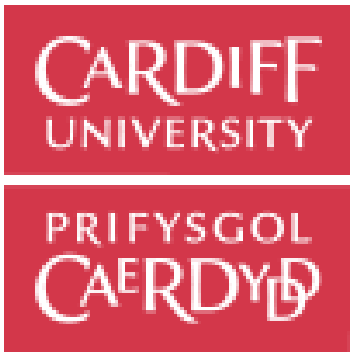


Can IoT Devices be Identified Through Their Device Type Fingerprint?

Author: Rachel Keyte

Supervisor: George Theodorakopoulos
Module number: CM3203
Module Title: One Semester Individual Project
Credits: 40

Abstract

The security of IoT devices is becoming an afterthought to producing devices that satisfy the growing interest of the population. This can expose users private data or in some cases, physical security.
This report demonstrates how the type of IoT device can be determined through passively monitoring its traffic across a network.
The packet header information is analysed to find the 'Largest Packet', 'Smallest Packet', 'Largest Gap', 'Smallest Gap' and 'Average Gap' for each 6 minute interval.
Once a device type fingerprint is found for each of the devices available they are used to train a Random Forest classifier model that is able to take previously unseen data and make predictions about the type of device present.
The results show that from using a device type fingerprint constructed from these attributes, the device type can be predicted with a high level of accuracy.

Acknowledgement

I would like to thank George Theodorakopoulos for his guidance and support for the duration of the project.

Contents

List of Figures

1 Introduction

Connection to the internet is no longer restricted to devices such as laptops and phones but is now extending to everyday objects. Cars, cameras, light bulbs and a variety of new devices are becoming connected and at a rapid rate. IoT refers to devices that can transfer data over a network without human interaction.
Pressure for companies to remain fashionable and innovative, ahead of their rivals means that the security of these devices is often compromised. So, can the type of device be determined from its traffic and what does that mean for businesses and consumers?

This project addresses the following problems: a) Can the fingerprints for certain IoT devices be identified through passive monitoring of the network traffic? And b) In an IoT environment, can the type of devices be determined using the device type fingerprint?

The first part of the project involves analysing network traffic from the IoT devices. The output produces a device type fingerprint to represent the device being read.
The results of the analysis are used initially as training data to create the model. Subsequent analysed data is used by the final model to make predictions about the previously unseen data. The most common prediction for the data determines the device type from the collection of previously seen devices in the training data.
The final stage is discussing the implications of the ability to determine the type of device on a network from a consumer and business perspective. These implications can be both positive and negative.

Additional objectives of this project are to prove that it can be done using a free and accessible solution that is available on multiple platforms.

The scope for the project includes IoT devices that use WiFi and can have both wired or wireless connections.
The range of devices being tested on is limited to an Amazon Echo Dot, Belkin Netcam, TP-link NC200 IP Camera, TP-link Smart Plug, Lamp, Samsung SmartThings device and a Hive with a motion sensor and door/window sensor connected. These are supplied by the IoT lab at Cardiff University which has set up a smart home based IoT testbed.
The scope of the project includes identifying the type of device and not the individual device being read.
An attacker is assumed to have no knowledge of the devices on the network but by obtaining the network traffic and predicting the devices present, the attacker is violating the privacy of the user/s. But how the attacker gains access to the network is not important for the purposes of this project.

The desired outcome for the project is to be able to read unseen data from a mystery device using a pool of previously seen devices. It should correctly reveal what type of device is present through analysis of the network traffic and then classification using a trained model with a high level of accuracy.

2 Background

This section describes the current state of research and gives some background information on the areas that a useful to be informed about, and support the understanding of the project.

2.1 Security Risks of IoT Devices

IoT devices are becoming more prevalent in everyday life with the demand for efficiency, accuracy and cost saving whilst reducing the need for human involvement of everyday processes. The overall aim of IoT devices is to have the physical objects around us connected to the internet and controllable remotely. Some examples of IoT devices can include fridges, insulin pumps and smart meters. The number of IoT devices around us is on the rise with a predicted 75.44 billion connected devices by 2025 (Figure 2.1). [29]
However these new devices have brought about many security risks which can range from attackers

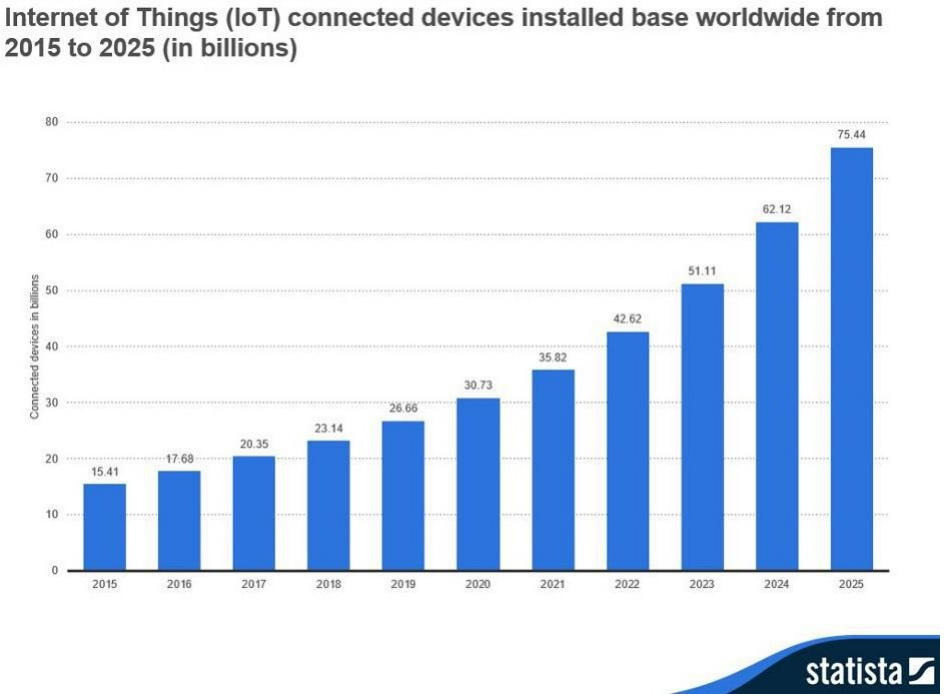


Figure 2.1: Statista - number of connected devices worldwide [29]

being able to turn your lights on and off to being able to spy into your home; moving security risks from online threats to physical ones.
In 2014 a family was using a Foscam baby monitor camera [16]. The couple were awoken by a strange man's voice in their home. They went into their daughters room to find that the camera was moving independently and a voice also emitting from the device.

Research by Nitesh Dhanjani [7] Showed that if the attacker knew that the baby monitor was

in the house, all they had to do was browse a URL, open a file in a hex editor and obtain the username and password from there.
The project demonstrates how easy it is to determine what types of devices are present on a network.

2.2 Device Fingerprints

A device type fingerprint takes into account multiple different factors to determine the type of device connected to the internet with some level of certainty. It involves recording factors such as the transmission rate, the quantity of data transmitted over a network and other leaked information.

Attributes such as IP and MAC addresses cannot be used as they do not provide any insight to the type of device and are easily forgeable by attackers.
Attributes such as those used by the IoT Scanner [1] for example, cFrames (control), mFrames (management) and dFrames (data) can be used to build up fingerprints for devices.

2.3 Data Collection and Analysis Tools

This section in the report discusses some tools and techniques that could be used to analyse the network traffic of devices and subsequently produce a device type fingerprint.

2.3.1 Passive Monitoring

Passive monitoring is a method for observing the network traffic which differs from active monitoring because it involves listening to the data and packets that are already on the network. Active monitoring on the other hand involves injecting and changing the network traffic. [31]
In this project passive monitoring is used to capture and analyse the packets that are sent and received from a device.

2.3.2 WiFi Pineapple

The WiFi Pineapple is an access point that can be used to capture network traffic and execute various network attacks as it was originally created for penetration testing. [19]
It is OS agnostic and displays information in an online GUI which would satisfy my additional requirements of providing an easy to use tool available on various operating systems.

Its capabilities would allow the interception of the network activity from the IoT devices by exploiting the automatic connections to known access points specified by SSID.

Although the device is widely available, the cost is approximately £100 [15] meaning that it would not satisfy the additional requirements completely.
Additionally, the capabilities of this device extend far beyond what would be required for the project. However it is interesting to note the possibilities of this device particularly the ability to passively intercept traffic from any device whilst being discrete and portable.

2.3.3 Wireshark

Wireshark is a free and open source network packet analyser. It works by capturing live traffic across a network which can be viewed in terms of many attributes such as the protocol used, IP addresses and length of the packets. The information is displayed aesthetically in a GUI with colour coding for quick analysis. It also runs on Windows, Linux, macOS, Solaris, FreeBSD, NetBSD, and many

others which satisfies all the additional requirements [32].

Wireshark would support the project by offering a means to capture and view the traffic of the IoT devices visually as well as being able to extract the information in the form of a XML, text or CSV file which makes further analysis simple.

Another useful feature is the filtering which can be used for cutting down a capture time or singling out a device where multiple connected devices are present.

Similar to the WiFi Pineapple, an attacker can sniff the traffic on a network without being detected.

2.3.4 Scapy

Python is a high level programming language that could be used to write a program that analyses the contents of a file. Scapy is a library within python that is used for packet manipulation [12]

Given the amount of packet information available in the packet headers and the ability to export this information from the pcap file, it is quite possible that standard Python libraries alone are sufficient for extracting device fingerprints without the need of the specialised library. As with the WiFi Pineapple, its capabilities extend beyond what is required for this project as it is mainly used for forging and decoding packets.

However its use in the project depends largely on what attributes and tools turn out to be useful. If the standard Python libraries are sufficient then there is no reason to use it.

2.3.5 Pandas/ MatPlotLib

Pandas [14] and Matplotlib [11] are Python Libraries used for data manipulation and visualisation. They provide a way to represent results in the form of various different graphs making comparisons and analysis undemanding.

They are useful in this project for spotting trends within the data that has been collected and aids in understanding whether an analysis of the device data has been effective or not. The simplicity of the libraries means that only a few lines of code are needed and it can be integrated directly into a Python analysis if needed.

2.4 Machine Learning

Machine Learning is used to create a model that extracts previously unknown and potentially useful patterns from the data. It takes the analysed data and classifies it as one of the devices with a certain degree of accuracy.
Machine learning was elected to classify the devices that are intercepted because a model is needed to make accurate predictions; a human might not be able to look at the data and accurately classify it. Additionally, having a model will also be much quicker with large quantities of data.

2.4.1 Classification Algorithms

Classification maps individual data items into predefined non overlapping classes. They are well suited to the project as the data being dealt with is labelled and in the form of a number of numerical attributes and a class value. [28]

2.4.2 Clustering

Clustering identifies a finite set of clusters to describe data. The algorithms work by grouping a set of objects and finding whether there is some relationship between them. [28]
For this project the number of potential devices is known but most clustering algorithms deal with unlabelled data and so are not useful with this type of data.
Its use depends on how well all the attributes are correlated together or how distinct the clusters are.

2.4.3 Artificial Neural Networks

An Artificial Neural Network (ANN) forms a network where information moves from the input layer to the hidden layers and then to the output layer. [18]

It is useful as a predictive model because they associate input and output nodes through pattern recognition. For my project, that would mean finding frequent links between the attribute values that make up the device fingerprint and the class (type of device). It is also used by A. Selcuk Uluagac et al in [10].

Many libraries are available to assist in building an ANN such as Tensorflow, Theano and Keras. They are available on all platforms and simplify the build of a neural network by doing most of the background work.

2.4.4 WEKA

WEKA provides a suite of machine learning algorithms [23]. It is well suited to the project because it has a GUI for ease of use, its free and available on multiple platforms. It is also advantageous to use WEKA over building one model considering it has the capabilities to build and test a number of algorithms at the same time for comparison. It would save time building and perfecting a final model and also means that a larger number of algorithms can be considered.
The following provides some background on the algorithms that were evaluated.

Multilayer Perceptron

Multilayer Perceptron (MLP) is a feed forward ANN in WEKA where each layer receives its input from the previous layer and forwards its output to the next layer. The nodes in this network are sigmoid by default and trained using backpropagation to classify. [18]

ZeroR

ZeroR is the simplest algorithm as it picks the class value that is the majority in the dataset and gives that for all the following predictions.
Given that all of my classes are balanced (240 instances), it is expected that it picks the first class value “Amazon Echo” and gives that as the answer for all the predictions. Therefore the expected result is (240/1680). ZeroR is chosen as a baseline where other algorithms are expected to exceed its accuracy. [2]

Naive Bayes

Naive Bayes is a probabilistic classifier which is based on Bayes’ Theorem. It assumes that the value of a feature is independent of the values of the other features with the same class value. For this project it would assume that the ’Smallest Packet Size’ and ’Average Packet Size’ are independent of each other for example [27].

Logistic Function

The logistic function builds and uses a multinomial logistic regression model. It generalises logistic regression for multiclass problems. [25]

SMO Function

The SMO function normalises all the attributes by default. To solve multi-class problems it uses pairwise classification which converts a multi-class problem into a series of binary ones. [22]

J48

J48 is an implementation of the C4.8 algorithm in Java. It is a fast binary decision tree learner that builds a tree using information gain/variance and prunes it using reduced-error pruning. [17]

Random Forest

Random Forest is an ensemble approach that builds multiple decision trees and amalgamates them together to get a more accurate prediction than using a single tree. Classes are determined by a majority vote from all the trees.
The benefits of taking a majority vote are that random errors cancel each other out and correct decisions are reinforced. [8]

REP Tree

Reduced Error Pruning Tree (REPT) creates multiple trees in different iterations and then selects the best one to be the representative. It builds a decision tree based on the information gain or reducing the variance. [21]

2.5 Related Work

Research into device and device type fingerprinting has been looked into in various research papers, however the aim of this project is to address the problem of knowing what types of devices are connected to a network whilst also providing a solution that is free and simple to use.

Some methods have involved actively fingerprinting devices such as [13]. Sergey Bratus et al. try to find information about the chipset, firmware or driver of a device by recording its responses to a set of altered frames. This project will focus on passive monitoring as the device and owner of the device should be unaware that it is happening.

[1] looks at a similar idea but specific to separating actively streaming cameras from non camera devices. I want to take this idea further and look at determine the type of any device being read. Their IoT scanner achieves passive monitoring and looks at attributes such as frames and bytes sent or received from devices across a network. These are the types of attributes that were considered when producing the device analysis for this project.

[10] looks at different ways to separate devices. A. Selcuk Uluagac et al. take the differences in the internal composition of devices such as variations in clock skew where the clock signal arrives at different components at different times.
GTID also gets additional information such as the vendor which is not relevant to this project as a pcap file is needed and this contains information about the vendor.
However, GTID also accomplishes passive fingerprinting and uses an artificial neural network to

classify the results which are aspects that are considered during the project.

All these research papers use different methods and metrics to classify their devices. This project focuses on the information available in the packet headers.

3 Approach

This section outlines the tools and techniques used in the project and a method for reaching the final results.

Many methods are available at each step of the project.This section of the report explains the tools and techniques that were chosen and justifications for the choices in relation to the project.

The data being used is from the IoT testbed [3]. The data is collected over consecutive days using Wireshark by Eirini Anthi in [4] providing a number of data files over 24 hour periods in the form of pcap files.

The software used for displaying a capture of the IoT device network traffic is Wireshark. It provides a useful GUI where the user can browse the packet information and IP addresses that are communicating across the network. This makes it easy to select an IP address and the filter functionality provides a simple command to use just the relevant packets for that IP address even if multiple devices are communicating during the capture. The software is also free and runs on multiple platforms.

The traffic is passively collected and the state of the intercepted packets (encrypted or decrypted) does not matter. There is no need to change the contents of the traffic and all the information needed for the purposes of classifying the devices can be extracted from the packet header.

Following this a Python script made with version 3.6.2 is used to analyse the extracted data and export the key information in a new CSV file. The script is developed originally, consisting of standard Python 3 libraries. It takes the packet header information and produces an analysis for every 6 minute interval of data where each interval makes up a record in the analysis. The attribute headers and class values are the same to get a consistent format each time and to be in a form that WEKA can take.
Python provides a favourable analysis as its simple to use, yet still fulfills the requirements of the analysis which produces the device type fingerprints for the devices. Furthermore, I have practise using python over the course of University, so no time was needed to become acquainted with the language.
Pandas and Matplotlib are useful for the visual representations of the data. They provides a variety of different ways to display the data such as scatter plots, histograms, line graphs, box plots etc All of which are straightforward to build from the ground up and tailored for specific data sets.
Furthermore, Python, Pandas and Matplotlib all have free public availability.

For the classification, WEKA version 3.8.2 is used to compare a comprehensive collection of algorithms and then build the final model.
WEKA is valuable because it provides multiple statistics for the models and compares them directly to each other.
Models can be built in seconds and saved so that they can be loaded onto other computers with WEKA installed to classify unseen data. The software is free and runs on almost any modern computing platform which means the models are not limited to certain users.
WEKA is easy to use as the GUI provides an uncomplicated way to locate and change hyperparameters to optimise algorithms. Testing and making predictions is also quick to complete. The only downside is that the test file has to be in a specific format eg headers the same as training data and classes with a '?'. But this is overcome in the project by the python script producing this format

automatically.
The general process and flow of the data is outlined in Figure 3.1 .

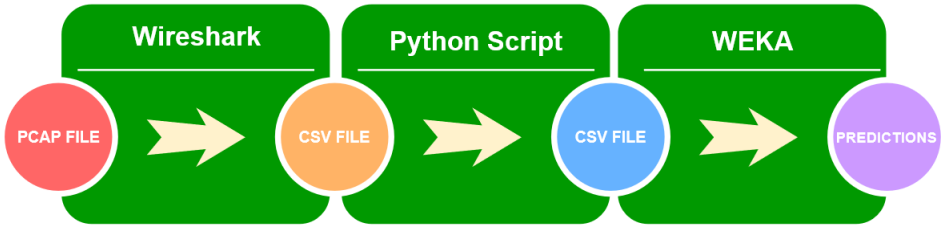


Figure 3.1: Outline of the process

3.1 Method

- View the data from the pcap file in Wireshark. It should contain packet information of wireless IoT devices sent and received from an access point.
- Apply a display filter by using one of the IP address to get data for just that device eg:
  ip.addr == 192.168.200.128
- Export Packet Dissections, As CSV
- Run the Python script in the command line using this exported CSV file as:
  python packetAnalysis.py device.csv
- Each 6 minute chunk of data is analysed as a record and exported into a new CSV file.
- Using WEKA 'Explorer', on the 'Classify' tab, load the trained model in the 'Results list'.
- Open the analysed CSV file is opened as the 'Supplied test set'
- Change the display options under 'More options', 'Output predictions' set to 'PlainText' to see the predictions for each of the records in the CSV file in the 'Classifier output' window.
- On the model, 'Re-evaluate model on current test set' to produce the predictions for the new data.

4 Data Preprocessing

This section describes how the information that constitutes the device type fingerprint was selected and analysed.

4.1 Packet Analysis - Specification and Design

The packet analysis is performed by the python 'packetAnalysis.py' file to satisfy a large array of initial functional requirements which are potentially needed for the device type fingerprint.

- Analyse the packets within a given time frame.
- Calculate how many packets were transmitted in that time.
- Calculate the maximum length of a packet transmitted. (Bytes)
- Calculate the minimum length of a packet transmitted. (Bytes)
- Calculate the average length of the packets transmitted. (Bytes)
- Count how many different protocols were used.
- Record the most common protocol used.
- Calculate the smallest time gap between consecutive packets being sent or received. (Seconds)
- Calculate the largest time gap between packets being sent or received. (Seconds)
- Calculate the average time gap between packets being sent received. (Seconds)

All of the information needed is taken from the packet headers in the pcap files. Having many features means it is harder to make an estimator so subsequent analysis determines which of the variables has a greater influence in distinguishing devices and therefore decreases the number of functional requirements.

The visualisation with the pandas graphs are to aid with the analysis of the packet information and functional requirements by providing figures on each feature of the devices. Files such as 'Analysis1', 'Analysis2', 'Analysis3', 'Analysis4', 'Analysis5', 'Eval1attributes', 'Eval1maxdepth', 'Eval1iterations' and 'Normalisation' provide the bar charts, box plots, histograms, scatter plots and line graphs needed.

4.2 Packet Analysis - Implementation

The packet analysis is implemented in Python 3 as 'packetAnalysis.py'. Firstly, it reads in the packet information from a CSV file, then it works out the consecutive intervals of data to use for each record of the analysis (Figure 4.1).

The main methods for the calculations are

- 'smallAndLarge()' which calculates the smallest and largest values in a given list.

```
# Get information within the time limit of the supplied csv
while timeLimit <= float(packetTime[-1]):

    # Break into 6 minute intervals
    timeStart = timeLimit
    timeLimit = timeLimit + 360

    # Get packet times within the correct range
    new = []
    for i in packetTime:
        if float(i) >= timeStart and float(i) < timeLimit:
            new.append(i)
    packetStar = packetLim
    packetLim = packetStar + len(new)
    if new:
        packetNumber = len(new)
        statements(packetLim, packetStar)
```

Figure 4.1: Selecting time intervals

- 'average()' which calculates the mean.
- 'burstiness()' which calculates the gaps between the packets being sent and received in the time frame and then uses smallAndLarge() and average() to calculate the 'Smallest Gap' 'Largest Gap' and 'Average Gap' (Figure 4.2).
- For some of the intermediate testing a method called protocols() is used which calculates the most commonly used protocol and the number of different protocols used but this does not feature in the final 'packetAnalysis' file (Figure 4.3).

```
def burstiness(l, lim, star):
    n = 0
    m = 0
    l = l[star:lim]
    temp = []
    for i in range(0, len(l)-1):
        new = float(l[i+1]) - float(l[i])
        temp.append(new)
    if not temp:
        m = 0
        n = (0, 0)
    else:
        m = average(temp)
        n = smallAndLarge(temp)
    return n, m
```

Figure 4.2: Calculating gaps between packets sent and received

Once the key information needed to meet the functional requirements has been calculated. The output is both printed to the command line and also exported to a new CSV file. This makes the information easily accessible for use in the next phase where this data will be classified.

