[8]:


       # Downsample test set so that there is equal chance that the classifier will choose any given class
       rds = RandomUnderSampler(random_state=42)
       test_X_downsampled, test_y_downsampled = rds.fit_resample(df_test[features_list], df_test['DeviceType'])

       # Show class counts after downsampling
       unique, counts = np.unique(test_y_downsampled, return_counts=True)
       print np.asarray((unique, counts)).T

```

```
# In[9]:


# Recreate df_test
df_test_downsampled = pd.DataFrame(test_X_downsampled,columns=features_list)
df_test_downsampled['DeviceType'] = test_y_downsampled


# # Run multiclass on all features

# In[10]:


multiclass_start = time.time()

preds, metrics, cms, feature_importance = b.run_multiclass(df_train, df_test_downsampled, features_list, y_list)

multiclass_end = time.time() - multiclass_start


# ## Report results

# ### Report confusion matrices

# In[11]:


b.plot_all_confusion_matrices(cms, y_list)


# ### Report metrics

# In[12]:


metrics_df = b.report_metrics(metrics, y_list, 'ble-multiclass_metrics')
display(metrics_df)


# ### Report feature importance

# In[13]:


f_i = b.report_featureimportance(feature_importance, features_list)
display(f_i)


# # Residuals Analysis

# ## Use *k* top features only

# ### Find *k* where *k* is the count of features that yields best BACC
```

```

    # In[14]:


    fs_start = time.time()

    # Find best features using KBest scheme
    feature_selection = []
    for i in range(0,len(f_i)):
        top_features = list(f_i.index[0:i+1])

        tf_preds, tf_metrics, tf_cms, tf_feature_importance = b.run_multiclass(df_train, df_test_downsampled, top_features, y_list,use_tuned=False)
        tf_metrics_df = b.report_metrics(tf_metrics, y_list, to_csv=False)

        ave = np.average(tf_metrics_df['Mean_Recall'])
        feature_selection.append(ave)

    fs_end = time.time() - fs_start


    # In[15]:


    k = feature_selection.index(max(feature_selection))
    print 'Best Mean Recall',max(feature_selection),":", k+1, "features"


    # In[16]:


    feature_selection


    # ### Run multiclass with top *k* features (*k* = 3)

    # In[17]:


    # Run multiclass with top 3 features
    tf_preds, tf_metrics, tf_cms, tf_feature_importance = b.run_multiclass(df_train, df_test_downsampled, list(f_i.index[0:k+1]), y_list,use_tuned=False)


    # In[18]:


    tf_metrics_df = b.report_metrics(tf_metrics, y_list, 'ble-topfeatures_metrics')
    display(tf_metrics_df)


    # ### Plot confusion matrices

    # In[19]:


    b.plot_all_confusion_matrices(tf_cms, y_list)
```

```python


# ### Report feature importance of TF3

# In[20]:


b.report_featureimportance(tf_feature_importance, f_i.index[0:k+1])


# ## Error analysis

# **Error 1**: The main error across all classifiers is the misclassification of door devices as temp devices.

# In[21]:


# Get door and temp packets
df_train_doortemp = df_train[(df_train['DeviceType']=='door') | (df_train['DeviceType']=='temp')]

df_test_doortemp = df_test_downsampled[(df_test_downsampled['DeviceType']=='door') | (df_test_downsampled['DeviceType']=='temp')]


# In[22]:


# Run multiclass on just the two device types
doortemp_preds, doortemp_metrics, doortemp_cms, doortemp_feature_importance = b.run_multiclass(df_train_doortemp, df_test_doortemp, features_list, ['door','temp'],use_tuned=False)


# In[23]:


doortemp_metrics_df = b.report_metrics(doortemp_metrics, ['door','temp'], 'ble-doortemp_metrics')
display(doortemp_metrics_df)


# In[24]:


b.plot_all_confusion_matrices(doortemp_cms, ['door','temp'])


# It appears that with just the two classes, the classifiers cannot distinguish between the two devices. The next step is to look at the feature selection

