Authenticating Raspberry Pi Devices

Hardware authentication of devices using polynomial regression and classification

Benjamin Frost, supervised by Dr Nikos Komninos

2019

# Contents

[Contents 1](#_Toc28528657)

[1 Introduction 3](#_Toc28528658)

[2 Related Studies 3](#_Toc28528659)

[2.1 Comparisons with Branstett’s paper 4](#_Toc28528660)

[3 Setup 4](#_Toc28528661)

[3.1 Data Collection 5](#_Toc28528662)

[3.1.1 Client 6](#_Toc28528663)

[3.1.2 Server 6](#_Toc28528664)

[4 Research 7](#_Toc28528665)

[4.1 Formatters 8](#_Toc28528666)

[4.2 Polynomial Regression 9](#_Toc28528667)

[4.3 Exploring Classifiers 11](#_Toc28528668)

[4.3.1 Naïve Bayes 12](#_Toc28528669)

[4.3.2 Random Forest 13](#_Toc28528670)

[4.3.3 Logistic Regression 13](#_Toc28528671)

[4.3.4 Support Vector Machine 13](#_Toc28528672)

[4.4 Support Vector Machine 14](#_Toc28528673)

[5 Device Authentication 16](#_Toc28528674)

[6 Conclusion 16](#_Toc28528675)

[6.1 Limitations 17](#_Toc28528676)

[6.2 Application 17](#_Toc28528677)

[7 Appendix 18](#_Toc28528678)

[7.1 Java Code 18](#_Toc28528679)

[7.1.1 Client.java 18](#_Toc28528680)

[7.1.2 Server.java 20](#_Toc28528681)

[7.1.3 Point.java 22](#_Toc28528682)

[7.1.4 Save.java 23](#_Toc28528683)

[7.1.5 Device.java 24](#_Toc28528684)

[7.1.6 Formatter.java 29](#_Toc28528685)

[7.1.7 Formatter50.java 35](#_Toc28528686)

[7.1.8 Formatter60.java 36](#_Toc28528687)

[7.1.9 Formatter70.java 37](#_Toc28528688)

[7.1.10 Formatter80.java 38](#_Toc28528689)

[7.1.11 Device.java 39](#_Toc28528690)

[7.2 Python Code 43](#_Toc28528691)

[7.3 Formatter Variables 47](#_Toc28528692)

[7.3.1 Formatter 50 47](#_Toc28528693)

[7.3.2 Formatter 60 48](#_Toc28528694)

[7.3.3 Formatter 70 48](#_Toc28528695)

[7.3.4 Formatter 80 48](#_Toc28528696)

[7.4 Classification Metrics 49](#_Toc28528697)

[7.4.1 Support Vector Machine 49](#_Toc28528698)

[7.4.2 Logistic Regression 50](#_Toc28528699)

[7.4.3 Naïve Bayes 51](#_Toc28528700)

[7.4.4 Random Forest 51](#_Toc28528701)

[7.4.5 K-Means Clustering 52](#_Toc28528702)

[8 Bibliography 52](#_Toc28528703)

# Introduction

The Internet of Things (IoT) is a widely understood new area of Computer Science, however possible methods of authentication of such devices are yet to be fully explored. The industry standard for verifying the authenticity of devices up until now has been to create an encryption certificate for said device and use this to ensure that devices are who they say they are. Imagine a smart home with sensors and smart devices in every room. You may have a fridge with sensors to read how much food you have available, thermometers in every room, or smart bulbs adjusting their colour temperature depending on your mood. For some of these devices, the processing power simply will not be available to run complex encryption algorithms to keep the data transfers secure. The alternative then, may be to look at other methods of verifying devices.

When it comes to the IoT, we are likely to have many very small devices in the future all transmitting data that will be personal to the owner and need to be kept private. This poses a few new problems when it comes to authentication. “As is often the case, IoT technology has moved more quickly than the mechanisms available to safeguard the devices and their users.” (Rouse, 2018). As developments are made into creating cheaper, more useful devices, perhaps not enough is being done to ensure that all avenues of information security for these devices are being explored. Additionally, as the demand for these small devices grows, pressure will be put on manufacturers to find solutions to these important security questions.

In this project I will be exploring how IoT devices can be authenticated not using an encryption certificate, but by using the hardware features of such devices. Having the ability to bypass the need to generate encryption certificates will allow the use of lower powered devices by reducing the amount of processing power required. Machine Learning is still a relatively new field in the area of Computer Science and its applications are still being researched. I will be looking at using Machine Learning methods to analyse and classify devices so that they may be authenticated as being who they say they are.