The graph plots work by taking the analysis output as a dataframe and plotting it in the style of the desired graph. For multiple plots on one figure, it is divided up into subplots. An example from 'Analysis1.py' is shown in Figure 4.4 .

The remaining Pandas and Matplotlib files all work in a similar way but are tweaked slightly depending on the input and desired output.

```
def protocols(l, lim, star):
    prot = {}
    for i in l[star:lim]:
        if i not in prot:
            prot[i] = 0
        prot[i] += 1
    amt = len(prot)
    most = max(prot, key=prot.get)
    return amt, most
```

Figure 4.3: Calculating the number of different protocols used and the most common

```
file = pd.read_csv(sys.argv[1])

fig = plt.figure()
fig.suptitle("NC200 Camera - 15min Data")
titles = ["Total # Pkts", "Largest Pkt", "Smallest Packet", "Average Packet Size",
          "# Protocols Used", "Largest Gap", "Smallest Gap", "Average Gap"]

pos = 331
count = 0

for i in file:
    ax = fig.add_subplot(pos)
    file[i].plot.bar(color='darkgreen')
    ax.title.set_text(titles[count])
    plt.tick_params( axis='x', which='both', bottom=False,  top=False, labelbottom=False)
    count += 1
    pos += 1

plt.show()
```

Figure 4.4: Analysis1 - Building graphs

Testing involved looking at multiple aspects of the analysis. Initially the tests prove that the 'packetAnalysis' file is functioning correctly, then the output is analysed to discover which attributes work best.

Testing whether the analysis calculations were correct involved comparing the calculations made by the python script to those done manually on smaller sized files. Confirming that the data produced was accurate ensured that the script met all the functional requirements for the analysis tool. For the graphs, trial and error was used to achieve a desired output.

The usefulness of the analysis output was tested by gathering a variety of pcap files and devices. With the help of the visualisation, the attributes being used could be changed and altered to get an optimum analysis that would distinguish the devices the most clearly.

4.2.1 Analysis 1. Collection Time

An assumption would be that the longer the time period the better the results but because of the attributes that were used, taking an analysis from the entire file at once was not useful because any outliers will become the values for 'Smallest Packet Size', 'Largest Packet Size', 'Smallest Time Gap' and 'Largest Time Gap'. Furthermore, just taking the first part of the file meant that a lot of the data was wasted and only one record was being produced each time.
A solution to this was producing multiple records per file which then brought about the issue of what time limit to choose to get optimal results.

It is also beneficial to choose a set time limit because the traffic volume (total number of packets exchanged) will vary significantly if the observation window is changed each time. Therefore, choosing to use set sized blocks removes this variation.

All the possible variables were calculated with data from the 6th (06Data) and four devices; NC200 Camera, Smart Plug, Amazon Echo and Belkin Netcam. The first analysis contained just four devices to make the output more manageable as the number of attributes was at nine and four different intervals were being tested:

- 30 second intervals - yielding 500 entries per device
- 2 minutes - 500 entries each
- 6 minutes - 241 entries each
- 15 minutes - 97 entries each

The number of entries was limited to 500 for 30 seconds and 2 minute intervals as too many would not show up in the graph plots.
Observations from trying all the variables are that the 'Most Common Protocol' attribute was redundant as majority of the data that was looked at for the four devices had TCP as its main protocol and therefore would not be useful in differentiating the device type. The other attributes had potential as they had much clearer distinctions.

The gaps in the records on the graphs (particularly for the 30 second intervals) (Figure 9.1) shows where no data was transmitted during that time interval but the decision to eliminate this from future analysis is based on how it would influence classification in the second part of the project. If multiple devices have 'zero' entries then the prediction will always be the device that had the most 'zero' entries during training which would distort the accuracy of the models.

For this part of the analysis some of the anomalies/outliers were removed because the purpose of the graphs is to show how consistent the majority values are and what the distribution is like for each attribute when taken at different time intervals. The outliers resize the axis too much so can be discounted for now. Some anomalies were removed from the following graphs:

- Amazon Echo, 30 second interval - Smallest Gap attribute
- Camera, 30 second interval - Average Gap and Largest Gap
- Camera, 6 minute intervals - Total number of Packets
- Camera, 15 minute intervals - Total number of Packets
- Smart Plug, 30 second intervals - Average and Largest Gap

Using 30 second intervals introduced a lot of variance and outliers which can be seen in Figure 4.5. Increasing the gap to 2 minutes showed more consistent patterns (Figure 4.6).

Both the 6 minute and 15 minute intervals had the most consistent values for all the attributes and do not change much between them.Figure 4.7, 4.8.
Given the choice between the two values 6 minutes would be preferred as having too big a gap emphasizes outliers eg smallest and largest values. Furthermore, having smaller time intervals means that less data is needed in a capture to produce the same number of records.

Therefore the time interval being used is 6 minutes. It is important for the remainder of the project and for future predictions that the intervals remain the same otherwise results could vary.

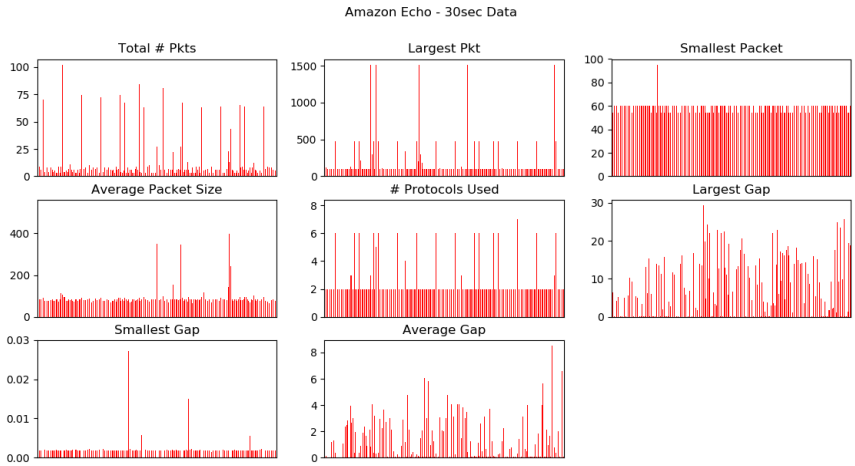


Figure 4.5: Amazon Echo 30 second intervals

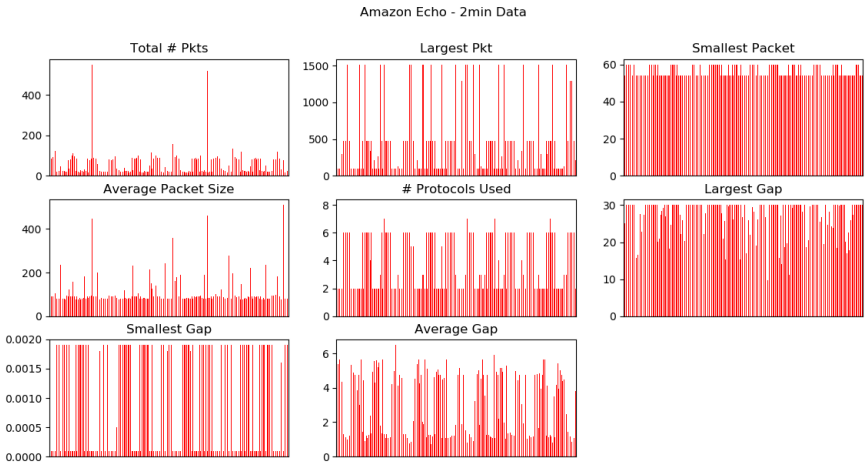


Figure 4.6: Amazon Echo 2 minute intervals

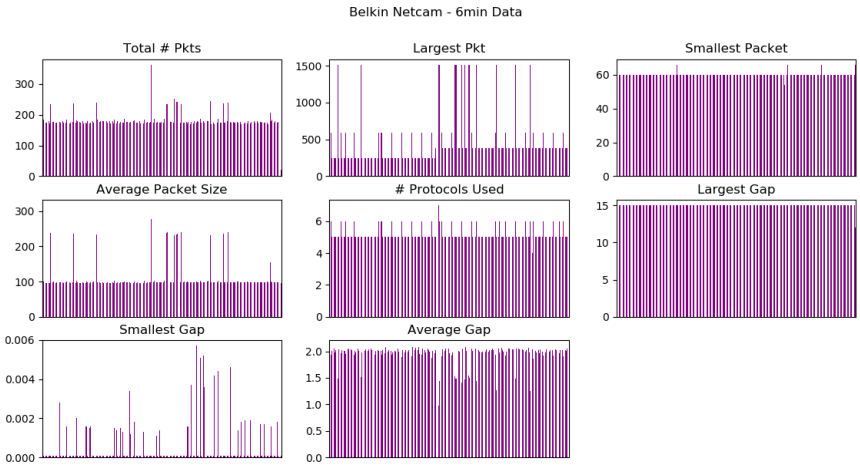


Figure 4.7: Belkin Netcam 6 minute intervals

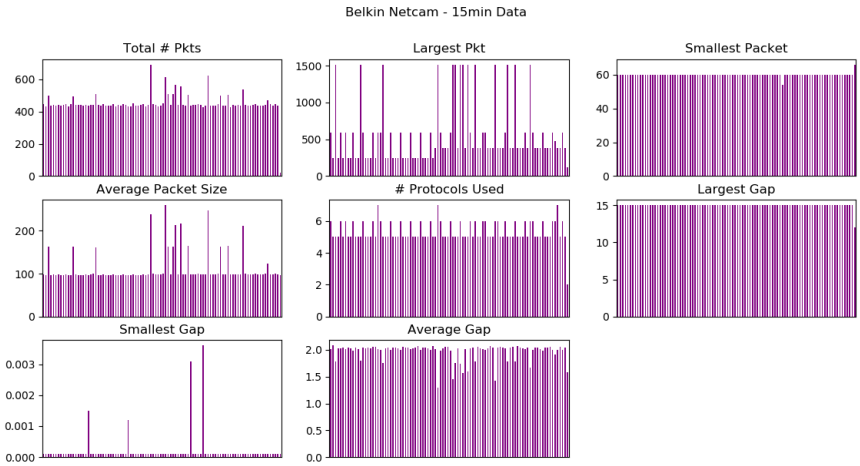


Figure 4.8: Belkin Netcam 15 minute intervals

4.2.2 Analysis 2. Decimal Places

The following comparison is used to find the optimum number of decimal places to be used to represent the smallest, largest and average time gaps which is particularly significant for the 'Smallest Gap' attribute as the values get extremely small.
The 06Data was used again with 6 minute intervals producing 241 entries for each of the devices. four, five, six and seven decimal places are being tested.

For the Smart Plug there were no significant differences between the values observed for the different number of decimal places as seen in Figure 4.9 .

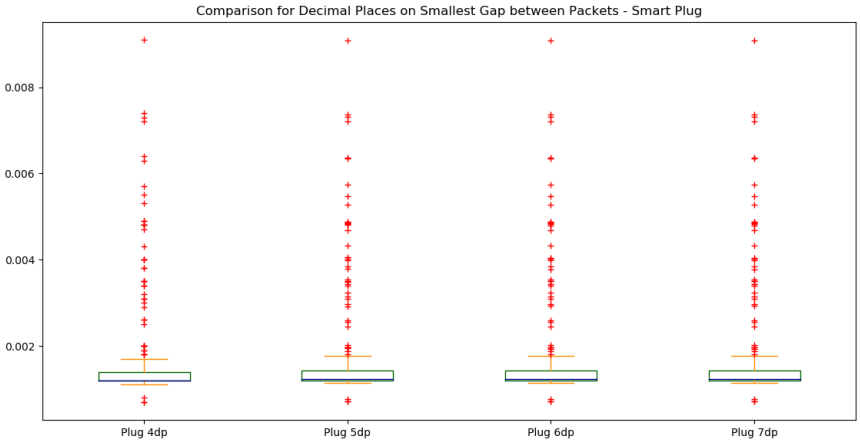


Figure 4.9: Smart Plug testing decimal places

For the Belkin Netcam four decimal places had a comparatively larger range of values (Figure 4.10) but the Interquartile Range (IQR) remained quite localised for all of the decimal places.

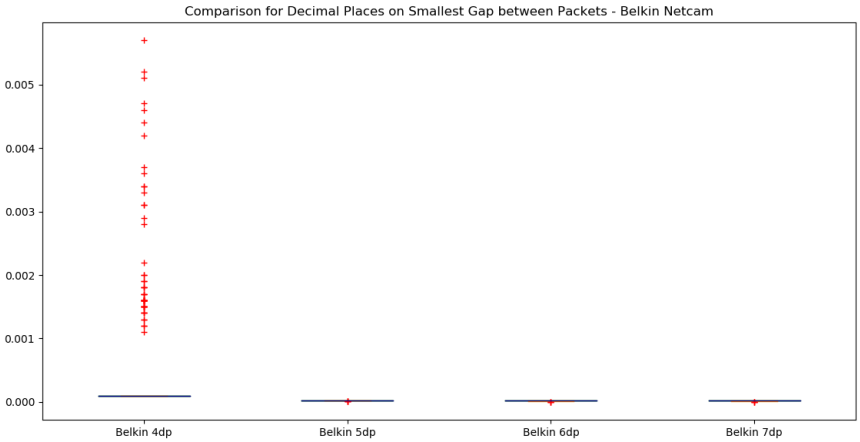


Figure 4.10: Belkin Netcam testing decimal places

The Amazon Echo showed four decimal places being the most consistent at 0.0001 but the modes of the others were much closer together at (five, 0.00002)(six, 0.000019) (seven, 0.000019) (Figure 4.11).

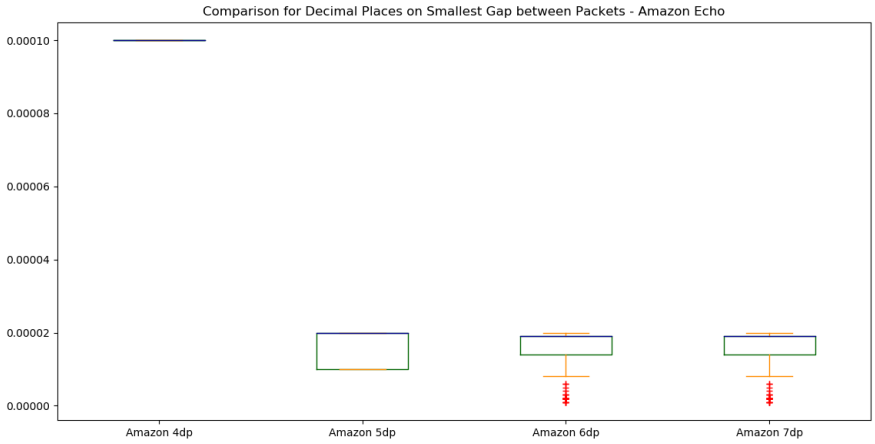


Figure 4.11: Amazon Echo testing decimal places

The Camera favoured four decimal places because of how close the IQR was and the consistency of the outliers (Figure 4.12).
Using six and seven decimal places seems to be pointless as they produce similar results in spread and consistency as using five decimal places.

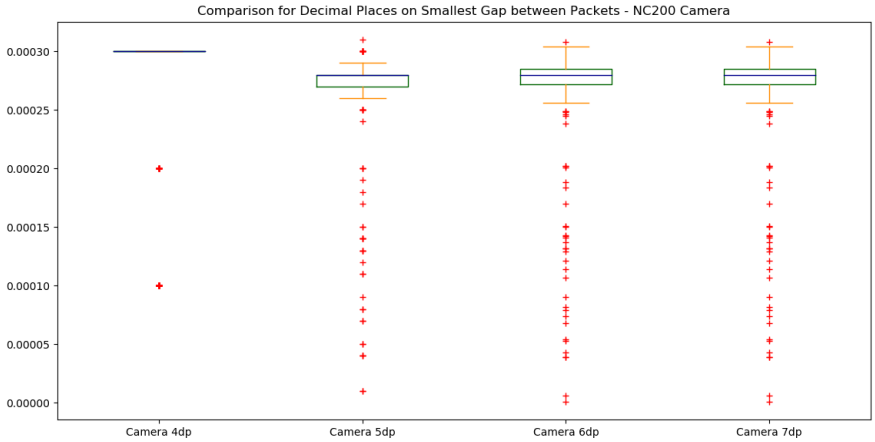


Figure 4.12: Camera testing decimal places

The values being used are decided as having four decimal places, although the Belkin Netcam would have been better with five decimal places because of the spread. The IQR's are still consistent so using four decimal places would not negatively affect the results too much.
Considering the Smart Plug produced similar results for all the values and the Amazon Echo and Camera would be better using four decimal places it seems to be the best option.

4.2.3 Analysis 3. Device Distribution

This part of the analysis also introduces the remainder of the devices (Hive, Lamp and Samsung SmartThings device). It tests to see if the distribution of values per device is consistent when larger amounts of data are introduced. Three days worth of data were combined together from the 6th, 16th and 21st to produce approximately 722 entries per device.
Binning was used for the histograms to reduce the amount of noise in the data which helps to observe the main distribution. Binning is the process of taking numerical values and transforming them into categorical ones by dividing the entire range into a series of consecutive and non-overlapping intervals, and then counting how many values fall into each of these intervals.

40 bins were used due to large range for each attribute. For example the range for the 'Total Number of Packets' was between 18 and 24,778 for the Hive. Having bins that are too small resulted in wide bars, where the distribution is less clear. Too large a value would make the bars small and less visible. Trial and error revealed 40 as a good medium for working with large amounts of data and different ranges for each attribute.

Some of the distributions were less clear than others, for example, the 'Average Gap Size' on the Amazon Echo has a more varied distribution with no obvious mode and the highest frequency not going much beyond 60 but the range was still small, mostly between 1 and 3 seconds (Figure 4.13).
From the histograms it is clear that the consistency carries through the majority of the data as

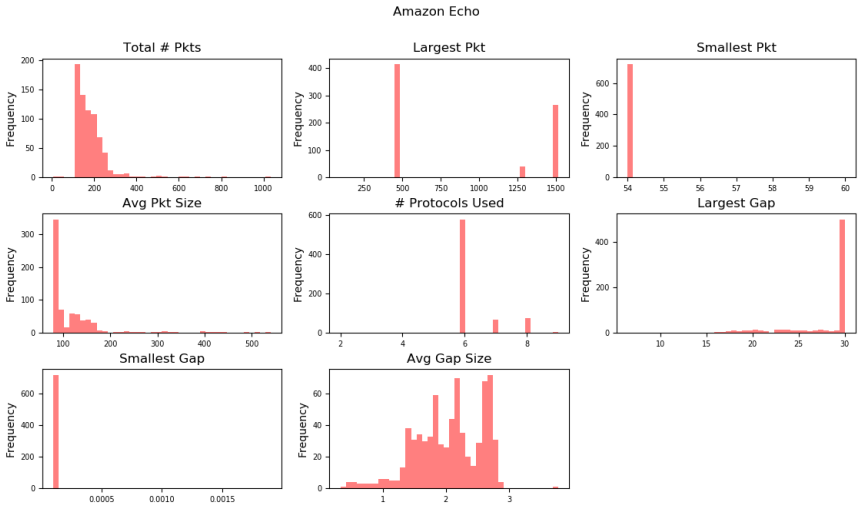


Figure 4.13: Amazon Echo three data sets combined

the modes of the devices are evident, particularly with the Belkin Netcam and Smart Plug (Figure 4.14, 4.15). This is promising because it means the data is more likely to be useful for classifying and making predictions.

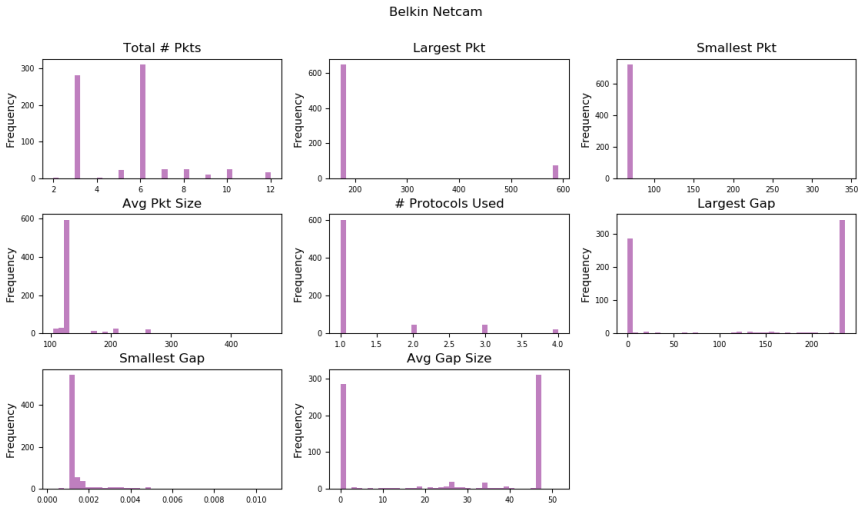


Figure 4.14: Belkin Netcam three data sets combined

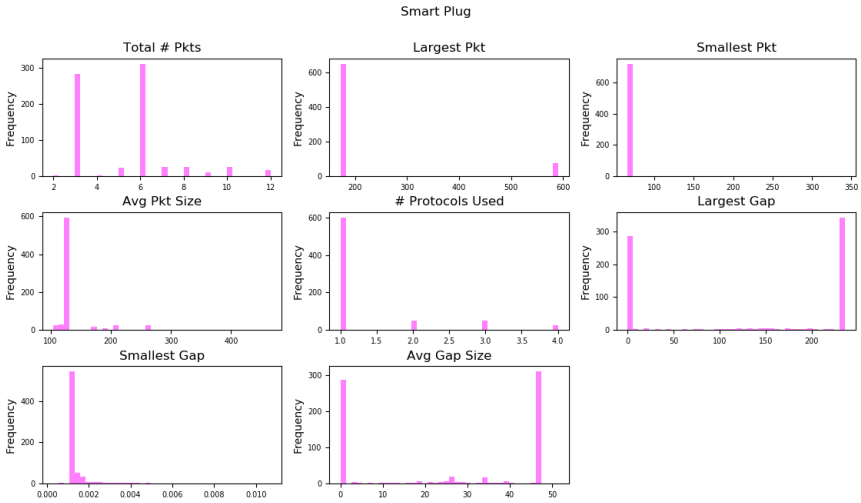


Figure 4.15: Smart Plug three data sets combined

4.2.4 Analysis 4. Device Comparison

In this section, the devices are combined together and compared to each other by attribute to see the overlap between devices. The aim is either to eliminate attributes that cannot distinguish the devices well enough, or keep attributes that show a more desirable distribution.
The same three data files are used as in the previous section but 100 bins are employed to show more precise values. The histograms are stacked and the different colours represent the different devices in each bin.