# In[25]:


b.report_featureimportance(doortemp_feature_importance, features_list)

```

```

        # In[26]:


        # Run door vs temp with top 3 features
        dt3_preds, dt3_metrics, dt3_cms, dt3_feature_importance = b.run_multiclass(df_train_doortemp, df_test_doortemp, features_list[0:3], ['door','temp'],use_tuned=False)


        # In[27]:


        b.report_metrics(dt3_metrics, ['door','temp'], 'ble-dt3_metrics')


        # No difference with top 3 features.

        # In[28]:


        f, axes = plt.subplots(3, 1, figsize=(6, 14))
        sns.countplot(x='DeviceType', hue='PacketLength',ax=axes[0], data=df_test_doortemp);
        sns.countplot(x='DeviceType', hue='BLE_LL_Length',ax=axes[1], data=df_test_doortemp);
        sns.countplot(x='DeviceType', hue='Assoc_Packets',ax=axes[2], data=df_test_doortemp);


        # # Report times

        # In[29]:


        print multiclass_end
        print fs_end
        end_time = time.time() - start_time
        print end_time
```

Appendix E. Multiclass Wi-Fi Classification

```
# coding: utf-8

# In[1]:


# from Pipeline import WifiPipeline
from MulticlassDCP import WifiMulticlassDCP

# General data processing
import numpy as np
import pandas as pd

# Plotting
import matplotlib.pyplot as plt
import seaborn as sns

# ML libraries
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import balanced_accuracy_score, precision_score, recall_score, confusion_matrix, matthews_corrcoef
from sklearn.model_selection import GridSearchCV, cross_val_score
from imblearn.under_sampling import RandomUnderSampler

# System libraries
import itertools
import random, time

# Warning filtering
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
plt.rcParams.update({'figure.max_open_warning': 0})


# # Create, process dataframe

# In[2]:


start_time = time.time()


# In[3]:


w = WifiMulticlassDCP()


# In[4]:


```

```
df = w.make_dataframe()

# Take out packets from router
df = df[df["DeviceType"]!="router"]
print len(df)


# # Prep dataset

# In[5]:


# Wifi: Define which features to use
features_list = [
        # Packet info
        "PacketLength",

        # Vendor
         "Belkin", "Dropcam", "Lifi", "Tp-link",

        # 802.11 Data subtype
        "Data", "QoS_Data", "QoS_Null",

        # Associated Packets
        "Assoc_Packets"]

# Define what the response classes are
y_list = ["bulb", "camera", "plug"]


# In[6]:


# Prep training set
df_train = df[df['Set']=='train']
print df_train['DeviceType'].value_counts()


# In[7]:


df_test = df[df['Set']=='test']

# Show initial test set imbalance
print "Initial test set distribution:"
print df_test['DeviceType'].value_counts()
df_test['DeviceType'].value_counts().sort_index().plot(kind='bar', title="Test Packet Counts Before Resampling",logy=True);


# In[8]:


# Downsample test set so that there is equal chance that the classifier will choose any given class
rds = RandomUnderSampler(random_state=42)
test_X_downsampled, test_y_downsampled = rds.fit_resample(df_test[features_list], df_test['DeviceType'])
```

```

        # Show class counts after downsampling
        unique, counts = np.unique(test_y_downsampled, return_counts=True)
        print np.asarray((unique, counts)).T


        # In[9]:


        # Recreate df_test
        df_test_downsampled = pd.DataFrame(test_X_downsampled,columns=features_list)
        df_test_downsampled['DeviceType'] = test_y_downsampled


        # # Run multiclass

        # In[10]:


        multiclass_start = time.time()

        preds, metrics, cms, feature_importance = w.run_multiclass(df_train, df_test_downsampled, features_list, y_list)

        multiclass_end = time.time() - multiclass_start


        # # Report results

        # ## Report confusion matrices

        # In[11]:


        w.plot_all_confusion_matrices(cms, y_list)
        plt.savefig('Results/CM/wifi-cm-full.png')


        # ## Report metrics

        # In[12]:


        metrics_df = w.report_metrics(metrics, y_list, 'wifi-multiclass_metrics')
        display(metrics_df)


        # ## Report feature importance

        # In[13]:


        f_i = w.report_featureimportance(feature_importance, features_list)
        display(f_i)