# Related Studies

This project was inspired by a paper by Jonathan Branstett (2016), looking at device authentication techniques using device hardware features. The aim of his paper was to verify if authentication based on hardware features was possible, and the types of variables that may be measured for best results. I aimed to build upon this work by looking at the real-world application of hardware feature device authentication and how practical it may be. Jonathan laid much of the groundwork for this project and provided the basis of the Java server and client code that I have built upon.

Other related studies are described below.

Park, Kim and Lim (2015) designed a framework to manage mutual authentication of devices based on Public Key Identification (PKI). They present a lighter certificate that could be used by smaller devices in such infrastructure. However, this assumes that the devices have enough computing capability to perform public key cryptography which is not always the case (Shin, Yeh and Kim, 2012). This issue can be bypassed either by providing more computational capabilities to devices or by using authentication scheme lighter than PKI.

Hirano, Okuda, Yamaguchi (2007), Diez, Touceda and Zeadally (2015) both proposed the use of smart cards to augment the storage and computing capabilities of devices in order to use public keys for authentication. They both used the Java Card technology which allow Java-based application to be run on smart cards.

The former designed a mutual device authentication framework based on the use of smart cards as USB device. The authentication is based on the IPsec protocol and use the device public key certificate as well as production-level attributes (like the device serial number) and owner-level attributes (like unique name chosen by the owner).

The latter propose wearable devices to be built with a smart card inside as these would be small enough to be built inside most devices. They designed a protocol based on Point-to-Point Protocol (PPP) and Extensible Authentication Protocol (EAP) using challenge-response and digital signature techniques to authenticate the wearable device to other entities of the system.

Shin, Yeh and Kim (2012) proposed an authentication scheme that avoid the use of public key cryptography and therefore does not need any additional device to improve the computing capabilities. They defined a light protocol only based on the use of hash functions, random number generators, bitwise operations (XOR) and a group key using the device serial number and a defined bit sequences called digital DNA to authenticate the devices.

## Comparisons with Branstett’s paper

I studied Branstett’s paper to see which techniques worked the best for him. Within his paper, polynomial regression was used to identify each device. If the known coefficients calculated through polynomial regression for the Raspberry Pi were the same as the coefficients calculated using the hardware data sent to the server, the device would be authenticated as being the claimed device.

Within my project however, I wanted to take a slightly different approach. Using the data file generated through the connection with the Raspberry Pi, I wanted to be able to use a classification model to predict the identity of the device.

The fundamental difference between Branstett’s project and my own was that Branstett was programming multiple identities into his Raspberry Pi, and having each identity transmit its data to the server. Each identity would have slightly different hardware characteristics, simulating different virtual devices within the Raspberry Pi. The server would then compare the known coefficients of each identity to the calculated coefficients and output its authenticity. In contrast to this, I wanted my project to be able to authenticate the Raspberry Pi itself, and not the virtual devices within the Pi. I was able to do this since I had access to more than one device. Despite these differences, a large proportion of the data collection Java code in my project was developed by Branstett.

# Setup

In terms of exploratory data analysis, the choice was easy to use Python as the main language for implementation of our Machine Learning regression and classification algorithms. Although the ability to implement the needed techniques was available in Java using the Apache Commons libraries, the breadth of knowledge and support online for Python, along with its user-friendly implementation of ML algorithms, outweighed the usefulness of using one language for the whole project. As a result of this choice, we ended up using both Java and Python in the final project. This was less of an issue than initially expected, and as I grew more confident using both languages concurrently, I was able to integrate python into the project without many problems.

Within Python, SciKit-Learn and Pandas provided the algorithms used for regression and classification, and dataset management respectively. The wealth of support online for these libraries served the project greatly and was a valuable resource in fixing errors. The full list of libraries is included in the appendix.

During the exploratory research stage of this project, I used IBM Watson to host a notebook online to run my Python code. The benefits of using this method are that it is easily sharable, you can backup your data and code to the cloud to access it from any device, and it provides a large amount of processing power to be able to run many classification algorithms.

## Data Collection

The concept that I wanted to implement with this project was that the Raspberry Pi device represented an IoT device. Therefore, I wanted the least amount of processing to happen on the device and instead leave everything to the much more highly powered server.

The data collection for this project was already fairly thoroughly developed. Branstett’s Java code laid the groundwork for transmitting and collecting this data for further analysis. For use in this project, I stripped down the project developed by Branstett to include just the essential data collection elements of the code. Since I would be implementing my own data analysis and classification algorithms in Python, I did not need to include the previously written analysis in Java.

The method for data collection was therefore as follows:

1. The client transmits to the server the device name such as HomePi or CityPi.
2. The client measures hardware data using the specified formatter (in the final project this was the 70% constant data formatter).
3. The client stores this raw data in a list, with each entry representing a different hardware feature.
4. This list is then sent to the server over LAN.
5. The server receives this data and sends it for further formatting before saving it to a CSV file.