The results of combining the data do not show any definite distinctions so it would be impossible to determine a device by human eye or to use just one attribute.

For the 'Total Number of Packets', some devices worked well with clear distinctions eg Camera,

Plug, Lamp where the devices are isolated from others (Figure 4.16). Although there are other areas where the Amazon Echo, Belkin Netcam and Hive have a more even and wide spread distribution (Figure 4.17) so the usefulness of this attribute depends on the device.

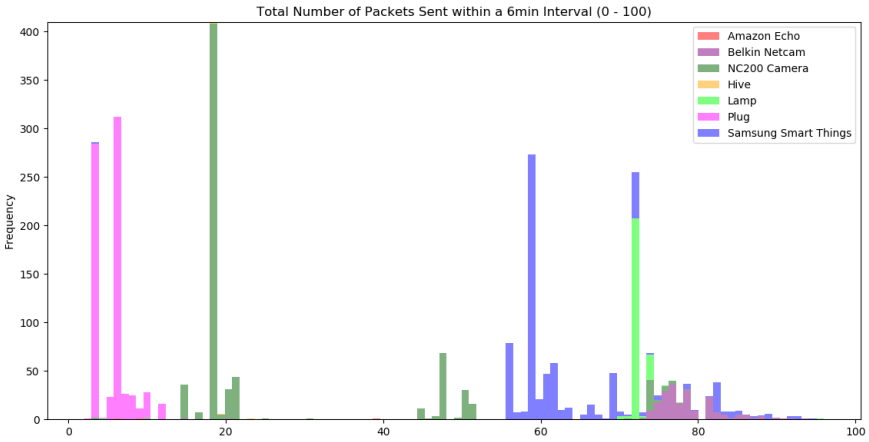


Figure 4.16: 'Total Number of Packets' device Comparison

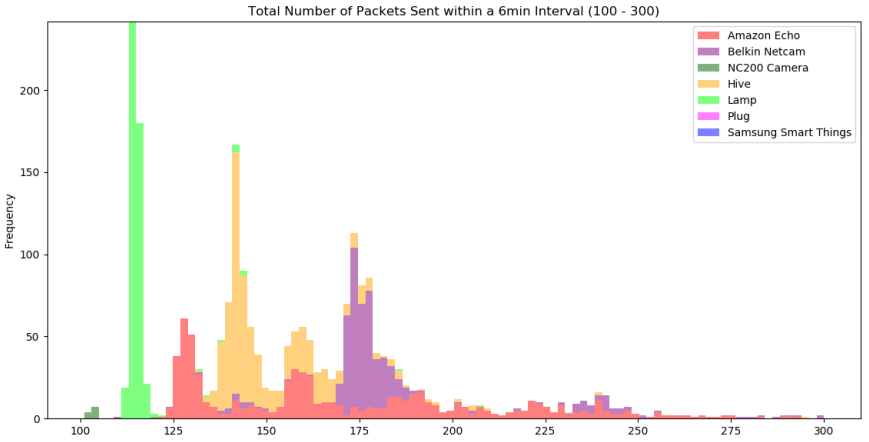


Figure 4.17: 'Total Number of Packets' device Comparison

For the 'Largest Packet' the lower end of the scale had clear peaks and distinctions between the devices. However, the upper end (particularly the value 1514 bytes) had multiple devices with this amount (Figure 4.18).

The 'Smallest Packet' attribute was very consistent for all of the devices which is a favourable quality, but the values for all the devices were consistent in the same places for example in Figure 4.19 which creates a lot of overlap.

The 'Largest Gap' seemed to be the best attribute as only the Hive and Amazon Echo overlapped (Figure 4.20).

The 'Smallest Gap' was not as clear where most of its values were 0.0001 For example the Amazon Echo had 721/723 of its records being this value (Figure 4.21). The Lamp was the only device to have a high value over 10 seconds so it would be easy to give a definite classification for (Figure 4.22).

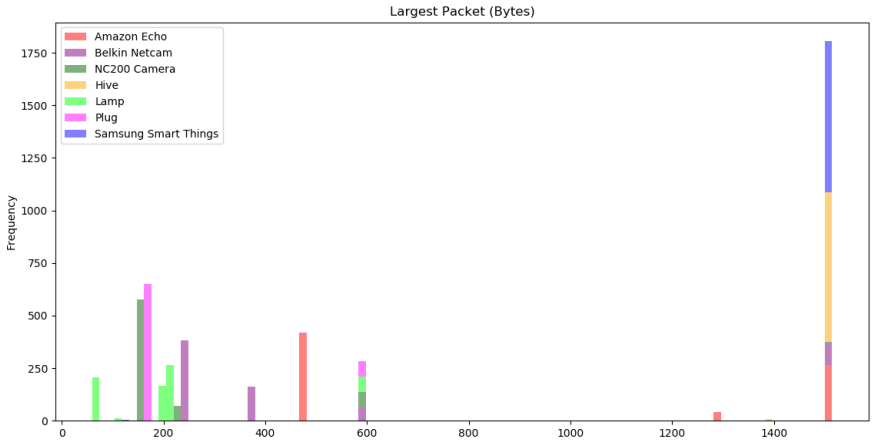


Figure 4.18: 'Largest Packet' device Comparison

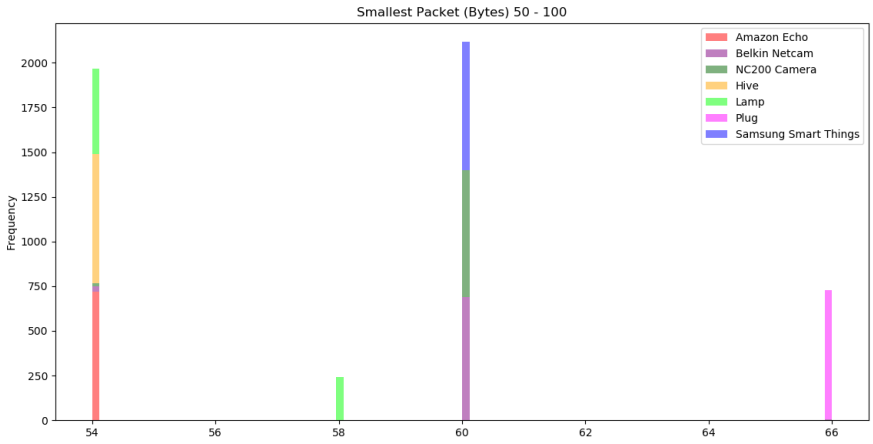


Figure 4.19: 'Smallest Packet' device Comparison (50-100 bytes)

For the 'Average Gap' clusters can be seen for the devices and for the Camera and the Lamp it would be quite effective in distinguishing them but for the values between 0 and 10 the results would not be so confident. Eg The Amazon Echo has a wider spread of lower frequency values (Figure 4.23).

The 'Total Number of Protocols' had a limited range of between 1 and 8 and considering most of the devices were spread over at least three of these values, a lot of overlap and spread occurs (Figure 4.24).

For the 'Average Packet Size' there is the occasional good value eg The peak for the Hive was at approximately 210 bytes, but there are a lot of the same values occurring (Figure 4.25).

All of the attributes had positive and negative parts to them but the least useful attributes were the 'Total Number of Protocols' and 'Average Packet Size'. Therefore these attributes were removed.

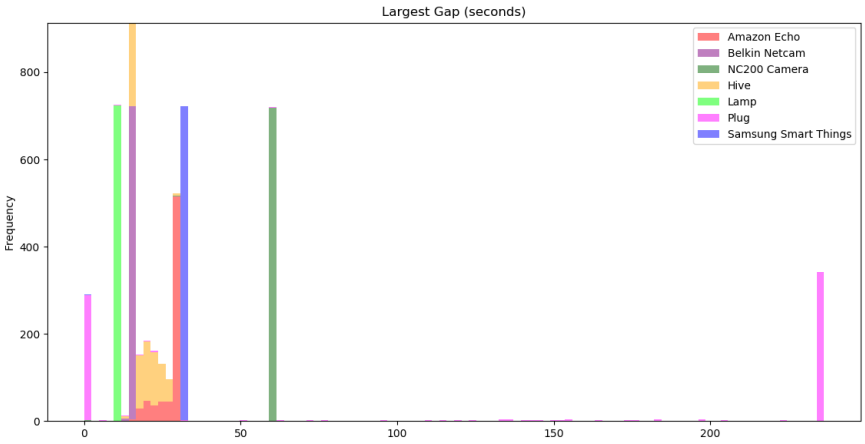


Figure 4.20: 'Largest Gap' device Comparison

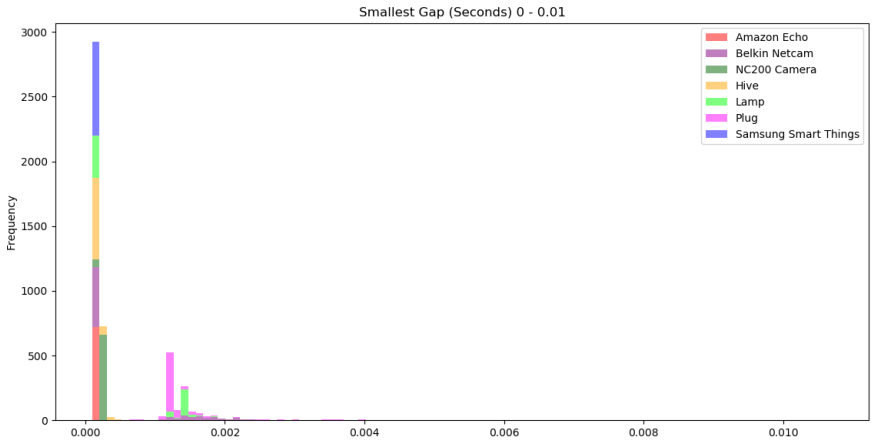


Figure 4.21: 'Smallest Gap' device Comparison

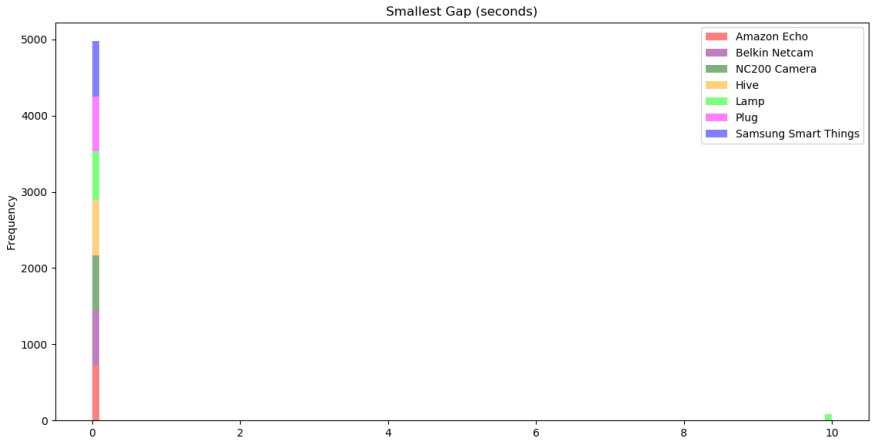


Figure 4.22: 'Smallest Gap' device Comparison

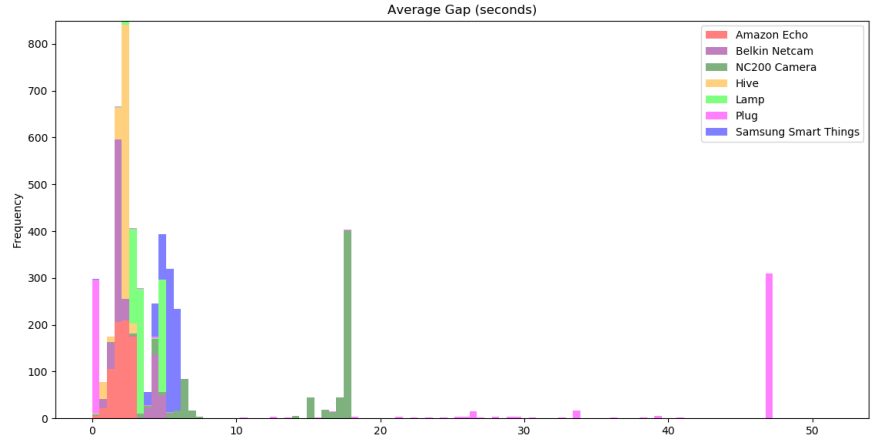


Figure 4.23: 'Average Gap' device Comparison

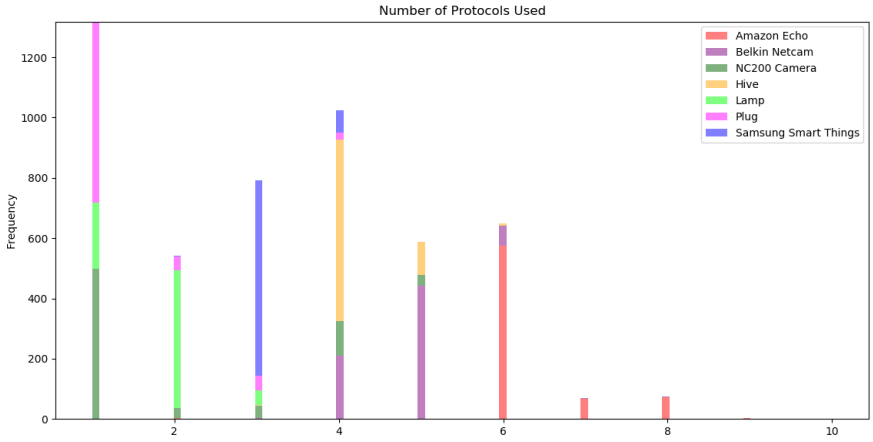


Figure 4.24: 'Number of Protocols Used' device Comparison

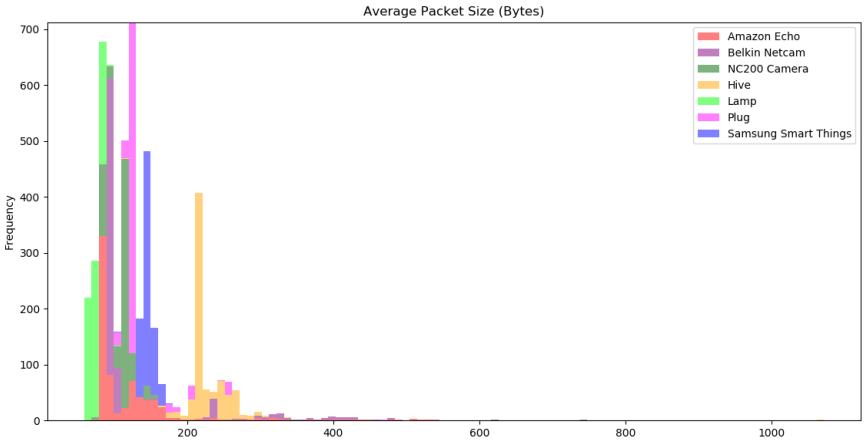


Figure 4.25: 'Average Packet Size' device Comparison

4.2.5 Analysis 5. Attribute Comparison

The scatter plots inspect the attributes by comparing them to each other. Separation for the devices would suggest a good classifier can be made from the data.
Data from the 6th 16th and 21st is used again with the 6 remaining attributes producing 5058 records in total.

Some of the scatter plots were reproduced with the outliers removed because the aim is to try and identify clusters in the data and having outliers extends the axis and squashed the data points.

- The values that were removed from the plots:
- Total Number of Packets - Hive and Camera values removed
- Smallest Packet - Plug value removed
- Smallest Gap - Lamp values removed

The 'Largest Gap' shows good clusters and that the Smart Plug data has much more spread than the others which is also mirrored in the 'Average Gap' plot (Figure 4.26, 4.27).

The relationship between 'Total Number of Packets' and 'Average Gap' forms a curve where the smaller the average gap between packets, the more packets that were sent (Figure 4.27).
The limited clusters within the data confirms that a model is needed to produce a classifier.

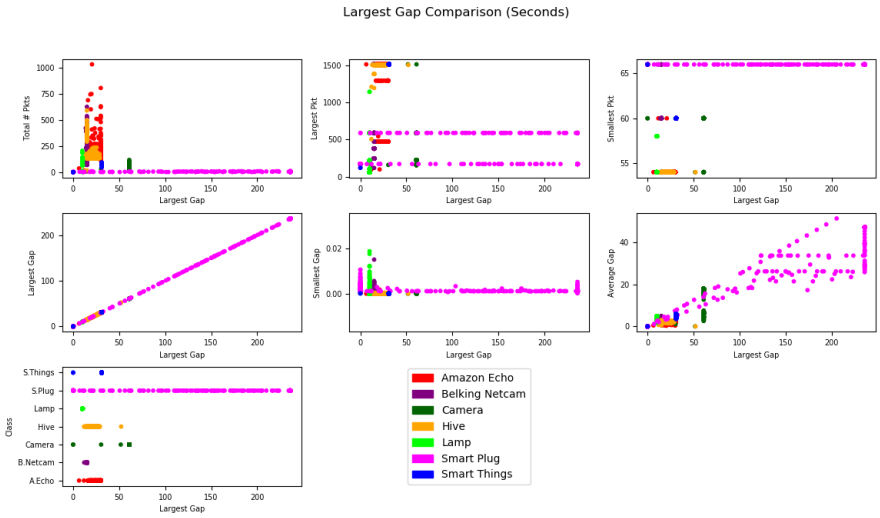


Figure 4.26: 'Largest Gap' comparison to remaining attributes

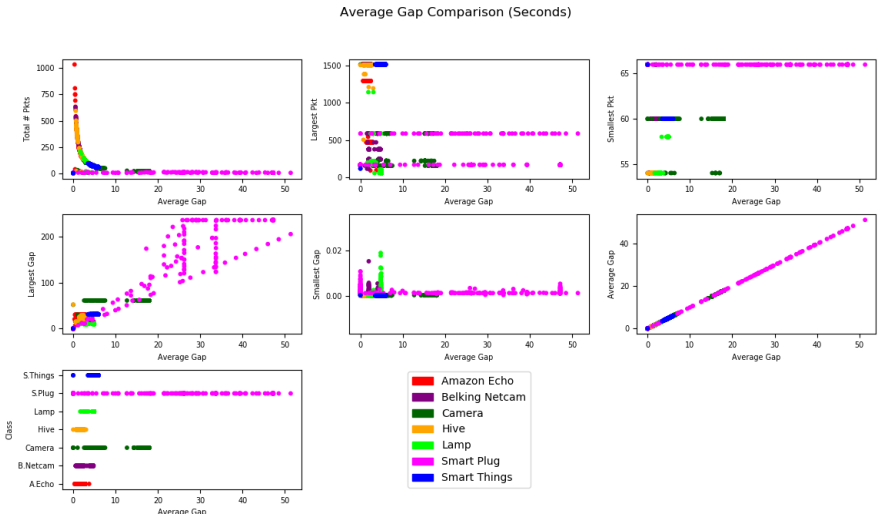


Figure 4.27: 'Average Gap' comparison to remaining attributes

5 Classification

Given that there is not always a clear distinction to be made from the data, machine learning is needed to create a model that is as accurate as possible to predict the class. Supervised machine learning is used on a variety of different decision based classification models.

5.1 Classification - Specification and Design

The objectives for the classification section of the project:

- A variety of algorithms are compared to each other on the training set.
- The top performing algorithms are selected and a number of hyperparameters tested.
- The prevailing algorithm is built in WEKA and the model saved.
- Unseen data can now be tested on the model.

5.2 Classification - Implementation

The models are built with the hyperparameters established and the training set supplied. After the build, the model can be saved and loaded onto any other computer with WEKA installed. It classifies unseen data that is given as a 'Supplied test set'.

The test results show the progression in selecting a model and justifications at each step of the analysis. The process is based off of the one performed by Jason Brownlee [5].

5.2.1 Algorithm Evaluation 1. All Algorithms

Given the nature of the problem, the algorithms selected can all solve multiclass classification problems. The following eight were compared:

- Multilayer Perceptron
- ZeroR
- Naive Bayes
- Logistic Function
- SMO Function
- REP Tree
- J48
- Random Forest

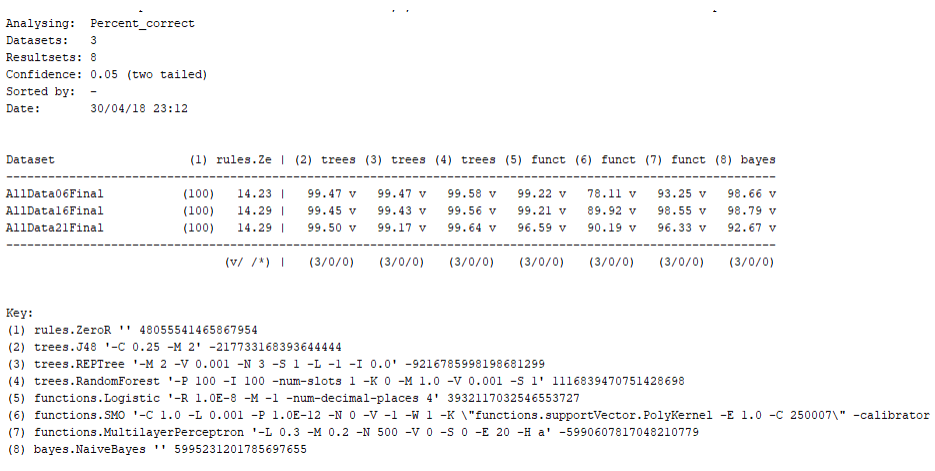
Each instance in the dataset describes measurements of the network traffic recorded for the seven devices. There are 6 numeric input attributes (one as a count, two measured in bytes and three measured in seconds) The task is to predict which device is being recorded for each of the instances.