        # # Residuals Analysis

        # ## Use only top 3 features

        # ### Run multiclass with top 3 features

        # In[14]:


        fs_start = time.time()
        # Run multiclass with top 3 features
        tf3_preds, tf3_metrics, tf3_cms, tf3_feature_importance = w.run_multiclass(df_train, df_test_downsampled, list(f_i.index[0:3]), y_list, use_tuned=False, use_priors=True)

        fs_end = time.time() - fs_start


        # In[15]:


        tf3_metrics_df = w.report_metrics(tf3_metrics, y_list, 'wifi-tf3_metrics')
        display(tf3_metrics_df)


        # ### Plot confusion matrices

        # In[16]:


        w.plot_all_confusion_matrices(tf3_cms, y_list)
        plt.savefig('Results/CM/wifi-cm-best3.png')


        # ### Report feature importance of 3 best

        # In[17]:


        w.report_featureimportance(tf3_feature_importance, f_i.index[0:3])


        # ## Remove vendor features

        # In[18]:


        # Remove vendors features
        nv_features = ['PacketLength', 'Data', 'QoS_Data', 'QoS_Null', 'Assoc_Packets']


        # In[19]:


        nv_start = time.time()

```

```
    # Run multiclass without vendors
    nv_preds, nv_metrics, nv_cms, nv_feature_importance = w.run_multiclass(df_train, df_test_downsampled, nv_features, y_list, use_tuned=False)

    nv_end = time.time() - nv_start


    # In[20]:


    nv_metrics_df = w.report_metrics(nv_metrics, y_list, 'wifi-novendor_metrics')
    display(nv_metrics_df)


    # In[21]:


    w.plot_all_confusion_matrices(nv_cms, y_list)
    plt.savefig('Results/CM/wifi-cm-novendor.png')


    # In[22]:


    w.report_featureimportance(nv_feature_importance, nv_features)


    # ## Error analysis

    # In[23]:


    def output_decisionpath(model, features_list, class_names, filename):
        # Source: https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-python-using-scikit-learn-38ad2d75f21c

        # Extract single tree
        estimator = model.estimators_[5]

        from sklearn.tree import export_graphviz
        # Export as dot file
        export_graphviz(estimator, out_file='tree.dot',
                        feature_names = features_list,
                        class_names = class_names,
                        rounded = True, proportion = False,
                        precision = 2, filled = True)

        # Convert to png using system command (requires Graphviz)
        from subprocess import call
        call(['dot', '-Tpng', 'tree.dot', '-o', 'Results/'+filename+'.png', '-Gdpi=600'])


    # **Error 1**: KNN and RF confuse camera and plugs

    # In[24]:

```

```python

    # Get camera and plug packets
    df_train_camplugs = df_train[(df_train['DeviceType']=='camera') | (df_train['DeviceType']=='plug')]

    df_test_camplugs = df_test_downsampled[(df_test_downsampled['DeviceType']=='camera') | (df_test_downsampled['DeviceType']=='plug')]


    # Isolate the two classes

    # In[25]:


    # Run multiclass on just the two device types
    camplugs_preds, camplugs_metrics, camplugs_cms, camplugs_feature_importance = w.run_multiclass(df_train_camplugs, df_test_camplugs, features_list, ['camera','plug'], use_tuned=False, use_priors=False)


    # In[26]:


    camplugs_metrics_df = w.report_metrics(camplugs_metrics, ['camera','plug'], 'wifi-camplugs_metrics')
    display(camplugs_metrics_df)


    # In[27]:


    w.plot_all_confusion_matrices(camplugs_cms, ['camera','plug'])
    plt.savefig('Results/CM/wifi-cm-camplug-full.png')


    # In[28]:


    w.report_featureimportance(camplugs_feature_importance, features_list)


    # Show features

    # In[29]:


    f, axes = plt.subplots(3, 1, figsize=(6, 14))
    sns.countplot(x='DeviceType', hue='Belkin',ax=axes[0], data=df_test_camplugs);
    sns.countplot(x='DeviceType', hue='Dropcam',ax=axes[1], data=df_test_camplugs);
    sns.countplot(x='DeviceType', hue='Assoc_Packets',ax=axes[2], data=df_test_camplugs);
    plt.legend(loc='upper right');


    # Use only top 3 features for camera vs plugs classification

```

```
        # In[30]:


        # Run multiclass on just the two device types
        camplugs_preds_bf, camplugs_metrics_bf, camplugs_cms_bf, camplugs_feature_importance_bf= w.run_multiclass(df_train_camplugs, df_test_camplugs, ['Belkin','Dropcam','Assoc_Packets'], ['camera','plug'], use_tuned=False, use_priors=False)


        # In[31]:


        camplugs_metrics__bf_df = w.report_metrics(camplugs_metrics_bf, ['camera','plug'], 'wifi-camplugs_bestfeatures_metrics')
        display(camplugs_metrics__bf_df)


        # In[32]:


        w.plot_all_confusion_matrices(camplugs_cms_bf, ['camera','plug'])
        plt.savefig('Results/CM/wifi-cm-camplug-best3.png')


        # In[33]:


        w.report_featureimportance(camplugs_feature_importance_bf, ['Belkin','Dropcam','Assoc_Packets'])


        # In[34]:


        print multiclass_end
        print fs_end
        print nv_end
        print time.time() - start_time
```

Appendix F. Classification Analysis Graphs

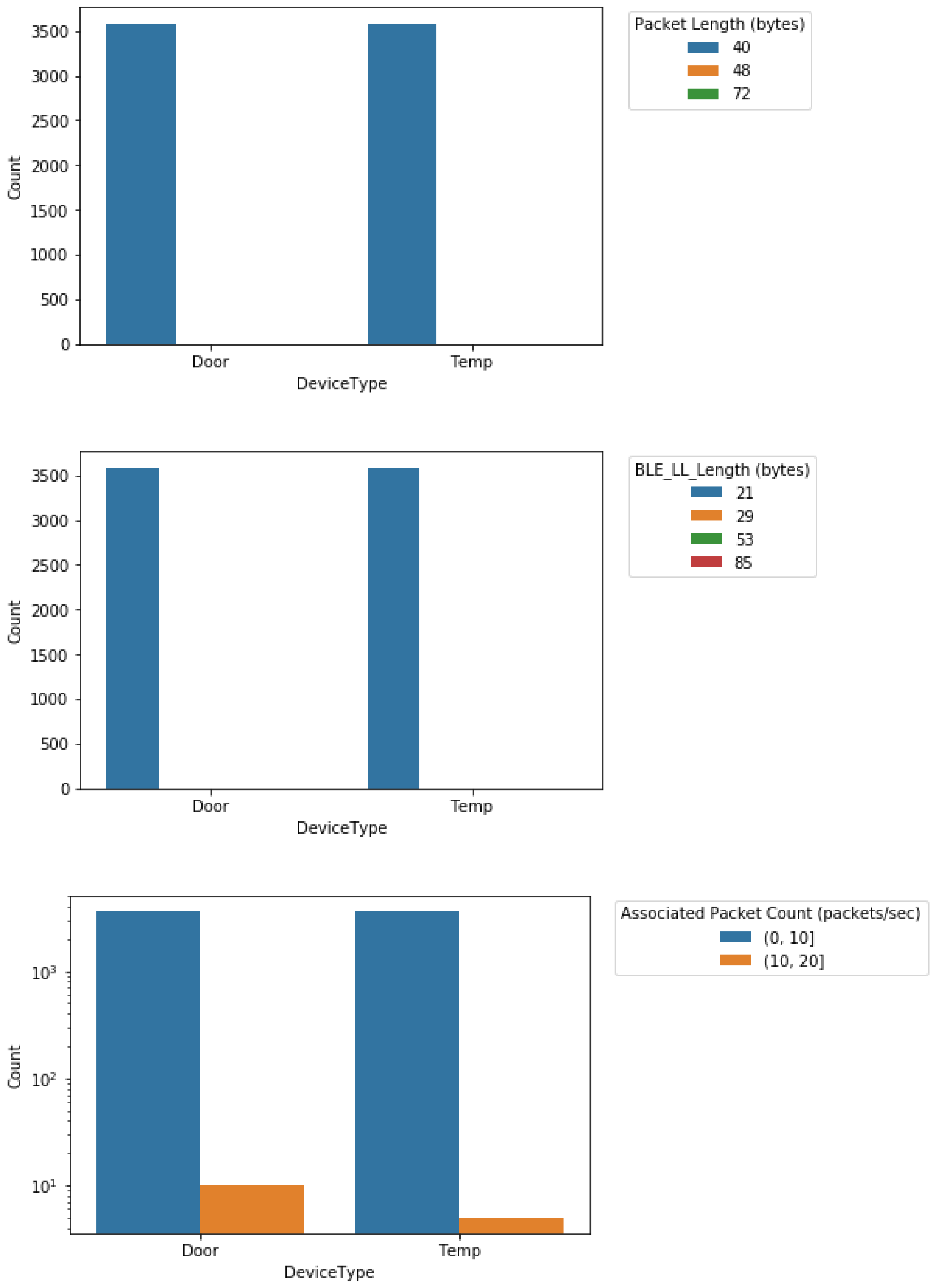
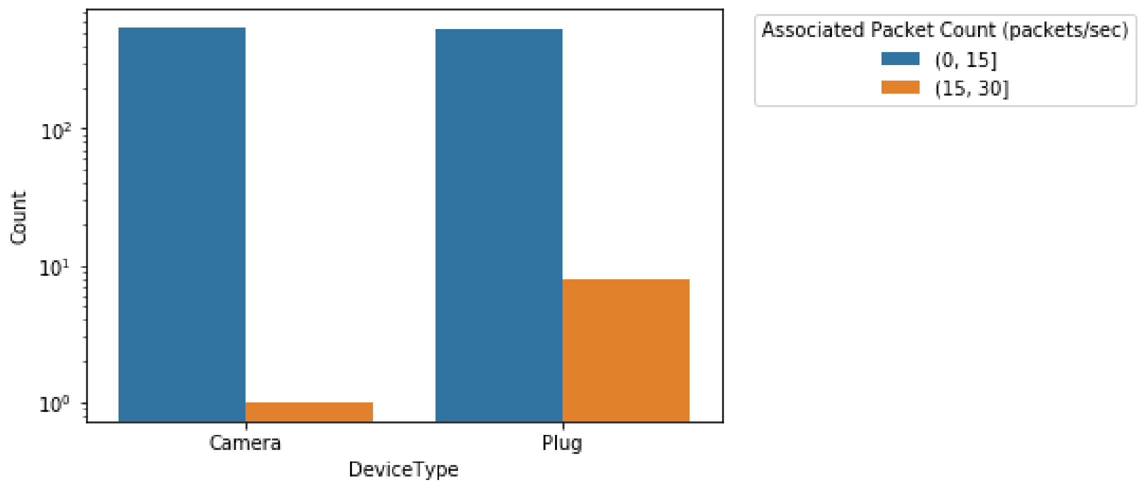
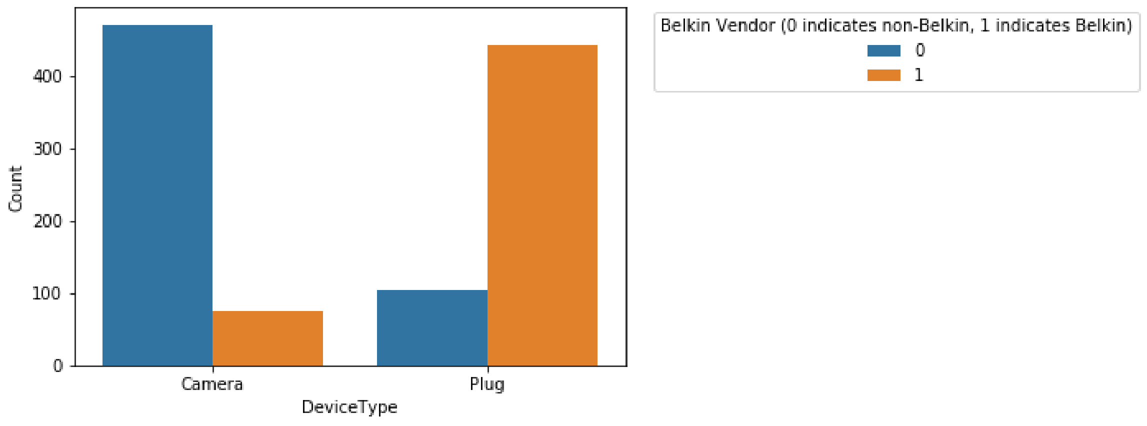


Figure 44. BLE Door Sensors vs. Temperature Sensors Features





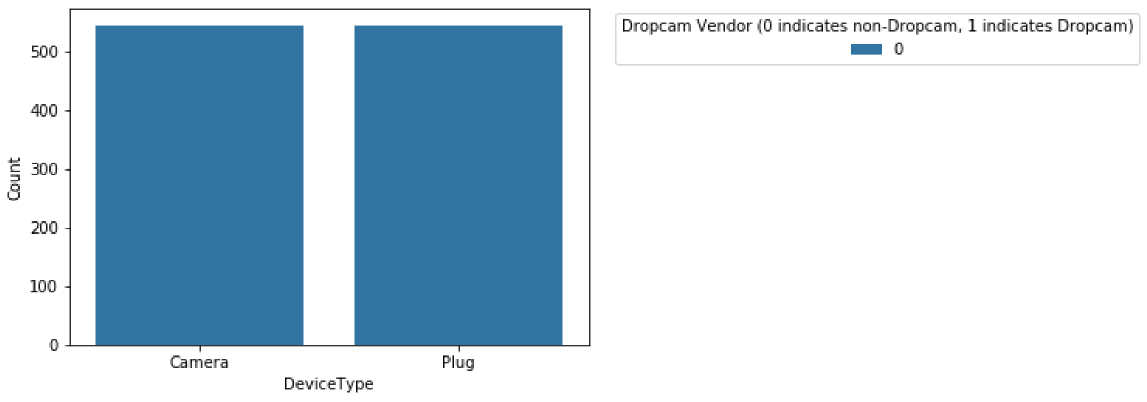


Figure 45. Wi-Fi Cameras vs. Plugs Features

Bibliography

1. United States Government Accountability Office, “Internet of Things: Enhanced Assessments and Guidance Are Needed to Address Security Risks in DoD,” 2017, [Online]. Available: https://www.gao.gov/assets/690/686203.pdf.

2. S. M. Beyer, “Pattern-of-Life Modeling using Data Leakage in Smart Homes,” Air Force Institute of Technology, 2018, [Online]. Available: https://scholar.afit.edu/cgi/viewcontent.cgi?article=2793&context=etd.