To prepare the training set, the data was checked so that no missing values were present and the 'Class' attribute was created as a nominal value with seven possibilities. The training sets consisted of the data from the 6th 16th and 21st. To get a balanced data the device with the least records sets the maximum. In this case, the Smart Plug had 240 and the remaining devices had 241. A random number generator was used to pick one record from each device to be removed to make it even.

10 fold cross validation was performed in the WEKA Experimenter on all the algorithms and all three datasets combined. 10 fold cross validation involves dividing the supplied data set into ten pieces then taking nine to train and one to test. A different nine are then selected for training and one for test so each section will be a test set once. All the ten results are averaged as a way to evaluate results and estimate error [24]. This method is selected because using the data as training data alone will give misleading results for the evaluation.

2400 results were loaded due to the aforementioned eight algorithms each being evaluated one hundred times from the 10-fold cross validation (repeated ten times) for all three of the datasets.

The test results show a pairwise test which compares all the results to the results of the selected algorithm which is initially, the baseline ZeroR. The classification accuracy is calculated as the number of correctly classified instances divided by the total number of instances multiplied by 100.
The test was done with 'Paired T-Tester (corrected)' which checks if the average difference in their performance over the data sets is significantly different and not random. The comparison field is set at the "Percent Correct" metric which is an accuracy measure showing the percentage of correctly classified instances. The significance level is set at the default of 0.05 (5%) meaning any differences in the comparisons are statistically significant with a confidence of 95%. A "*" indicates that the results are significantly lower than the results from the baseline algorithm. No character next to the result in the table indicates that the results are not significantly different. Finally a "v" next to the results show that the algorithm outperformed ZeroR.
From the results in Figure 5.1 It is clear that all the algorithms performed better than ZeroR (which is to be expected).
The test was then repeated with the baseline algorithm being changed to the one that appears to



```
Analysing:  Percent_correct
Datasets:   3
Resultsets: 8
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       30/04/18 23:12


Dataset                   (1) rules.Ze | (2) trees (3) trees (4) trees (5) funct (6) funct (7) funct (8) bayes
-------------------------------------------------------------------------------------------------------------
AllData06Final           (100)   14.23 |   99.47 v   99.47 v   99.58 v   99.22 v   78.11 v   93.25 v   98.66 v
AllData16Final           (100)   14.29 |   99.45 v   99.43 v   99.56 v   99.21 v   89.92 v   98.55 v   98.79 v
AllData21Final           (100)   14.29 |   99.50 v   99.17 v   99.64 v   96.59 v   90.19 v   96.33 v   92.67 v
-------------------------------------------------------------------------------------------------------------
                               (v/ /*) |   (3/0/0)   (3/0/0)   (3/0/0)   (3/0/0)   (3/0/0)   (3/0/0)   (3/0/0)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) trees.J48 '-C 0.25 -M 2' -217733168393644444
(3) trees.REPTree '-M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
(4) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(5) functions.Logistic '-R 1.0E-8 -M -1 -num-decimal-places 4' 3932117032546553727
(6) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator
(7) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a' -5990607817048210779
(8) bayes.NaiveBayes '' 59952312017856976555
```

Figure 5.1: All algorithms compared to ZeroR in WEKA

achieve the best results which was Random Forest (Figure 5.2) to show which of the algorithms it outperformed by a statistically significant amount. The algorithms were then ranked (Figure 5.3) showing how many times each algorithm performed better (was more accurate) than the others on each dataset. To outperform another algorithm, the results must be statistically significant.

```
Analysing:  Percent_correct
Datasets:   3
Resultsets: 8
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       30/04/18 23:12


Dataset                  (4) trees.Ra | (1) rules (2) trees (3) trees (5) funct (6) funct (7) funct (8) bayes
-----------------------------------------------------------------------------------------------------------
AllData06Final           (100)   99.58 |   14.23 *   99.47     99.47     99.22     78.11 *   93.25 *   98.66 *
AllData16Final           (100)   99.56 |   14.29 *   99.45     99.43     99.21     89.92 *   98.55 *   98.79 *
AllData21Final           (100)   99.64 |   14.29 *   99.50     99.17 *   96.59 *   90.19 *   96.33 *   92.67 *
-----------------------------------------------------------------------------------------------------------
                              (v/ /*) |   (0/0/3)   (0/3/0)   (0/2/1)   (0/2/1)   (0/0/3)   (0/0/3)   (0/0/3)


Key:
(1) rules.ZeroR '' 48055541465867954
(2) trees.J48 '-C 0.25 -M 2' -217733168393644444
(3) trees.REPTree '-M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
(4) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(5) functions.Logistic '-R 1.0E-8 -M -1 -num-decimal-places 4' 3932117032546553727
(6) functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator
(7) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a' -5990607817048210779
(8) bayes.NaiveBayes '' 59952312017856976555
```

Figure 5.2: All algorithms compared to Random Forest in WEKA

```
Analysing:  Percent_correct
Datasets:   3
Resultsets: 8
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       30/04/18 23:12


>-<   >   < Resultset
 14  14   0 trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
 12  12   0 trees.J48 '-C 0.25 -M 2' -217733168393644444
 11  12   1 trees.REPTree '-M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -9216785998198681299
  5   8   3 functions.Logistic '-R 1.0E-8 -M -1 -num-decimal-places 4' 3932117032546553727
 -2   7   9 bayes.NaiveBayes '' 59952312017856976555
 -4   7  11 functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a' -5990607817048210779
-15   3  18 functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calil
-21   0  21 rules.ZeroR '' 48055541465867954
```

Figure 5.3: Algorithms Ranked in WEKA

The evaluation in Figure 5.2 and 5.3 are useful for narrowing down algorithms for further analysis. The results show that the ZeroR, Multilayer Perceptron, Naive Bayes and SMO Function performed the worst in all three data sets and the trees performed the best. Therefore the top four performing algorithms J48, REP Tree, Random Forest and Logistic Function will be analysed further.

5.2.2 Algorithm Evaluation 2. Model Optimisation

The next stage of the evaluation involves adjusting the hyperparameters of the selected algorithms to get an optimal result for each.

Some of the hyperparameters have not been altered, for example the 'Seed' parameter as any difference this makes to the results would be purely coincidental. The seed is used for randomising the data.

Random Forest

The depth of a node is the number of edges from the node to the tree's root node. (A root node will have a depth of 0). Increasing the maximum depth generally decreases the speed of algorithms but also improves the performance therefore any value selected will be a trade off. Maximum depth values between 1 and 15 are chosen, where 'infinite' denotes a tree of unlimited depth. Figure 5.4 shows that the maximum depth plateaus out to the unlimited value at around eight plus which implies that the depth either does not go deeper than 8 or adding more depth does not improve the performance. However, considering that the time taken to build and use the trees is so small, no limit will be put on the depth of the tree as the performance is more important at this point especially for such small changes in time taken.

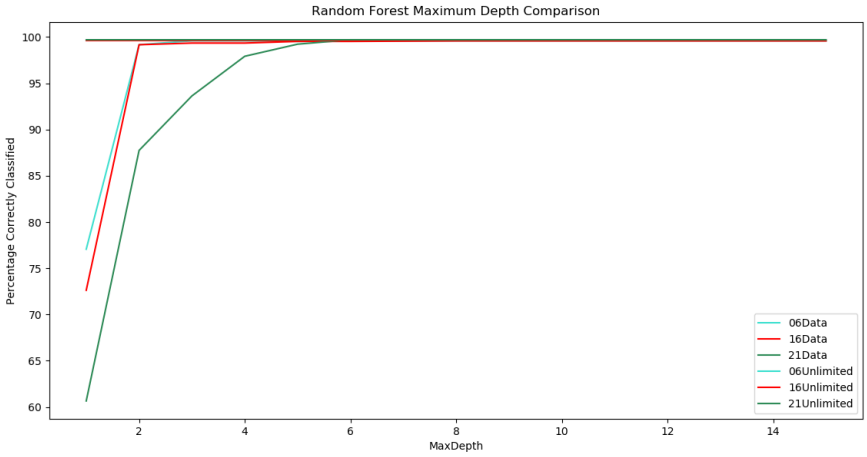


Figure 5.4: Random Forest max depth

The Iterations parameter refers to the number of trees being built in the ensemble 20 to 500 iterations were tested in intervals (Figure 5.5). 100 to 500 iterations showed the best and similar results in accuracy. Therefore 100 iterations were chosen as the time taken to build the model and make predictions is less.

The number of decimal places threshold used throughout the trees is tested on values between 0 and 4 because the data being used is limited to 4 decimal places. The performance was not affected at all with the different values and the time remained around the same no matter what the decimal places used on the split were. Therefore the default value of 2 decimal places will be used (Figure 5.6).

Feature selection was used to determine the impact of using certain attributes on the accuracy of the model (Figure 5.7). As a base comparison, no attributes were removed. For the data from the 21st the performance increased when removing the 'Largest Packet' attribute. For the data on the 16th removing the 'Average Gap' and 'Total Number of Packets' individually improved the accuracy of the model. For the data from the 6th removing the 'Total Number of Packets' and 'Largest Gap' attribute. Considering that removing the 'Total Number of Packets' resulted in improved performances on two of the data sets and did not affect the performance on the data from the 21st, this attribute will be removed from the final Random Forest model. However the other attributes will remain as the performance increase was not consistent over all three of the data sets.

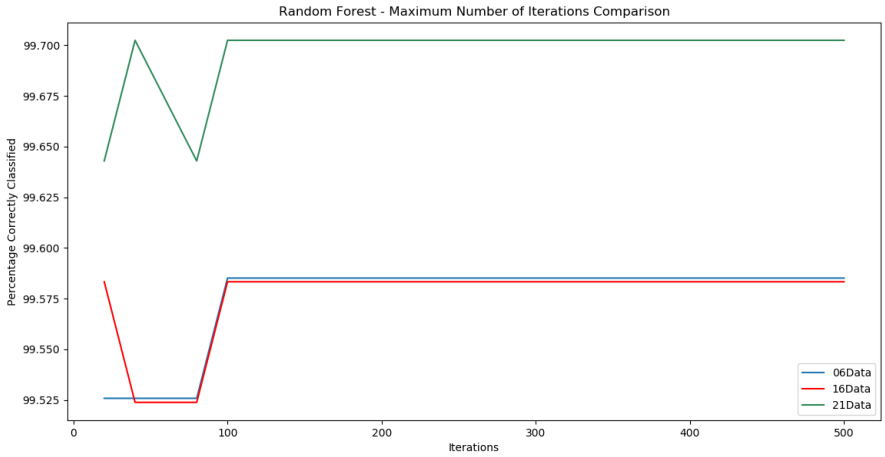


Figure 5.5: Random Forest number of iterations

	06Data		16Data		21Data	
Decimal	% Correct	Time Taken to Build Model (s)	% Correct	Time Taken to Build Model (s)	% Correct	Time Taken to Build Model (s)
0	99.5851	0.14	99.5833	0.14	99.7024	0.14
1	99.5851	0.19	99.5833	0.11	99.7024	0.14
2	99.5851	0.45	99.5833	0.13	99.7024	0.13
3	99.5851	0.16	99.5833	0.13	99.7024	0.13
4	99.5851	0.11	99.5833	0.11	99.7024	0.13

Figure 5.6: Random Forest number of decimal places used

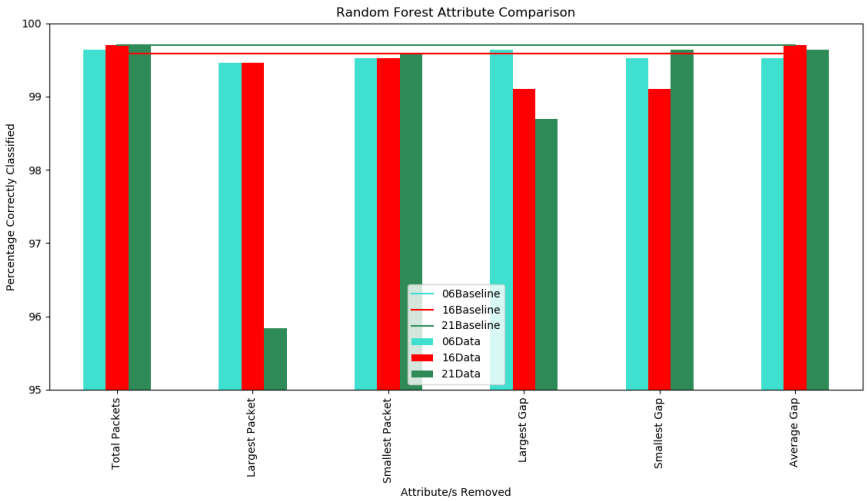


Figure 5.7: Random Forest feature selection

J48

Pruning is applied to the tree to reduce the size and to avoid unnecessary complexity in the model which could lead to overfitting to the training data. Ideally, the model is more efficient than the unpruned tree. (Subtree raising was left on when pruning was 'true').

Originally, performing cross validation produced no difference in the accuracy for different folds and pruning vs non pruning. This led to the data sets being combined so that the training that was done on the 06Data was tested on the 16Data then training on the 16Data test 21Data then training the 21Data test on the 06Data.
The number of folds determines the amount of data used for pruning. One fold is used for pruning, the rest for growing the rules. The number of folds tested were 2, 3, 5 and 8. When pruning was done, the structure of the tree was simpler, less leaves and a smaller sized tree. Except for the model trained on the 16Data and tested on the 21Data - which remained the same. The size of the tree refers to the number of nodes present. However the change in accuracy was negligible, showing that the data taken from each of the days had a lot of similarities (Figure 5.8)
Due to the fact that the model is more generalised, yet there is no effect on accuracy the J48 tree will be pruned but the number of folds made no difference so will be left on the default of 3.

2 folds				3 folds				5 folds				8 folds			
Trained on 06Data		Leaves	Size of tree			Leaves	Size of tree			Leaves	Size of tree			Leaves	Size of tree
Pruned	Tested on 16Data			Pruned	Tested on 16Data			Pruned	Tested on 16Data			Pruned	Tested on 16Data		
Yes	72.2619	12	23	Yes	72.2619	12	23	Yes	72.2619	12	23	Yes	72.2619	12	23
No	72.3214	14	27	No	72.3214	14	27	No	72.3214	14	27	No	72.3214	14	27
Trained on 16Data															
Pruned	Tested on 21Data			Pruned	Tested on 21Data			Pruned	Tested on 21Data			Pruned	Tested on 21Data		
Yes	90.7143	12	23	Yes	90.7143	12	23	Yes	90.7143	12	23	Yes	90.7143	12	23
No	90.7738	12	23	No	90.7738	12	23	No	90.7738	12	23	No	90.7738	12	23
Trained on 21Data															
Pruned	Tested on 06Data			Pruned	Tested on 06Data			Pruned	Tested on 06Data			Pruned	Tested on 06Data		
Yes	96.9643	13	25	Yes	96.9643	13	25	Yes	96.9643	13	25	Yes	96.9643	13	25
No	96.9643	17	33	No	96.9643	17	33	No	96.9643	17	33	No	96.9643	17	33

Figure 5.8: J48 Pruning Data

The 'MinNumObj' parameter refers to the minimum number of instances for data separation per branch. The confidence factor was set to 1.0 as in [9] to minimise the effects of post pruning however it would not eliminate it completely. As expected, the larger the 'MinNumObj' value, the worse it performed but not by a great amount. For example there was no change for data from the 6th when cross validation was used (Figure 5.9). The 'MinNumObj' parameter will be set to 2 because although increasing the number of instances would reduce the size of the tree, it decreases the accuracy so choosing a value involves compromise.

On-line Pruning J48

All3Data							
minNumObj	1	2	5	10	20	30	40
Training data	99.9802	99.9207	99.782	99.6235	99.5839	99.3461	99.0886
Cross Validation	99.7424	99.683	99.6235	99.5839	99.4848	99.0886	98.9697
06Data							
minNumObj	1	2	5	10	20	30	40
Training data	100	99.7629	99.5851	99.5851	99.5851	99.5851	99.5851
Cross Validation	99.4665	99.4665	99.4665	99.4665	99.4665	99.4665	99.4665
16Data							
minNumObj	1	2	5	10	20	30	40
Training data	99.881	99.8214	99.6429	99.2857	98.9881	98.9881	98.9881
Cross Validation	99.5238	99.4643	99.4048	98.869	98.8095	98.8095	98.8095
21Data							
minNumObj	1	2	5	10	20	30	40
Training data	100	99.7619	99.7024	99.6429	99.6429	99.2262	98.631
Cross Validation	99.5238	99.5238	99.5238	99.5238	99.4643	98.869	98.0357

Figure 5.9: J48 minimum number of objects

Feature selection showed that certain datasets did better without certain attributes (Figure 5.10). The only value being removed however is the 'Total Number of Packets' because similarly with Random Forest, the 6th and 16th did better without it and for the 21Data it made no difference to the accuracy of the model.

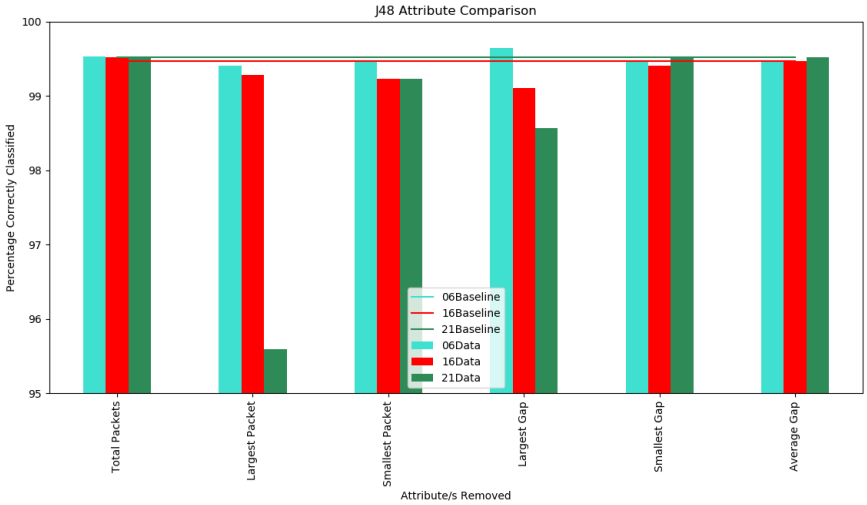


Figure 5.10: J48 Feature Selection

REP Tree

Looking at the 'MaxDepth' parameter produces similar results to Random Forest. Once the tree reaches a depth of about 4 - 6 it stops improving in accuracy. This is because the tree does not increase in depth when the maximum allowed is increased. Therefore the 'MaxDepth' will be set to 6 (Figure 5.11).

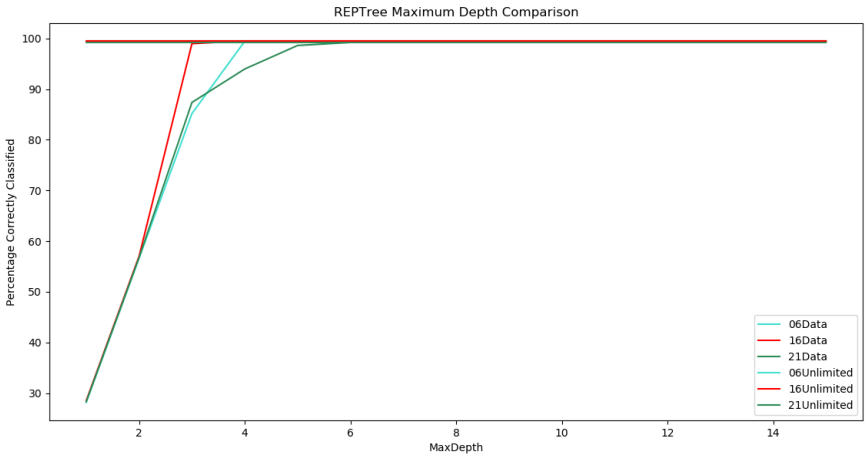


Figure 5.11: REP Tree maximum depth

Pruning and the number of folds were tested in the same way as the J48 algorithm. For the data trained on the 06Data and tested on the 16Data it was better to prune the tree in terms of accuracy. With the other two datasets the accuracy was improved when pruning was not done (Figure 5.12). For the cases when pruning was not done, the results were the same regardless of the number of folds, this is because if the tree is not being pruned, the number of folds does not matter.
For two of the sets not pruning the tree produced better results and the difference between pruned and unpruned was much larger than than the first dataset, therefore the REP Tree final modal will

not be pruned.