3. A. Nordrum, “Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated,” *IEEE Spectrum*, 2016. [Online]. Available: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated. [Accessed: 06-Jun-2018].

4. A. Razaque, P. Oddo, F. H. Amsaad, M. Sangavikar, S. Manchikatla, and Niraj, “Power reduction for Smart Homes in an Internet of Things framework,” in *IEEE International Conference on Electro Information Technology*, Grand Forks, North Dakota, USA, 2016, pp. 117–121.

5. M. S. Gast, *802.11 Wireless Networks: The Definitive Guide*, 2nd ed. O’Reilly Media, Inc., 2005.

6. R. Heydon, *Bluetooth Low Energy: The Developer’s Handbook*. Prentice Hall, 2012.

7. “Bluetooth Core Specifications,” 2018. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification. [Accessed: 06-Apr-2018].

8. K. Townsend, “Introduction to Bluetooth Low Energy - GATT.” [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. [Accessed: 19-Jul-2018].

9. Anders Strand, “Bluetooth Low Energy: Central Tutorial,” *Nordic Semiconductor*, 2016. [Online]. Available: https://devzone.nordicsemi.com/tutorials/b/bluetooth-low-energy/posts/ble-central-tutorial. [Accessed: 06-May-2018].

10. A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM Journal of Research and Development*, vol. 3, no. 3, p. 210, 1959.

11. A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. O’Reilly Media, Inc., 2017.

12. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2016.

13. F. Provost and T. Fawcett, *Data Science for Business*. O'Reilly, 2013.

14. G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, 1st ed. Springer, 2013.

15. A. Padmanabha and C. Williams, "K-nearest Neighbors," *Brilliant.org*. [Online]. Available: https://brilliant.org/wiki/k-nearest-neighbors/. [Accessed: 20-Aug-2018].

16. J. Erman, A. Mahanti, and M. Arlitt, "Internet Traffic Identification using Machine Learning," in *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, San Francisco, California, USA, 2006, pp. 1–6.

17. T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*, Taormina, Sicily, Italy, 2004, p. 121.

18. T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.

19. B. J. McGregor A., Hall M., Lorier P., "Flow Clustering Using Machine Learning Techniques.," in *Passive and Active Network Measurement*, 2004.

20. L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic Classification on the Fly," *ACM Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review*, vol. 36, no. 2, 2006.

21. J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and Discriminating Between Web and Peer-To-Peer Traffic in the Network Core," in *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, Banff, Alberta, Canada, pp. 883–892.

22. M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," in *Proceedings of the 4th ACM/SIGCOMM on Internet Measurement Conference (IMC)*, San Diego, California, USA, 2004.

23. A. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Banff, Alberta, Canada, 2005.

24. T. Auld, A. W. Moore, and S. F. Gull, “Bayesian neural networks for internet traffic classification.,” *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, 2007.

25. J. S. Atkinson, “Your WiFi Is Leaking: Inferring Private User Information Despite Encryption,” University College London, 2015, [Online]. Available: http://discovery.ucl.ac.uk/1470734/.

26. B. Copos, K. Levitt, M. Bishop, and J. Rowe, “Is Anybody Home? Inferring Activity from Smart Home Network Traffic,” *Proceedings - 2016 IEEE Symposium on Security and Privacy Workshops, SPW 2016*, pp. 245–251, 2016.

27. Y. Meidan *et al.*, “ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis,” in *Proceedings of the ACM Symposium on Applied Computing*, Marrakech, Morocco, 2017, pp. 506–509.

28. P. Wang, F. Ye, X. Chen, and Y. Qian, “Datanet: Deep learning based encrypted network traffic classification in SDN home gateway,” *IEEE Access*, vol. 6, pp. 55380–55391, 2018.