2 folds			3 folds			5 folds			8 folds		
Trained on 06Data											
Pruned	Tested on 16Data	Size of Tree	Pruned	Tested on 16Data	Size of Tree	Pruned	Tested on 16Data	Size of Tree	Pruned	Tested on 16Data	Size of Tree
Yes	73.3333	21	Yes	84.881	21	Yes	73.3333	21	Yes	73.0952	17
No	72.381	25	No	72.381	25	No	72.381	25	No	72.381	25
Trained on 16Data											
Pruned	Tested on 21Data	Size of Tree	Pruned	Tested on 21Data	Size of Tree	Pruned	Tested on 21Data	Size of Tree	Pruned	Tested on 21Data	Size of Tree
Yes	87.3214	19	Yes	85.8929	19	Yes	90.5952	19	Yes	87.1429	19
No	90.7738	25	No	90.7738	25	No	90.7738	25	No	90.7738	25
Trained on 21Data											
Pruned	Tested on 06Data	Size of Tree	Pruned	Tested on 06Data	Size of Tree	Pruned	Tested on 06Data	Size of Tree	Pruned	Tested on 06Data	Size of Tree
Yes	82.0238	21	Yes	74.5086	21	Yes	77.0833	21	Yes	82.0238	19
No	96.9029	29	No	96.9029	29	No	96.9029	29	No	96.9029	29

Figure 5.12: REP Tree Pruning Data

Feature selection shows that a few attributes made no difference once being removed. For example the 'Smallest Gap' and 'Average Gap'. However based on the fact that more devices could be added and for new data, these attributes have the potential to be useful, so all the attributes remain for the REP Tree. Figure 5.13

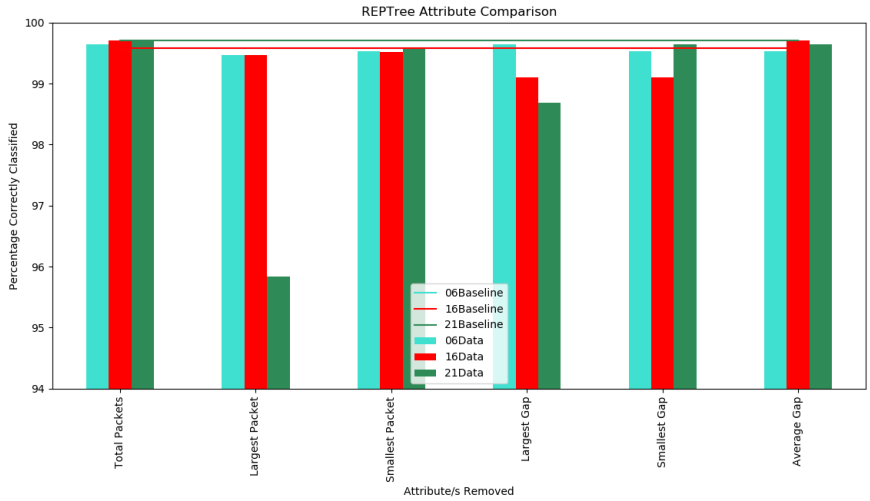


Figure 5.13: REP Tree Feature Selection

Logistic Function

The 'MaxIts' field (maximum iterations) was tested on values between 1 and 200 plus an infinite amount. The accuracy values were inconsistent with 200 iterations not matching the infinite value for the 06Data, exceeding the infinite value for the 16Data and matching the infinite value for the 21Data.
For all three datasets, a maximum of 100 iterations got extremely close to the infinite with (0.2372, 0.2381, 0.0595) percentage differences in the values (Figure 5.14).
The general theme is that the more iterations that can be done the more accurate the results will be, however there is a decreasing rate of increase. Due to the difference in accuracy being so small

with the larger number of iterations but taking so much longer to produce. 100 iterations will be used.

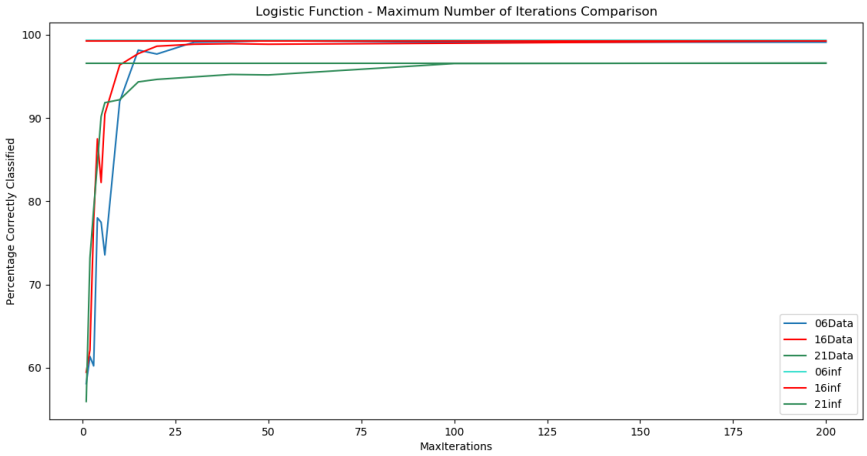


Figure 5.14: Logistic Function maximum iterations

Conjugate Gradient Descent has the option to be 'true' or 'false' to determine if it is used or not for the Logistic Function. It is generally faster for problems with many parameters, however for my data it made minimal difference to the accuracy and the root mean squared error (RMSE) (Figure 5.15). The RMSE is the difference between values predicted by a model and the values actually observed [30]. Presumably, the data does not possess enough parameters to make a significant difference so the field will be left as false.

	06Data		16Data		21Data	
ConGradDes	% Correct	RMSE	% Correct	RMSE	% Correct	RMSE
FALSE	99.348	0.0423	99.2262	0.0456	96.6071	0.0861
TRUE	99.2887	0.044	99.2262	0.0458	96.4881	0.0861

Figure 5.15: Logistic Function conjugate gradient descent

Feature selection for the Logistic Function revealed that removing the 'Average Gap' for the 06Data and 16Data made a positive impact on the accuracy. However, due to the 21 dataset benefitting from this attribute, all of the features will remain (Figure 5.16).

A summary of the determined hyperparameters can be seen in Figure 5.17 .

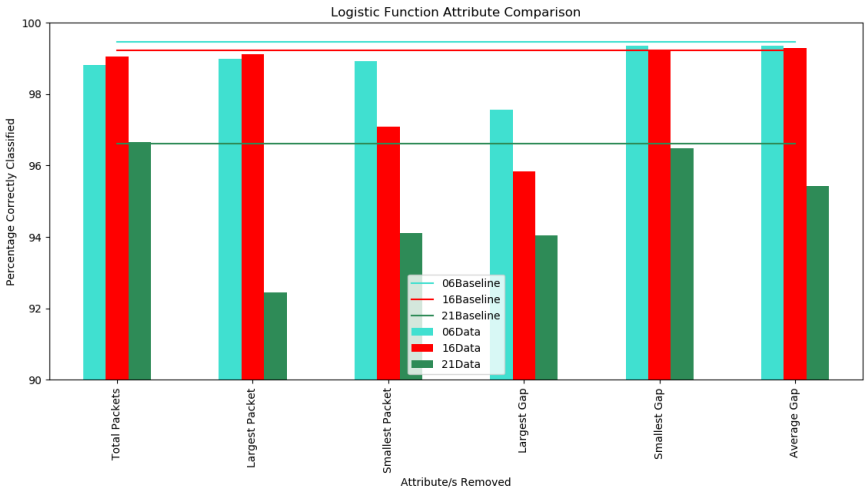


Figure 5.16: Logistic Function Feature Selection

Random Forest	
MaxDepth	unlimited
Iterations	100
Decimal Places	2
Attributes	Remove 'Total Packets'
REP Tree	
MaxDepth	6
Pruning	Not pruning
Attributes	All
J48	
Pruning	Pruning
Folds	Default
MinNumObj	2
Attributes	Remove 'Total Packets'
Logistic Function	
Iterations	100
ConGradDes	Not using
Attributes	All

Figure 5.17: Summary for all four algorithms

5.2.3 Algorithm Evaluation 3. Final Models

Building the final models for the four algorithms involved combining all three of the datasets (6th, 16th and 21st) to produce the training data. The more training data used, the more accurate the model is likely to be on unseen data. An equal amount 721 entries for each file produced a total of 5047 instances to be trained on. It is important that the data is balanced so that no device has more weighting when predictions are made.
The hyperparameters decided in 'Evaluation 2.' will be applied to the models.

To test the models, unseen data from a different day (26th) is used. The new file is filtered by IP address and each algorithm is supplied with seven files where the class value is unknown. Each device has approximately 240 records meaning the total amount of records to be classified are about 1680. The actual device can be confirmed after the classification by comparing the IP address to Table 13.1 in the appendix.

The final models for the J48 and REP Tree can be seen in Figure 13.36 and 13.37 in the appendix as visualisations in WEKA. Logistic Function and Random Forest do not have models that are able to be visualised.

Confusion Matrix Random Forest 26Data							
a	b	c	d	e	f	g	<-- classified as
240				1			a = A.Echo
1	240						b = B.Netcam
		241					c = Camera
			241				d = Hive
				241			e = Lamp
					240		f = S.Plug
						241	g = S.Things

Figure 5.18: Confusion matrix Random Forest 26Data

Confusion Matrix REP Tree 26Data							
a	b	c	d	e	f	g	<-- classified as
240				1			a = A.Echo
1	240						b = B.Netcam
		241					c = Camera
			241				d = Hive
				241			e = Lamp
					240		f = S.Plug
						241	g = S.Things

Figure 5.19: Confusion matrix REP Tree 26Data

Confusion Matrix J48 26Data							
a	b	c	d	e	f	g	<-- classified as
240				1			a = A.Echo
	241						b = B.Netcam
		241					c = Camera
			241				d = Hive
				241			e = Lamp
					240		f = S.Plug
						241	g = S.Things

Figure 5.20: Confusion matrix J48 26Data

Confusion Matrix Logistic Function 26Data							
a	b	c	d	e	f	g	<-- classified as
200			40	1			a = A.Echo
	226		15				b = B.Netcam
	1	240					c = Camera
3	1		237				d = Hive
				241			e = Lamp
					240		f = S.Plug
					1	240	g = S.Things

Figure 5.21: Confusion matrix Logistic Function 26Data

A confusion matrix for each of the algorithms summarises information about the actual and predicted data. It shows for a particular device say a 'Lamp', where Lamp instances that are correctly classified as a Lamp (True Positive) all non Lamp instances that are correctly classified as not a Lamp (True Negative). Non Lamp instances that are incorrectly classified as a Lamp (False Positive) and Lamp instances that are incorrectly labelled as not being a Lamp (False Negative).
The confusion matrices from the 26Data (Figures 5.18, 5.19, 5.20, 5.21) show that the accuracy of the models are:

- Random Forest - 99.88%
- REP Tree - 99.88%
- J48 - 99.94%
- Logistic Function - 96.38%

Interestingly, all of the algorithms misclassified the same Amazon Echo instance as a Lamp implying that it was probably an outlier in the data. From the results it is obvious that the Logistic Function is performing significantly worse than the other three algorithms so it can be eliminated from the selection.

The remaining 3 algorithms performed accurately on the unseen data and there was no significant difference between them. A new unseen data set was then used (11th) along with the following performance measures from the model build and training set as these measures are used by Lakshmi Devasena C in [6]:

Confusion Matrix Random Forest 11Data							
a	b	c	d	e	f	g	<-- classified as
241							a = A.Echo
	241						b = B.Netcam
		241					c = Camera
			241				d = Hive
				241			e = Lamp
					240		f = S.Plug
						241	g = S.Things

Figure 5.22: Confusion matrix Random Forest 11Data

Confusion Matrix REP Tree 11Data							
a	b	c	d	e	f	g	<-- classified as
241							a = A.Echo
1	240						b = B.Netcam
		241					c = Camera
1			240				d = Hive
				241			e = Lamp
					240		f = S.Plug
	1					240	g = S.Things

Figure 5.23: Confusion matrix REP Tree 11Data

- Accuracy
- Confusion Matrix
- RMSE (Root Mean Squared Error)
- Mean Absolute Error

For the accuracy, the incorrect number of instances were '0' for Random Forest, '4' for REP Tree and '8' for J48 where the most problems came from distinguishing between the Hive and Amazon Echo (Figures 5.22, 5.23, 5.24).
The RMSE for Random Forest was '0.0072', for REP Tree '0.014' and for J48 '0.0212'.
The Mean Absolute Error is the average of the variance between predicted and actual values. It is a good measure to gauge the performance and shows the Random Forest and REP Tree have the same value of '0.0004' and J48 has a mean absolute error of '0.0009'.
A new data set was used considering how similar the performances were on the unseen 26Data in case the previous one was too close to the training data. The data was run yielding the following accuracy results:

- Random Forest - 100%
- REP Tree - 99.81%
- J48 - 99.69%

Confusion Matrix J48 11Data							
a	b	c	d	e	f	g	<-- classified as
241							a = A.Echo
	238				3		b = B.Netcam
		241					c = Camera
2			239				d = Hive
				241			e = Lamp
					240		f = S.Plug
						241	g = S.Things

Figure 5.24: Confusion matrix J48 11Data

6 Results

This section outlines the final results for the project including the attributes that make up the device type fingerprint, the final model used for classifying the data and how using these two results can produce device predictions on an unseen dataset.

6.1 Device Fingerprints

For this project, a device type fingerprint is made up of information found in the packet headers. A pcap file is filtered by an IP address of interest in Wireshark and exported as a CSV file.
The relevant information is then extracted by the Python script 'packetAnalysis.py' which includes the features:

- 'Largest Packet Size' which is taken as the largest sized packet (in bytes) to be sent or received by the target device during a 6 minute period.
- 'Smallest Packet Size' measured in bytes. Which is the smallest packet sent or received within a 6 minute period.
- 'Largest Gap' measured in seconds, which is the largest gap between packets being sent and received during a 6 minute period.
- 'Smallest Gap' similarly, is the smallest gap between packets being sent and received during the 6 minute periods. (measured in seconds)
- 'Average Gap' which is the mean of the gaps between packets during the 6 minute periods, measured in seconds.

An example of the data outputted by the python script is shown in Figure 6.1.
The amount of records produced depends on the capture time (the length of the pcap file) because each record represents an analysis of the consecutive 6 minute periods.
Figure 6.2. shows a visual representation and comparison of what the device type fingerprints look

L Pkt	S Pkt	L Gap	S Gap	A Gap	Class
590	60	15.0037	0.0002	4.0929	?
245	60	15.0648	0.0039	4.7963	?
1514	60	15.0229	0.0001	2.4009	?
1514	54	15.004	0.0001	2.4666	?
1514	60	15.0025	0.0001	2.4845	?
245	60	15.0109	0.0025	4.2635	?
245	60	15.0122	0.0001	4.5438	?
245	60	15.0084	0.0025	4.6044	?
245	60	15.0113	0.0024	4.485	?
1514	60	15.0121	0.0001	2.4491	?
590	60	15.0034	0.0031	4.0576	?
245	60	15.006	0.0019	4.4041	?
1514	60	15.0118	0.0001	2.3706	?
245	60	15.0145	0.0001	4.5624	?
245	60	14.9997	0.0025	4.4089	?
245	60	15.0265	0.0001	4.5069	?
245	60	15.0126	0.0019	4.6985	?
245	60	15.0384	0.0022	4.4682	?
1514	60	15.0032	0.0001	2.5036	?
245	60	15.0196	0.0024	4.5649	?
590	60	15.0088	0.0036	3.9472	?
245	60	15.0138	0.0024	4.6045	?
245	60	15.0201	0.0024	4.6044	?

Figure 6.1: Example of the analysed data output

like for each of the IoT devices. The results are normalised using min-max normalisation [18]. The data used is from the 19th and the first five records of each device are shown.

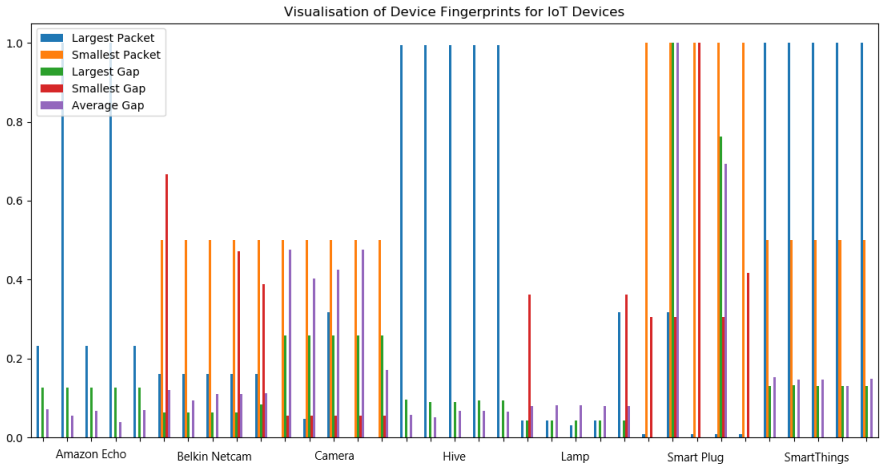


Figure 6.2: Visual of device type fingerprints (normalised)

6.2 Device Predictions

For this section the predictive ability of the Random Forest classifier is demonstrated. Two tests are performed using pcap files from the 14th and 19th, containing network traffic from the seven devices. They are opened in Wireshark and filtered by each of the IP addresses of interest which are then exported as CSV files.
The CSV files are then passed to the Python script 'packetAnalysis.py' which produces the analysis in the form of a new CSV file with the 'Class' values set to '?'. The tests are done on unknown and unlabelled data to replicate a real life scenario and to not influence the results of the classifier. The entire collection of results for each device can be found in 'Analysis14' and 'Analysis19'. The data is grouped into one file for convenience but was computed separately.

In WEKA, the Random Forest model is loaded into the 'Explorer' and the analysed CSV file loaded as a 'Supplied test set'. From "More options..." the "Output predictions" field is set to 'PlainText'. Possible outcomes for each device include:

- Case 1 where the device is correctly predicted its actual class.
- Case 2 where the device is incorrectly predicted as a different class.

The results are validated by comparing the IP addresses to those in Table 13.1 .
A screenshot of example output from WEKA can be found in Figure 13.37 from the appendix.

14Data

The accuracy of the model over the whole dataset from the 14th was 99.7%. Each individual prediction can be found in a separate file 'PredictionsOn14Data'. The results from all the data files are summarised in the following confusion matrix (Figure 6.3).

Confusion Matrix Random Forest 14Data							
a	b	c	d	e	f	g	<-- classified as
241							a = A.Echo
	239				2		b = B.Netcam
		241					c = Camera
			240	1			d = Hive
				241			e = Lamp
					240		f = S.Plug
	2					239	g = S.Things

Figure 6.3: Confusion matrix Random Forest 14Data

Confusion Matrix Random Forest 19Data							
a	b	c	d	e	f	g	<-- classified as
241							a = A.Echo
	241						b = B.Netcam
		241					c = Camera
1			240				d = Hive
				241			e = Lamp
					240		f = S.Plug
						241	g = S.Things

Figure 6.4: Confusion matrix Random Forest 19Data

19Data

The accuracy of the model over this dataset from the 19th was 99.9%. Each individual prediction can be found in a separate file 'PredictionsOn19Data'. The results for all the data are summarised in the following confusion matrix (Figure 6.4).

The final prediction for a file is the majority predicted from all the instances.
Given the results, theoretically, even with only one 6 minute period the chance of a correct classification is extremely high. Therefore the capture time of the traffic is greatly reduced which makes the whole process more viable as an option to determine devices on a network.

7 Implications

This section discusses the implications of the results discovered. Having the ability to discover the type of device on a network can affect businesses and the general public in both negative and positive ways.

7.1 Business Perspective

For some businesses, determining the devices on a network could be of critical importance for ensuring the security of the company's network by confirming access control. If all the types of devices are expected eg printer, laptop, lamp then there is no problem, or if an unknown device is connected, they will know what type of device they are looking for.

The medical industry in particular could be a beneficiary of the ability to predict devices that are present. Devices such as pacemakers, insulin pumps and smart blood pressure monitors could be useful to be able to detect on a person. For example, given the situation where paramedics arrive and a person is unconscious, being able to detect what devices they have on them would be useful for administering treatment and providing care to the patient.

For the companies producing these IoT devices, the growing interest in new exciting devices by consumers can mean the pressure to be innovative and the first one to release new devices sometimes causes the security of their products is compromised. This could mean an attacker with knowledge of the type of device could better plan a device specific attack.

7.2 General Public Perspective

As these ubiquitous devices are becoming more popular, the general public will become more concerned with the potential malicious use of a device type fingerprint in their own home.

In the vicinity of someone's home, an attacker can collect traffic, extract the device type fingerprint and predict what devices are present, violating the privacy of the user.
Possessing this information could lead to other additional attacks. For example, a physical attack where an attacker can determine whether or not someone is worth robbing before breaking in. In addition to this, IoT devices are being used to keep homes secure with motion sensors and locking systems. For an attacker, knowing these devices are present will allow them to avoid detection.
Alternatively, an attacker could know if a device such as a baby monitor is present so that other cyber attacks can be performed.
An attacker can particularly take advantage of situations such as hotels or coffee shops where many devices are present as they can be read unobtrusively because of the passive nature of the process. Therefore the user cannot know if their privacy has been invaded.

An even more concerning fact is that the solution discovered uses free software that is available on multiple platforms. The availability coupled with the simplicity of the process means that an attacker does not need a wealth of knowledge about the technology to achieve the same results.

7.3 Avoiding Attacks

As Nitesh Dhanjani [7] points out, there are some ways that a user can safeguard against an attack on their IoT device.
Not using default credentials is a simple but key point as stated on [p. 64]. Users were not prompted to change the login details from the default of 'admin' and a blank password. This led to a huge number of people vulnerable to having their privacy violated.
Similar to this, using a secure password helps to greatly decrease the chance of an attacker using brute force to guess the login details to a device and not relying on single-factor authentication for security systems or accounts with sensitive information.

8 Evaluation

This section evaluates how well the project addresses the problem outlined in the introduction. It shows to what extent the goals were achieved and discusses the strengths and weaknesses of the project and results.

During the project, producing a device type fingerprint from the packet headers was proven to be achievable. No specific number of attributes were aimed for, however the analysis ended up with five attributes ('Largest Packet Size', 'Smallest Packet Size', 'Largest Gap', 'Smallest Gap' and 'Average Gap'). Although in an ideal world, one attribute would be enough, for this project, it would not have been ideal as the separation between devices is not so clear cut.
In addition to this, classification was possible using the device type fingerprints and a Random Forest model to produce predictions on unseen data.

In addition to achieving the main goals, the programming language used (Python) and the software used (Wireshark and WEKA) were free.
The fact that the process can be done on many different environments means the solution is available to more users and has less limitations.