29. “Plugable Bluetooth Adapter.” [Online]. Available: https://plugable.com/products/usb-bt4le/. [Accessed: 05-Feb-2019].

30. “Alfa Indoor WiFi USB Antenna.” [Online]. Available: https://www.alfa.com.tw/WiFi USB Antenna.html. [Accessed: 05-Feb-2019].

31. “Ubertooth One.” [Online]. Available: https://www.wallofsheep.com/collections/lan-taps/products/ubertooth-one-fully-assembled. [Accessed: 05-Feb-2019].

32. Radiocommunication Sector of International Telecommunications Union, “Isolation between antennas of IMT base stations in the land mobile service,” 2011. [Online]. Available: https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2244-2011-PDF-E.pdf. [Accessed: 17-Dec-2018].

33. P. Roshan and J. Leary, *802.11 Wireless LAN Fundamentals*. Cisco, 2013.

34. “macvendors.co.” [Online]. Available: http://macvendors.co/. [Accessed: 05-Feb-2019].

35. “IEEE Standards Association Registration Authority.” [Online]. Available: https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries. [Accessed: 05-Feb-2019].

36. “Documentation of scikit-learn 0.20.2.” [Online]. Available: https://scikit-learn.org/stable/documentation.html. [Accessed: 13-Dec-2018].

37. “Jupyter Notebook Documentation.” [Online]. Available: https://jupyter-notebook.readthedocs.io/en/stable/. [Accessed: 05-Feb-2019].

38. G. Jurman, S. Riccadonna, and C. Furlanello, “A comparison of MCC and CEN error measures in multi-class prediction,” *PLoS ONE*, vol. 7, no. 8, 2012.

39. J. Gorodkin, “Comparing two K-category assignments by a K-category correlation coefficient,” *Computational Biology and Chemistry*, vol. 28, no. 5–6, pp. 367–374, 2004.

40. J. Tang, S. Alelyani, and H. Liu, “Feature selection for classification: A review,” *Data Classification: Algorithms and Applications*, pp. 37–64, 2014.