8.1 Tools and Techniques Evaluation

Using Python for the analysis gave flexibility to the process which helped to solve problems effectively. This was particularly useful at the beginning of the project as the analysis needed to be changed and adjusted when trying new attributes for the device type fingerprint. Testing a variety of data on the Python script also revealed some initially unseen situations such as:

- Only two packets being sent in the time frame and at the same time. Which provided a problem with finding the gaps between the packets being sent.
- Some anomalies where the time for the previous packet is greater which resulted in some negative time gaps.
- Only one packet being sent or received during the time frame which disrupted the average calculations.
- And no packets in the time frame where the results were not added to the analysed output as the values would otherwise end up as inf or -inf.

Overall, using Python was a good idea as it made it easy to change and adjust the analysis as the project progressed.

The main benefit of using WEKA was the opportunity it provided for multiple algorithms to be compared. Without WEKA, each of the models would need to be built and tested manually for a comparison which would be too time consuming.

Another beneficial aspect of the process was that Wireshark, Python and WEKA can all use CSV files which helps with the flow of data so it does not need to be transformed between different file formats.
Overall, the methodology chosen for the project was appropriate.

8.2 Strengths

The main strengths of the project are the simplicity of the process, the accuracy of the predicted results and that it met all the goals that were set at the start.
Due to the success of the results with an accuracy in the high ninety percents, if a prediction needed to be made with a small section of only 6 minutes, the probability of the user getting the correct prediction is still very high. However, given that different states of devices are not included and to increase the reliability of the predictions it would be recommended that a larger collection period is used. For example, if a device is active for part of the time and then inactive for the other part, then the predictions will still accurately highlight the section where the device is inactive.

Given that the Random Forest classifier performs so well, if time restraints become more critical or lots of data needs to be analysed at once then it may be possible to remove some of the attributes. Although this would decrease the accuracy of the model, removing one or two attributes would only have a small negative effect.

8.3 Weaknesses

There were times when the project went in the wrong direction, mainly in the first part of the project when the aim was to try and put together a device type fingerprint. Some of the options tried were dead ends and even for the attributes that were used in the end, a lot of alterations were made on the way for example the intervals and decimal places to use.

A weakness of the project is the liquidity of the solution. Only one device can be extracted and analysed from the pcap file at a time. Furthermore, switching between CSV files means the solution is not fully automated and it is not done in real time. As stated in the work by A. Selcuk Uluagac et al. *"A fingerprinting technique needs to be quick in order to be of any practical use"* [10].

As discussed further in the 'Future Work' section the testing done could be extended which would allow me to be more confident in the results.
Firstly, the tests were only done on data from a controlled environment. Using a real life scenario might demonstrate how realistic the process is.
Furthermore, using more devices of the same type would show whether the classification is for device types or for a specific device.
Lastly, adding in more devices of different types would show how effective the attributes are when dealing with new devices.

The scalability of the project is limited by the amount of available IoT devices and the cost of having more than one of each type available. For the classification of devices, if the model is to be improved to make predictions about more devices, access is needed to at least one of those new devices to train the model again.

9 Future Work

This section outlines the potential future improvements for the project and what further developments could be made if more time was allowed for the project or if financial limitations were removed.

Device Fingerprinting

From reading [10] it is clear that device fingerprinting and device type fingerprinting are closely related. Providing additional functionality of identifying a specific device as well as its type would add to the usefulness of the process.

Potential Attributes

Another factor that was realised during the project is that given the ability of some attackers to change the data rate, modify packet sizes and inject random delays the reliability of the results produced may be affected which would affect the trust of the process therefore other attributes may need to be looked into.
During the project, potential options such as just looking at the sent packets, received packets or the ratio between sent and received as in [1]. Similarly, during the experimentation of 30 seconds intervals it revealed a lot of gaps where no data was sent for certain devices such as the Smart Plug (Figure 9.1). Therefore, looking at the burstiness of the traffic might reveal some patterns for devices.

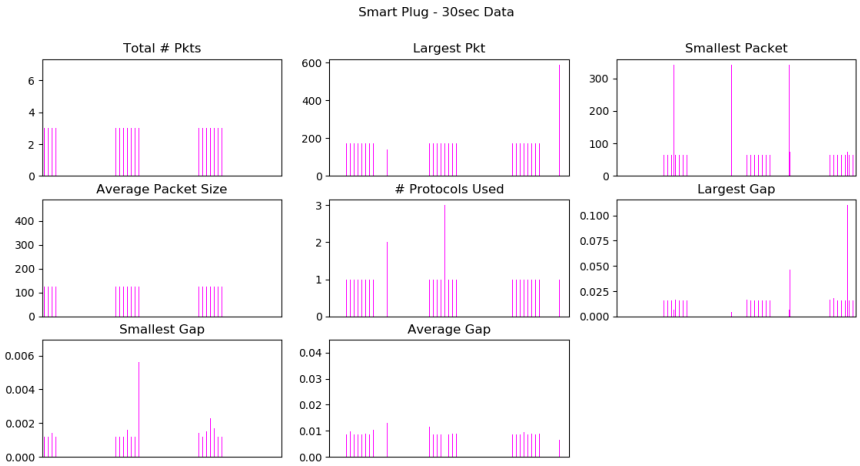


Figure 9.1: Smart Plug data 30 second intervals

Unknown Devices

It would be nice to extend the project to include previously unseen devices as they do with GTID [10]. Practically, it would be more beneficial for the functionality to include this, although a starting point may be to just have a separate class to label data that does not fit any of the available classes.

Variety of Testing

One form of testing that could not be done was seeing if the classification for each device could be carried over to different devices of the same type.
Due to the fact that only one device of each type was available, it is not certain that the behaviour demonstrated and used for the classification would be replicated in all other devices of that type. Therefore for future testing, multiple devices of the same type should be used along with some devices of the same type and different brand to confirm that the results represent the device type and not the specific device that it was trained on.

Evaluating more Algorithms

Due to the fact that only eight algorithms were tested, it would be a good idea, given more time, to evaluate the performance of a larger variety of algorithms. Given the highest performing algorithms were decision trees it would be particularly beneficial to look at the success of other trees in predicting the device type.

Speed of the Process

For applications such as medical purposes or any urgent situations, automating or speeding up the process would be key factor.
Currently, the process is clunky, the user has to move from Wireshark to the Python script to WEKA in different CSV files. This is by no means difficult and takes a minimal amount of time, however it prevents the device type predictions from being a fully automated process.

Active vs Inactive

The project also uncovered the issue of active and inactive devices producing varying traffic. The difference would be enough that they would either need their own categories, for example 'Active Lamp' and 'Inactive Lamp', or the model will have to be trained on equal amounts of active and inactive records. However some devices are more complex like the Amazon Echo does not only have one active state and one inactive state. For example, they may have different traffic depending on whether they are playing music, on a phone call or using the voice assistant. In [1] observations showed a large difference between the total number of packets being received for the same Amazon Echo device which could have been caused by the device being active or inactive.
For the current process, the data is evaluated in blocks so if the state changes midway through at the point where it changes to inactive, the predictions should be accurate.

Controlled Experiments

The training and test data were taken from replica conditions in the IoT test lab. Although the testing is done using unseen and unlabelled data, it was not an exact replica of a real life scenario. Therefore an idea for future testing is to move outside the IoT lab to see if the results achieved match those from the lab.
Furthermore, the data used was not entirely standard as some attacks were being performed on the devices at various unknown points. [4]. If more time was available, it would be beneficial to collect traffic in a controlled situation to ensure that the results would be the same.

Testing on a Larger Scale

The attributes used were sufficient for the devices that were available. Increasing the scale of the project and adding more devices may result in the current attributes overlapping too much for

classification to be viable. Therefore, testing and adding more devices will determine whether the attributes need to change or new ones need to be added.

10 Conclusion

The two main goals of this project were to a) Discover whether device type fingerprints for certain IoT devices can be found and b) Whether the type of IoT device can be determined just through reading the device type fingerprint using a classifier. The project was a success in discovering attributes that constitute a device type fingerprint and using a Random Forest classifier that can predict the device type for unknown data with a high accuracy. A user should be able to select an IP address from some network traffic they have, analyse that device and get predictions on what type of device it is.

I am happy that the classifier performs as accurately as it does. The core functionality is there but improvements are still to be considered in terms of scalability and complete testing. As mentioned in future work, the solution is perhaps not optimal in terms of efficiency. There are additional steps in the project where the user exports CSV files and adds them in to other parts of the process. Additionally, complete and thorough testing is yet to be completed with more devices and data.

Overall, the project has been a success. The main aims of the project have been met and it provides an insight into an emerging area of device security. The project lends itself to potential uses in businesses and industries such as medicine, and sheds light on potential security concerns for everyday users. As people become more interested in having IoT devices in their homes, the pressure for companies producing these devices puts more focus on being innovative and on trend rather than being concerned about the security of their products. Once a balance between innovation and security is met, succinctly put in [1] *"It might then be possible to gradually realise the old dream (or perhaps nightmare?) of the 'smart home'"*.

11 Reflection

In this section I discuss some of the challenges I faced during the project, changes from the initial plan and reflect on the project as a whole.

I decided to do a project in this area as its emerging and relevant to today's society. It combines security and data analysis to answer questions about things "We know we don't know" from the phrases popularised by Donald Rumsfeld [26]. In the case of my project, the question was about identifying IoT devices through a device type fingerprint taken from the network traffic.

The main difference between the initial plan and the actual project was the shift away from the collection of the data so the project was focussing on the analysis and classification. The way the data is captured turned out to be less important for the project than producing a practical device type fingerprint and accurate predictions. Both of these parts taking longer than expected also contributed to the change in plan.

Another difference to the initial plan was the timings for the tasks. This was mainly due to my predictions being incorrect and overestimating the time available.
For some of the tasks such as the introduction, in my plan the task was to be completed in week three however in reality, finishing this task was not possible till after I knew how the project planned out so not being able to complete tasks was demotivating.
For other tasks such as the analysis of the network traffic, I allocated two weeks to find the attributes that would constitute the device type fingerprint, however this task ended up being the majority of the project and took many weeks to perfect. However I think the underestimation was mostly due to the fact that I did not know what I was going to find, how taxing the analysis would be or if it was even possible before I started it.
This was definitely the main challenge that I faced as it was difficult to find the attributes from the data that would be distinct enough as it turned out that the traffic for the same device varies substantially.

In addition to underestimating tasks, the time taken to learn WEKA for the classification problem and Pandas/ Matplotlib for the data visualisation were not factored in, and therefore meant I had to work twice as hard for certain parts of the project to stay on track.

For the second part of the project I had trouble when trying to evaluate the performances of the algorithms on unseen data. The aim was to get statistics for each of the algorithms about their predicted values compared to the actual values, however when labelled data was passed to the models only a small amount of the total records had predictions made for them.
To overcome this problem I passed the models the unseen data without any labels and calculated the metrics such as accuracy myself, Luckily, only two or three were misclassified each time so the accuracy was quick to calculate.

I found that successes in the project led to complacency sometimes. For example, once I had found the attributes that I wanted to use for the device type fingerprint I realised that I relaxed a bit, so maintaining focus throughout the project is important for keeping to the schedule.
The success for the first part of the project also meant that I had finally got the Python script to a point where I was finished tweaking it. However, due to the amount of changes it meant that the script was messy and definitely not the most efficient way of getting to the answer. I found that

I did not want to alter it incase I broke it or because spending more time on it would not further the project. Eventually, after most of the project was complete, I decided to improve the packet analysis file.

Failure in the project forced me to look at my mistakes but led to innovation of new, better ideas. For example, at the start of the project I was using small amounts of data and found that no patterns were becoming visible. This part of the process was daunting as the basis of my project relied on me finding a feasible distinction in the data. Fortunately, using larger amounts of data revealed some more common values occuring for attributes, shown more clearly in the histograms.

Towards the end of the project I realised that one of the algorithms available on WEKA called 'IBK' was the 'K nearest neighbour' algorithm which would have been a good one to put in the evaluation, however I decided not to add it in as it would have meant redoing a large proportion of the project which I did not have time for.

I feel as though the project has enhanced my problem solving abilities and given me a great insight into data analysis and machine learning techniques.
New explicit knowledge that I have acquired such as using Pandas, Matplotlib and WEKA has become tacit and internalised through practise as my understanding of them has greatly increased.

For the initial stages of the project I found that having lots of thoughts and ideas led to information overload. The more I looked into the data, the more possibilities became apparent. There were many variables to test and control such as the interval time, attributes to try, collection times to use, modes and ranges of attributes etc, so sorting out a chronological order to complete tasks was a challenge in organising my ideas.

For some parts of the project, (particularly with discovering the device type fingerprints as there was a lot of trial and error) I learnt the importance of both single and double loop learning.

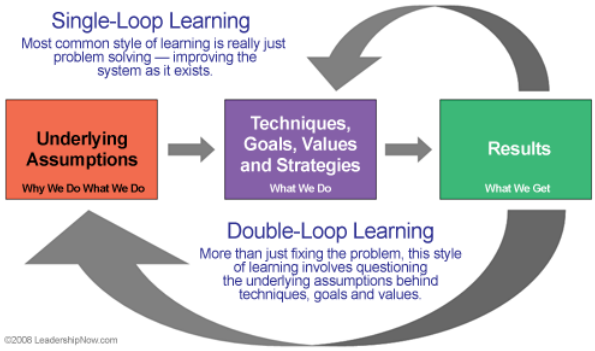


Figure 11.1: Single and double loop learning model [20]

From Figure 11.1, going down a particular route for the analysis would be the assumption and the desired result was finding something that could be used to determine a device type fingerprint. If the result was not met, single loop learning would involve altering the current method and double loop learning questions the assumption and therefore leads to new ideas being tried.
For example, in my project my assumption was that using the 'Protocol' column in the packet header would get me an attribute that I needed. Initially I tried the most common protocol, however this led to all the devices having TCP as the most commonly used protocol which was not useful. Single loop learning led me to change the way I was analysing this column and use the number of different protocols used instead. This looked more promising however later into the project it proved to be

less practical so double loop learning led me to look into other potential attributes.

Overall, I am happy with the results and the project in general. I did not expect the predictions to be so accurate so I tested a lot of data to ensure that it was not a coincidence that the data for testing and data for training were similar.
It has taught me the importance of time management and not to underestimate tasks. I have also learnt how to compare and evaluate the performances of machine learning models.
I managed to achieve everything that I set out to do in my goals and I found the project challenging and interesting to work on.

12 Table of Abbreviations

GUI	Graphical User Interface
CSV	Comma-separated values
pcap	Packet capture file
XML	eXtensible Markup Language
IoT	Internet of Things
OS	Operating System
ANN	Artificial Neural Network
IP	Internet Protocol
MAC	Medium Access Control
IQR	Interquartile Range
TCP	Transmission Control Protocol

13 Appendices

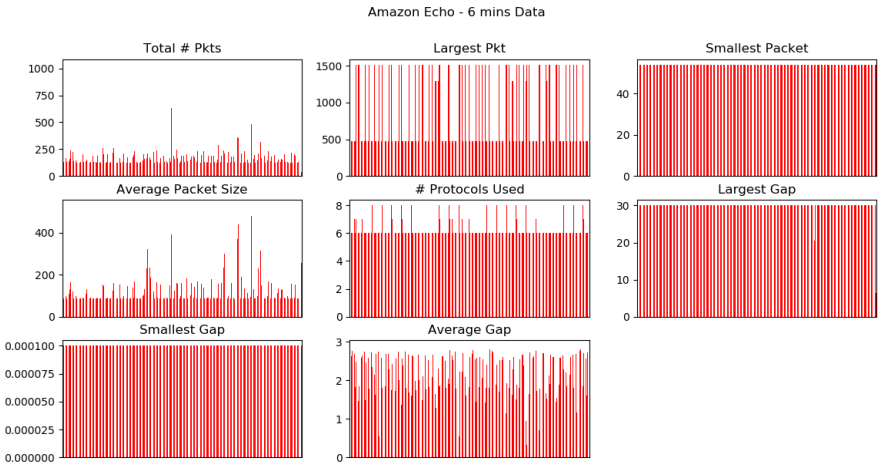


Figure 13.1: Amazon Echo 6 minute intervals

Amazon Echo Dot	192.168.200.109	Wireless
Belkin IP Camera	192.168.200.114	Wireless
TP-link NC200 Camera	192.168.200.127	Wired
Hive	192.168.200.121	Wired
Lifx Lamp	192.168.200.125	Wireless
TP-link Smart Plug	192.168.200.123	Wireless
Samsung SmartThings	192.168.200.122	Wired

Table 13.1: Devices and IPs [3]

Figure 13.2: Amazon Echo 15 minute intervals

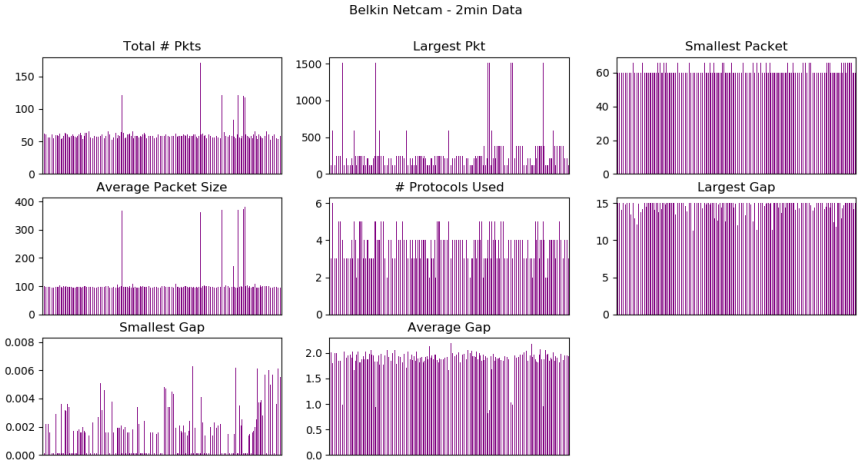


Figure 13.3: Belkin Netcam 2 minute intervals

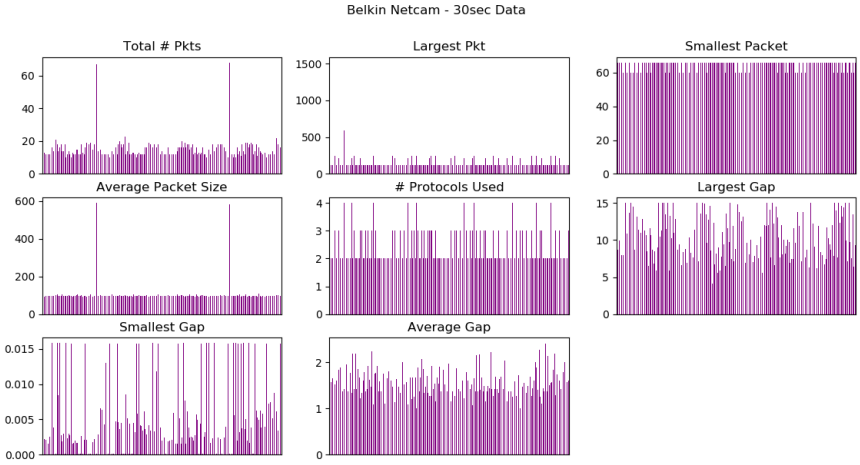


Figure 13.4: Belkin Netcam 30 second intervals

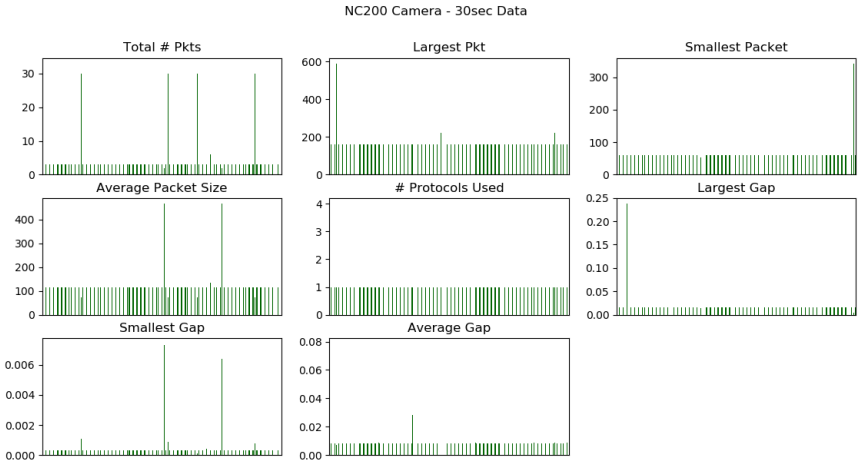


Figure 13.5: Camera 30 second intervals

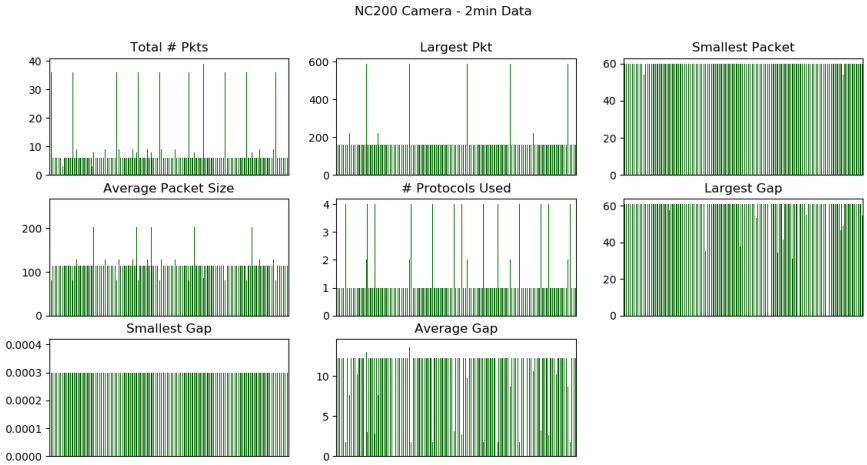


Figure 13.6: Camera 2 minute intervals

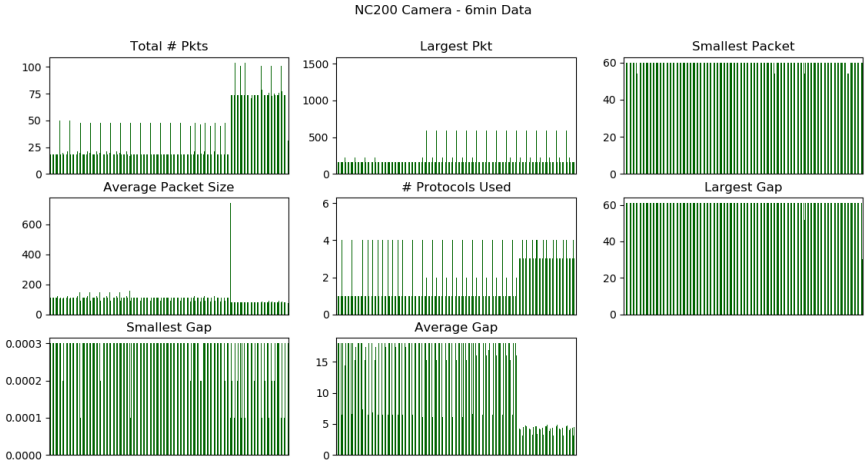


Figure 13.7: Camera 6 minute intervals

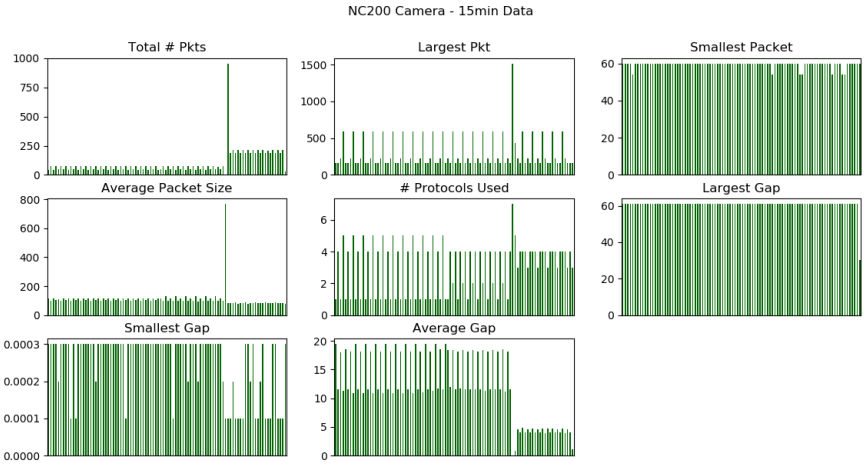


Figure 13.8: Camera 15 minute intervals

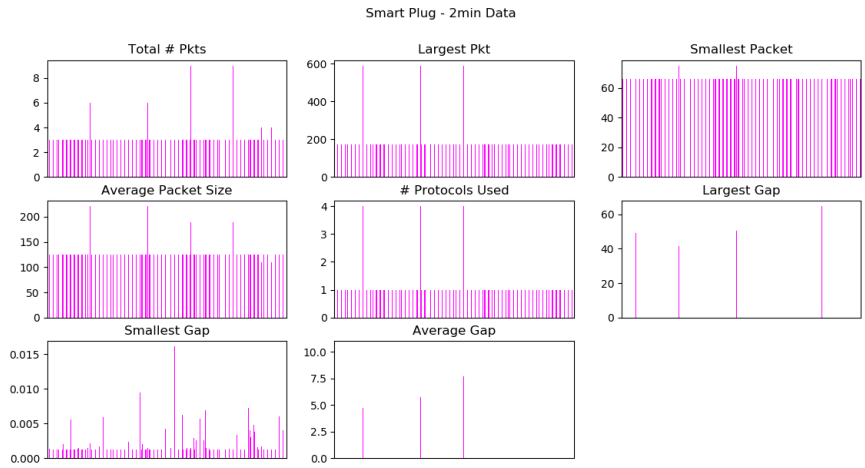


Figure 13.9: Smart Plug 2 minute intervals

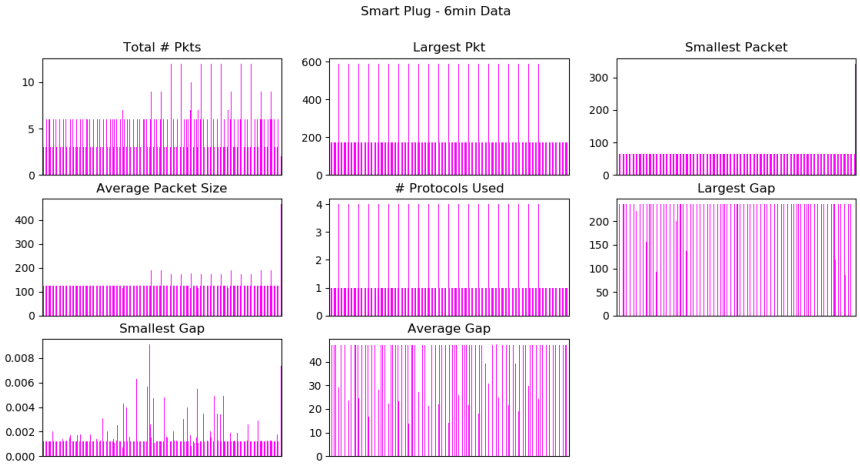


Figure 13.10: Smart Plug 6 minute intervals

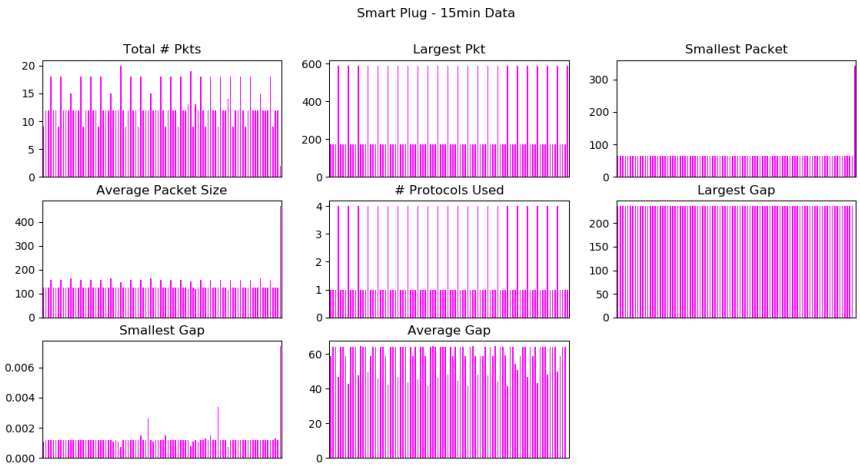


Figure 13.11: Smart Plug 15 minute intervals

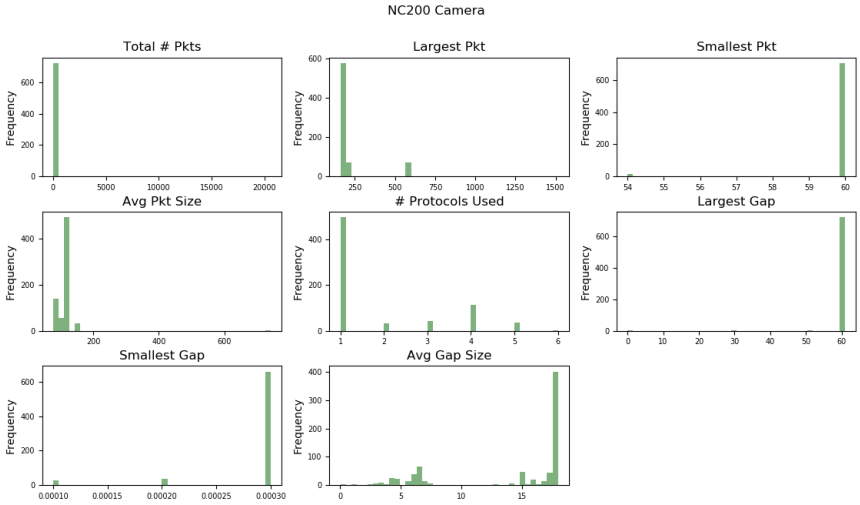


Figure 13.12: Camera distribution of values for attributes

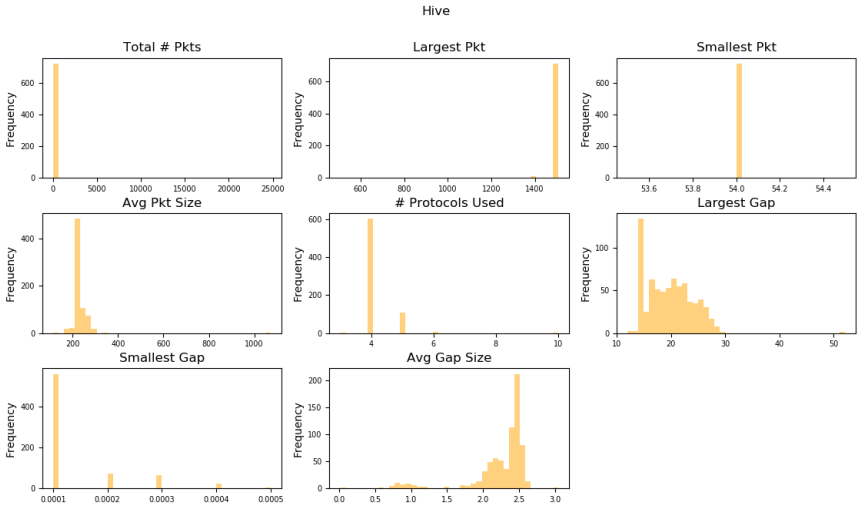


Figure 13.13: Hive distribution of values for attributes

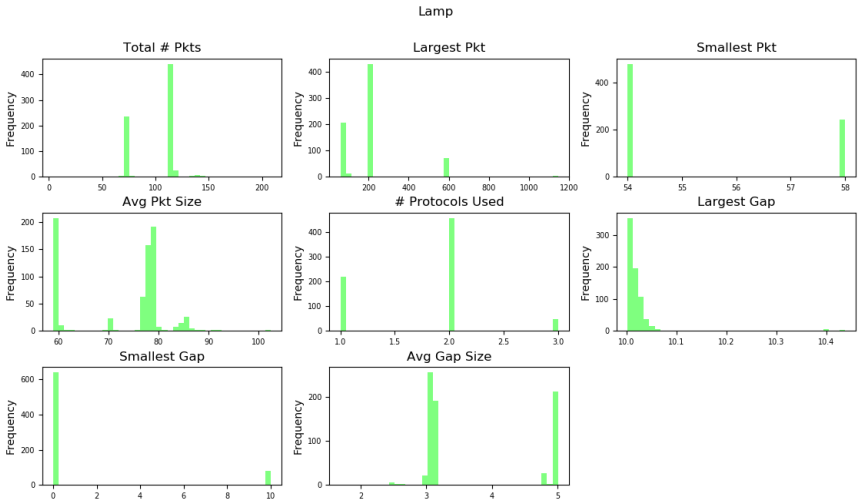


Figure 13.14: Lamp distribution of values for attributes

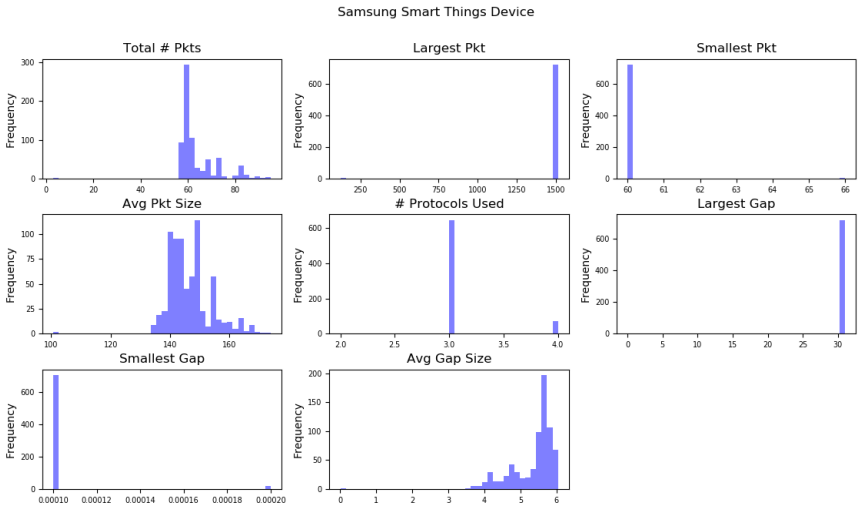


Figure 13.15: SmartThings distribution of values for attributes

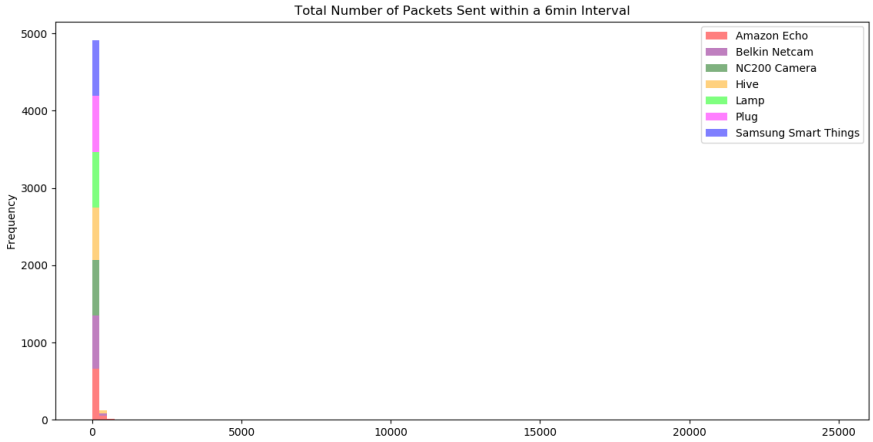


Figure 13.16: Distribution for Total Number of Packets

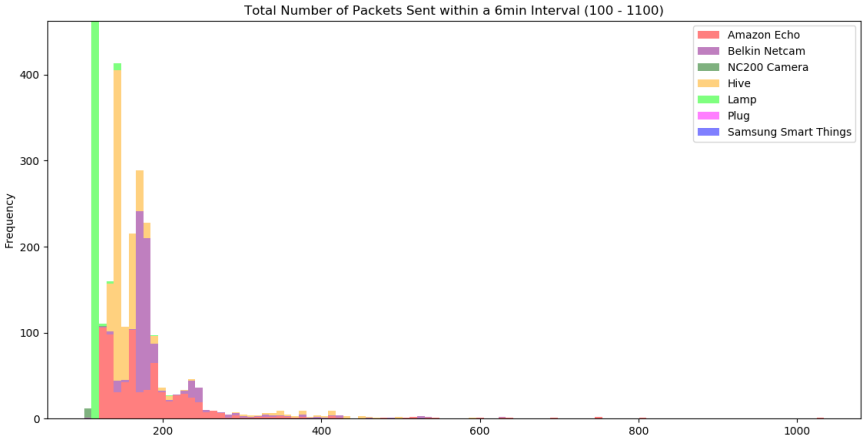


Figure 13.17: Distribution for Total Number of Packets (100 +)

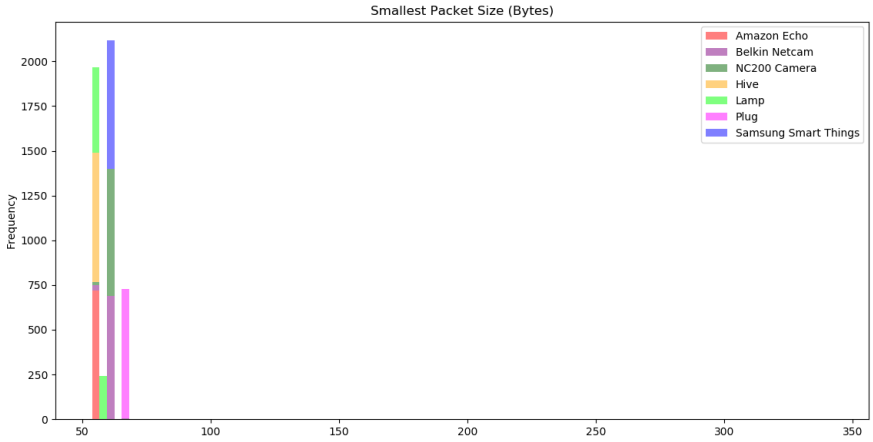


Figure 13.18: Distribution for Smallest Packet

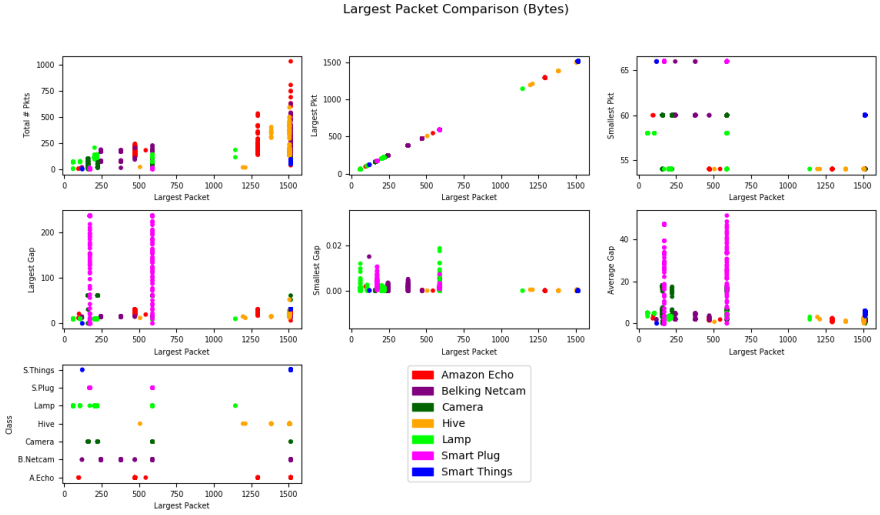


Figure 13.19: Attribute comparison to Largest Packet

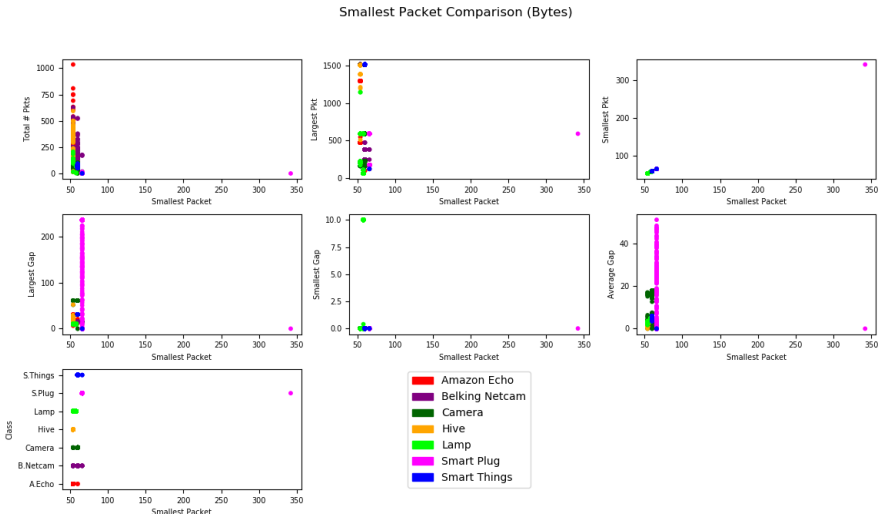


Figure 13.20: Attribute comparison to Smallest Packet

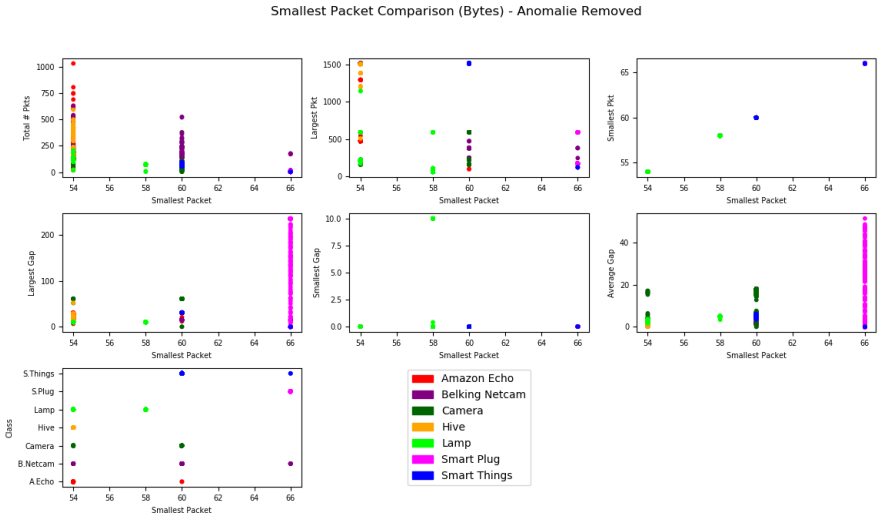


Figure 13.21: Attribute comparison to Smallest Packet (anomalies removed)

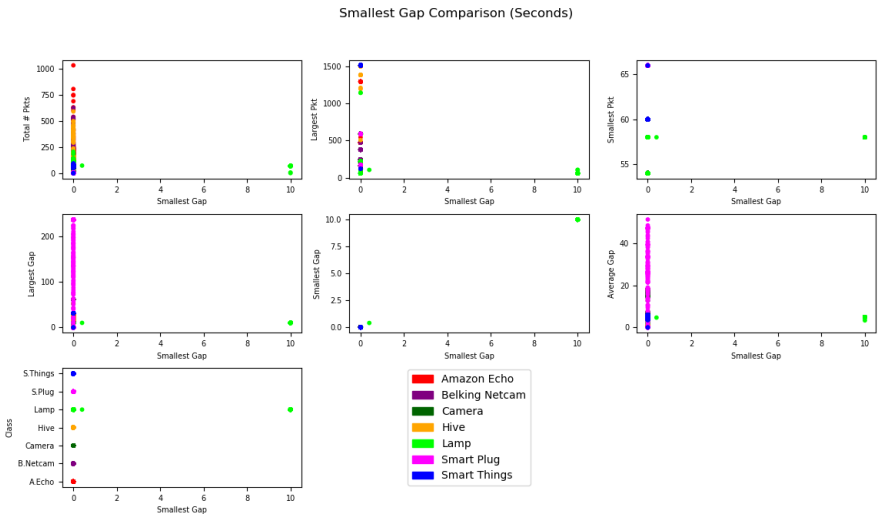


Figure 13.22: Attribute comparison to Smallest Gap

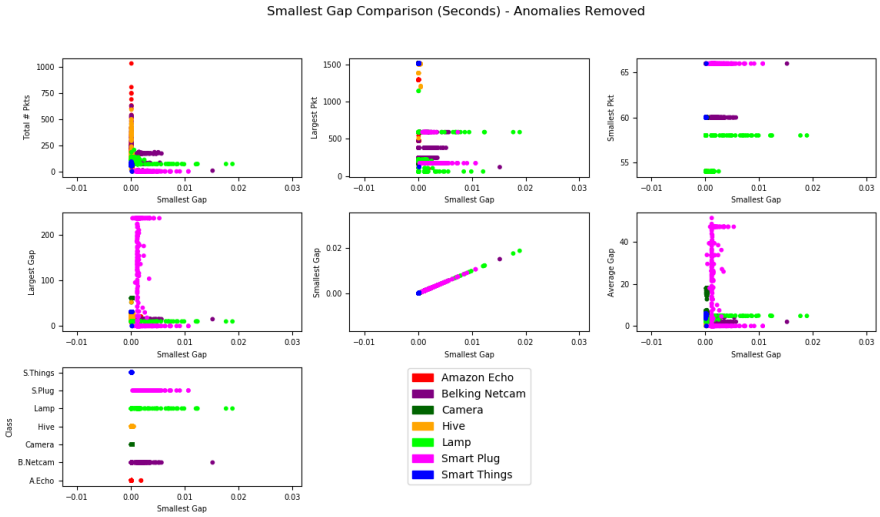


Figure 13.23: Attribute comparison to Smallest Gap (anomalies removed)

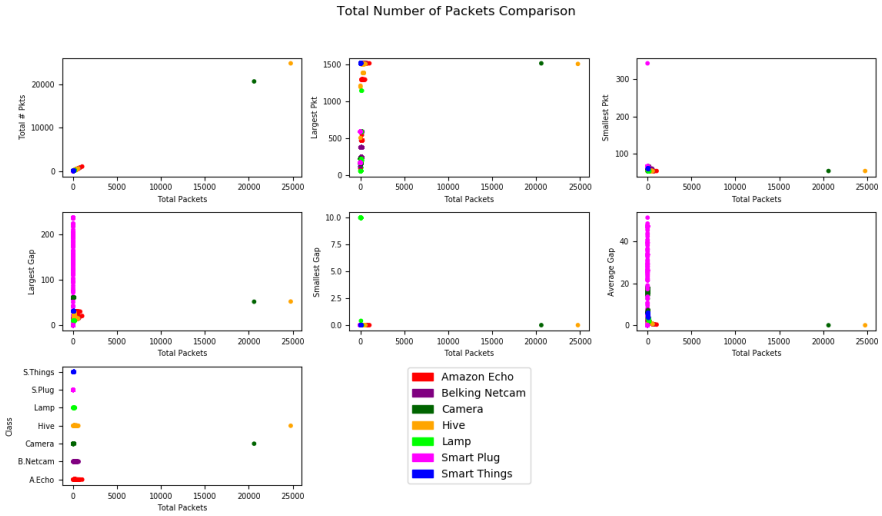


Figure 13.24: Attribute comparison to Total Packets

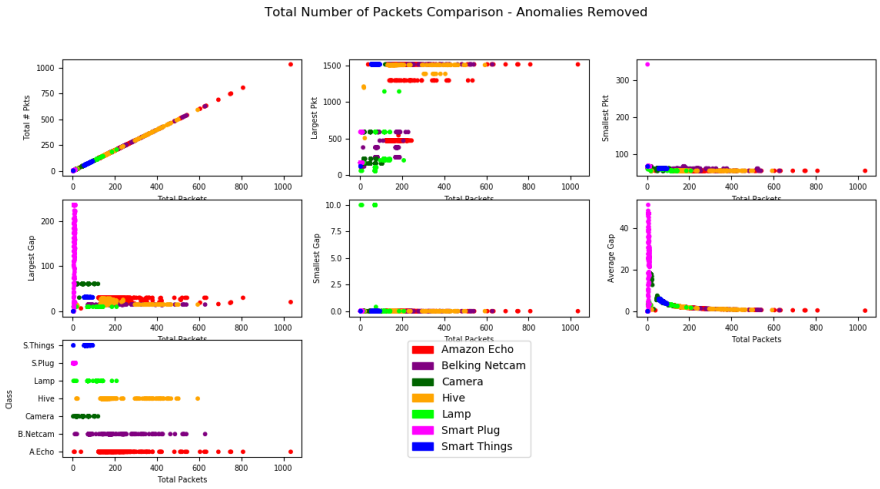


Figure 13.25: Attribute comparison to Total Packets (anomalies removed)

	% Correctly Classified		
Attribute	06Data	16Data	21Data
Baseline	99.4665	99.4643	99.5238
Total Packets	99.5258	99.5238	99.5238
Largest Packet	99.4072	99.2857	95.5952
Smallest Packet	99.4665	99.2262	99.2262
Largest Gap	99.6443	99.1071	98.5714
Smallest Gap	99.4665	99.4048	99.5238
Average Gap	99.4665	99.4643	99.5238

Figure 13.26: J48 feature selection results

Attribute	06Data	16Data	21Data
Baseline	99.4665	99.2262	96.6071
Total Packets	98.8145	99.0476	96.6667
Largest Packet	98.9923	99.1071	92.4405
Smallest Packet	98.933	97.0833	94.1071
Largest Gap	97.5697	95.8333	94.0476
Smallest Gap	99.348	99.2262	96.4881
Average Gap	99.348	99.2857	95.4167

Figure 13.27: Logistic Function feature selection results

	% Correctly Classified		
MaxIterations	06Data	16Data	21Data
1	58.0913	59.4643	55.9524
2	61.3515	62.1429	73.2143
3	60.2253	77.5595	79.0476
4	78.0083	87.5	84.5238
5	77.4748	82.2619	90.1786
6	73.5625	90.4762	91.8452
10	91.9976	96.369	92.2024
15	98.1624	97.7381	94.3452
20	97.6882	98.631	94.6429
30	99.1108	98.869	94.9405
40	99.1701	98.9286	95.2381
50	99.2887	98.869	95.1786
100	99.1108	98.9881	96.5476
200	99.1108	99.2857	96.6071
infinite	99.348	99.2262	96.6071

Figure 13.28: Logistic Function iterations results

Attribute	06Data	16Data	21Data
Baseline	99.5851	99.5833	99.7024
Total Packets	99.6443	99.7024	99.7024
Largest Packet	99.4665	99.4643	95.8333
Smallest Packet	99.5258	99.5238	99.5833
Largest Gap	99.6443	99.1071	98.6905
Smallest Gap	99.5258	99.1071	99.6429
Average Gap	99.5258	99.7024	99.6429

Figure 13.29: Random Forest feature selection results

	% Correctly Classified		
Iterations	06Data	16Data	21Data
20	99.5258	99.5833	99.6429
40	99.5258	99.5238	99.7024
80	99.5258	99.5238	99.6429
100	99.5851	99.5833	99.7024
250	99.5851	99.5833	99.7024
500	99.5851	99.5833	99.7024

Figure 13.30: Random Forest iterations results

MaxDepth	06Data	16Data	21Data
1	77.0599	72.619	60.6548
2	99.1701	99.1667	87.7381
3	99.5851	99.3452	93.631
4	99.5851	99.3452	97.9167
5	99.5258	99.5238	99.2262
6	99.5851	99.5238	99.7024
8	99.5851	99.5833	99.7024
12	99.5851	99.5833	99.7024
15	99.5851	99.5833	99.7024
Infinite	99.5851	99.5833	99.7024

Figure 13.31: Random Forest maximum depth results

Attribute	06Data	16Data	21Data
Baseline	99.4665	99.5238	99.2262
Total Packets	99.4072	99.4048	99.5833
Largest Packet	99.4665	99.3452	94.6429
Smallest Packet	99.348	99.2262	98.9286
Largest Gap	99.6443	98.9881	98.2738
Smallest Gap	99.4665	99.5238	99.2262
Average Gap	99.4665	99.5238	99.2262

Figure 13.32: REP Tree feature selection results

```
=== Predictions on user test set ===

    inst#     actual  predicted error prediction
        1        1:? 2:B.Netcam       1
        2        1:? 2:B.Netcam       1
        3        1:? 2:B.Netcam       0.93
        4        1:? 2:B.Netcam       1
        5        1:? 2:B.Netcam       0.51
        6        1:? 2:B.Netcam       0.73
        7        1:? 2:B.Netcam       0.73
        8        1:? 2:B.Netcam       0.69
        9        1:? 2:B.Netcam       0.51
       10        1:? 2:B.Netcam       0.93
       11        1:? 2:B.Netcam       0.93
       12        1:? 2:B.Netcam       0.93
       13        1:? 2:B.Netcam       0.69
       14        1:? 2:B.Netcam       0.93
       15        1:? 2:B.Netcam       1
       16        1:? 2:B.Netcam       0.93
       17        1:? 2:B.Netcam       0.93
       18        1:? 2:B.Netcam       1
       19        1:? 2:B.Netcam       1
       20        1:? 2:B.Netcam       0.73
       21        1:? 2:B.Netcam       0.69
       22        1:? 2:B.Netcam       1
       23        1:? 2:B.Netcam       0.93
       24        1:? 2:B.Netcam       0.93
       25        1:? 2:B.Netcam       0.93
```

Figure 13.33: Example of WEKA output

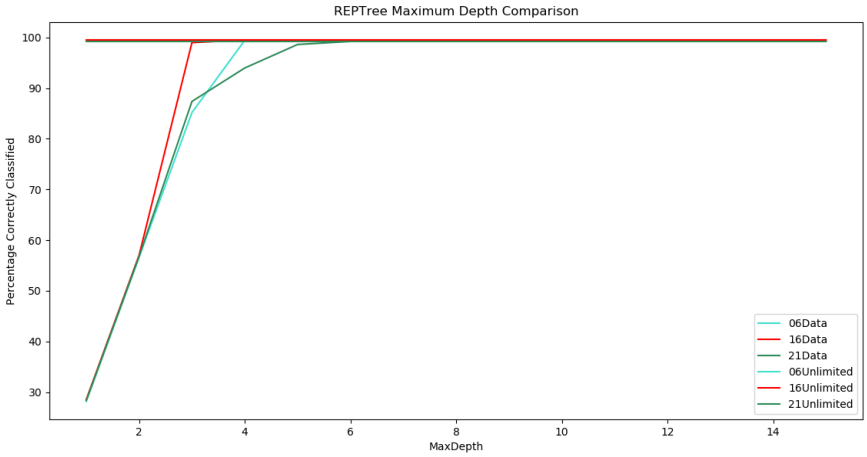


Figure 13.34: REP Tree maximum depth results

Random Forest

=== Confusion Matrix ===

a	b	c	d	e	f	g	<-- classified as
721	0	0	0	0	0	0	a = A.Echo
0	721	0	0	0	0	0	b = B.Netcam
0	0	721	0	0	0	0	c = Camera
0	0	0	721	0	0	0	d = Hive
0	0	0	0	721	0	0	e = Lamp
0	0	0	0	0	721	0	f = S.Plug
0	0	0	0	0	0	721	g = S.Things

=== Summary ===

Correctly Classified Instances	5047	100 %
Incorrectly Classified Instances	0	0 %
Kappa statistic	1	
Mean absolute error	0.0004	
Root mean squared error	0.0072	
Relative absolute error	0.1563 %	
Root relative squared error	2.0587 %	
Total Number of Instances	5047	

Time taken to build model: 0.67 seconds

REPTree

=== Confusion Matrix ===

a	b	c	d	e	f	g	<-- classified as
720	0	0	0	1	0	0	a = A.Echo
1	720	0	0	0	0	0	b = B.Netcam
0	0	721	0	0	0	0	c = Camera
1	0	1	719	0	0	0	d = Hive
0	0	0	0	721	0	0	e = Lamp
0	0	0	0	0	721	0	f = S.Plug
0	0	0	0	0	0	721	g = S.Things

=== Summary ===

Correctly Classified Instances	5043	99.9207 %
Incorrectly Classified Instances	4	0.0793 %
Kappa statistic	0.9991	
Mean absolute error	0.0004	
Root mean squared error	0.014	
Relative absolute error	0.1611 %	
Root relative squared error	4.0138 %	
Total Number of Instances	5047	

Time taken to build model: 0.06 seconds

Logistic Function

=== Confusion Matrix ===

a	b	c	d	e	f	g	<-- classified as
679	1	1	40	0	0	0	a = A.Echo
0	699	0	20	0	2	0	b = B.Netcam
0	2	719	0	0	0	0	c = Camera
23	1	0	697	0	0	0	d = Hive
0	0	0	0	721	0	0	e = Lamp
0	0	0	0	0	721	0	f = S.Plug
0	0	0	0	0	2	719	g = S.Things

=== Summary ===

Correctly Classified Instances	4955	98.1771 %
Incorrectly Classified Instances	92	1.8229 %
Kappa statistic	0.9787	
Mean absolute error	0.009	
Root mean squared error	0.0647	
Relative absolute error	3.6566 %	
Root relative squared error	18.4969 %	
Total Number of Instances	5047	

Time taken to build model: 1.44 seconds

J48

=== Confusion Matrix ===

a	b	c	d	e	f	g	<-- classified as
718	2	0	0	1	0	0	a = A.Echo
0	719	0	0	0	2	0	b = B.Netcam
0	0	721	0	0	0	0	c = Camera
2	0	1	718	0	0	0	d = Hive
0	0	0	0	721	0	0	e = Lamp
0	0	0	0	0	721	0	f = S.Plug
0	0	0	0	0	0	721	g = S.Things

=== Summary ===

Correctly Classified Instances	5039	99.8415 %
Incorrectly Classified Instances	8	0.1585 %
Kappa statistic	0.9982	
Mean absolute error	0.0009	
Root mean squared error	0.0212	
Relative absolute error	0.3662 %	
Root relative squared error	6.0517 %	
Total Number of Instances	5047	

Time taken to build model: 0.01 seconds

Figure 13.35: Training the models comparison

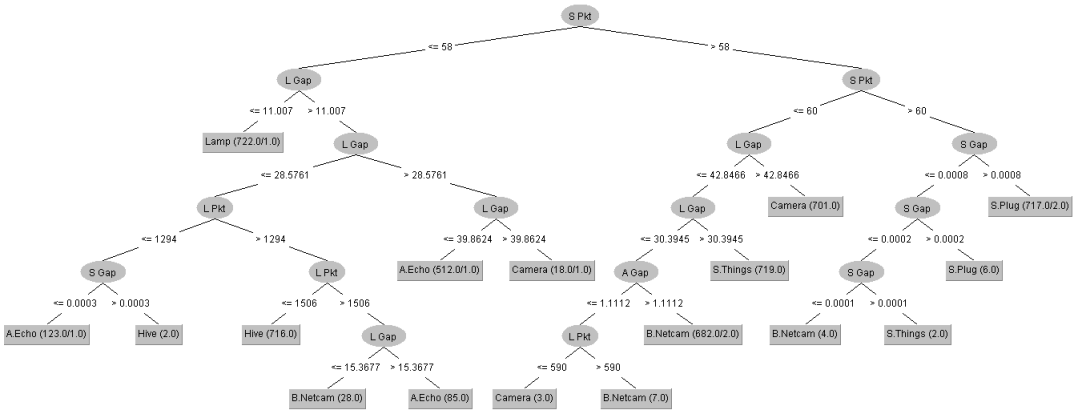


Figure 13.36: J48 model in WEKA

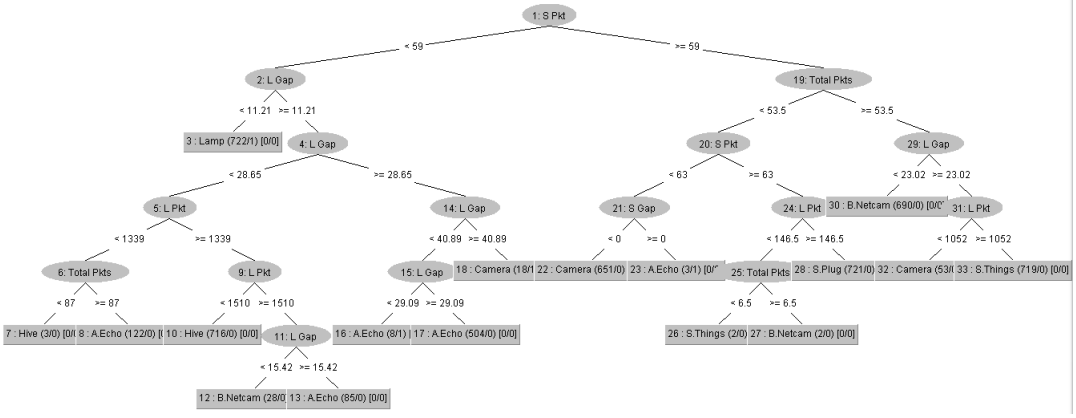


Figure 13.37: REP Tree model in WEKA

References

[1] Anon. Iotscanner: Detecting and classifying privacy threats in iot neighborhoods.

[2] Anon. Zeror. http://chem-eng.utoronto.ca/~datamining/dmc/zeror.htm. Accessed: 29-04-2018.

[3] Eirini Anthi. Cardiff university iot testbed. https://d3c37n1gbqjffz.cloudfront.net/data.html, 2018. Accessed: 03-05-2018.

[4] Eirini Anthi, Lowri Williams, and Pete Burnap. Pulse: An adaptive intrusion detection for the internet of things.

[5] Jason Brownlee. How to compare the performance of machine learning algorithms in weka. https://machinelearningmastery.com/compare-performance-machine-learning-algorithms-weka/, 2016. Accessed: 16-04-2018.

[6] Lakshmi Devasena C. Comparative analysis of random forest, rep tree and j48 classifiers for credit risk prediction, 2014.

[7] Nitesh Dhanjani. *Abusing the Internet of Things*. O'Reilly, 2015.

[8] Niklas Donges. Towards data science - the random forest algorithm. https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd. Accessed: 29-04-2018.

[9] Sam Drazin and Matt Montag. Decision tree analysis using weka.

[10] A. Selcuk Uluagac et al. A passive technique for fingerprinting wireless devices with wired-side observations.

[11] John Hunter et al. Matplotlib home. https://matplotlib.org/, 2018. Accessed: 03-03-2018.

[12] Philippe Biondi et al. Scapy documentation. https://scapy.readthedocs.io/en/latest/index.html, 2018. Accessed: 28-02-2018.

[13] Sergey Bratus et al. Active behavioral fingerprinting of wireless devices, 2008.

[14] Tom Augspurger et al. Pandas - home. https://pandas.pydata.org/, 2018. Accessed: 03-03-2018.

[15] Hak5 Gear. Hakshop products. https://hakshop.com/products/wifi-pineapple, 2018. Accessed: 18-02-2018.

[16] Melanie Greenwood. Ohio couple terrorized after hacker takes over baby-monitoring camera. http://www.nydailynews.com/news/national/baby-monitoring-camera-hacked-taunts-family-article-1.1771399#ixzz30I8zJ08B, 2014. Accessed: 05-02-2018.

[17] Gaganjot Kaur and Amit Chhabra. Improved j48 classification algorithm for the prediction of diabetes. https://pdfs.semanticscholar.org/2456/a979fbe8eea47b90d625c1a064162be5382e.pdf, 2014. Accessed: 02-05-2018.

[18] Yuhua Li. Emerging technology - deep learning lecture. https://learningcentral.cf.ac.uk/webapps/blackboard/execute/content/file?cmd=view&content_id=_4570081_1&course_id=_381761_1, 2018. Accessed: 06-05-2018.

[19] Hak5 LLC. Wifi pineapple home. https://www.wifipineapple.com, 2018. Accessed: 18-02-2018.

[20] Michael McKinney. Change the way you see yourself. https://www.leadershipnow.com/leadingblog/2008/05/, 2008. Accessed: 07-05-2018.

[21] The University of Waikato. Weka class documentation reptree. http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/REPTree.html. Accessed: 29-04-2018.

[22] The University of Waikato. Weka class documentation smo. http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html. Accessed: 29-04-2018.

[23] The University of Waikato. Weka - home. https://www.cs.waikato.ac.nz/ml/weka/, 2018. Accessed: 06-05-2018.

[24] University of Waikato. Future learn - cross-validation. https://www.futurelearn.com/courses/data-mining-with-weka/0/steps/25384. Accessed: 02-05-2018.

[25] University of Waikato. Weka class logistic. http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/Logistic.html. Accessed: 02-05-2018.

[26] Alun Preece. Knowledge management - lecture. https://learningcentral.cf.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_381753_1&content_id=_4257247_1&mode=reset. Accessed: 06-05-2018.

[27] ScienceProg. Simple explanation of naive bayes classifier. https://scienceprog.com/simple-explanation-of-naive-bayes-classifier/, 2016. Accessed: 02-05-2018.

[28] Dr Jianhua Shao. Large scale databases - data mining lectures. Accessed: 06-05-2018.

[29] Statista. Internet of things - number of connected devices worldwide 2015-2025. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. Accessed: 05-02-2018.

[30] Stephanie. What is root mean square error (rmse)? http://www.statisticshowto.com/rmse/, 2017. Accessed: 06-05-2018.

[31] Dan Sullivan. What's on your network? the need for passive monitoring. http://www.tomsitpro.com/articles/network_monitoring-netflow-it_security-networking-snmp,2-561-2.html, 2013. Accessed: 06-05-2018.

[32] Wireshark. Wireshark home. https://www.wireshark.org/, 2018. Accessed: 19-02-2018.