This process was the primary use of the Java code in this project. The only other time the Java code was used was when displaying the output of the Python code.

The formatters used were set up so that different amounts of constant and variable data were collected in each run of the program. For example, we used the following formatters, with each percentage relating to the amount of variable data being collected in each run:

* 80%
* 70%
* 60%
* 50%

The full list of variables used for each formatter can be found in the appendix.

For each run of the program, the client measures metrics about itself and communicates these to the server. Both client and server run Java code to measure and receive the data. The client will measure data about itself 54 times, with processes running in the background. This is so that data can be measured over a period of time and a consistent dataset can be measured.

Each data point measured has an X and Y value. For example, CPU usage vs time, or free memory vs total memory.

### Client

Two Raspberry Pi devices were used for this project, one in central London in the City, University of London Computer Science offices, and the other at home. These were used to test the validity of my classification algorithms since I wanted to be able to tell devices apart from one another. The Raspberry Pi represents an IoT device, and allows us to easily measure metrics about the device such as performance under load, temperature, memory usage etc. The Raspberry Pi device used at home was the Raspberry Pi 4. This device is relatively highly powered for a computer of its size and cost, and this allowed me a degree of flexibility when developing the code to run on the client. Comparatively, the Pi used in London was the model 3 which was less powerful. Within the Python code, the City and Home Raspberry Pi devices are referred to as CityPi and HomePi respectively. Therefore, within this report I will continue to use those abbreviations for the sake of clarity.

The reason for using Raspberry Pi devices for this project was primarily since they supported easy and accurate measurements of hardware data using the ‘vcgencmd’ command. This command allowed a range of data to be collected easily through the terminal, and examples of such data can be found in the appendix. Additionally, Raspberry Pi devices are low cost and easily configurable to this specific project due to their Linux operating system.

### Server

The server in this project was a Windows 10 desktop computer which ran the server-side Java code. In the final project all processing and results were calculated server-side, so that the least amount of processing was allocated to the client, which in our case was a Raspberry Pi.

#### Development

In the early stages of the project, I wrote the code for the client and compiled it to a Jar file, and then transferred this across to the Raspberry Pi. This iterative method of development grew to be very time consuming and it meant that I was not able to make small changes to my code easily. I soon realised that since the HomePi was powerful enough, I was able to install IntelliJ Idea onto the device and edit code on-the-go. This saved me a great deal of time when I needed to make small changes since they would be implemented instantly.

I was not able to implement the same method with the CityPi however, since the RAM available was not enough to run the operating system and the IDE. Therefore, when I had a working client Jar file that functioned on my Home Pi, I was able to transfer this across to the City Pi. The fact that I was able to run an IDE on at least one of the Pis was a great help in terms of speeding up the development of my program.

#### Access

I needed to be able to log into the devices remotely, especially since the CityPi was situated in London. Both devices ran in headless mode without a monitor, so it was essential to implement a method of viewing the desktop. I explored two different solutions to be able to access the Raspberry Pi on the go. Firstly, I used TeamViewer to access the Pis. This was effective in the sense of giving a fluid and easy to use UI, however I found the client to be resource heavy, which affected the performance of the CityPi. Secondly, I used RealVNC. This comes preinstalled on the devices and is streamlined to give reliable performance. The switch to this software helped the performance of both Pi devices and made it easier to implement my code.

Unfortunately, I did encounter issues when it came to transmitting data across the internet. The implementation of peer to peer data transfer was beyond the scope of this project, and therefore I was limited to recording data over LAN. Despite this, I was able to record data from both devices and generate an accurate model based on that data, described below.

# Research

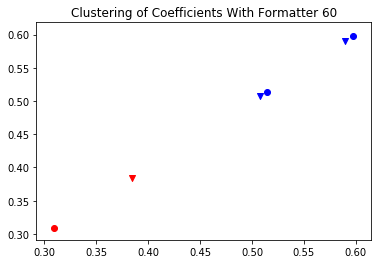
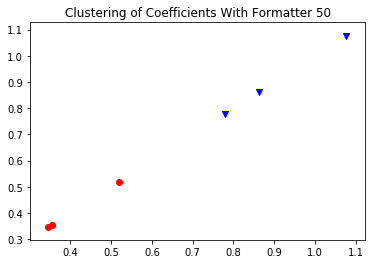
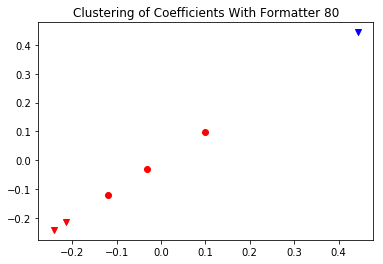
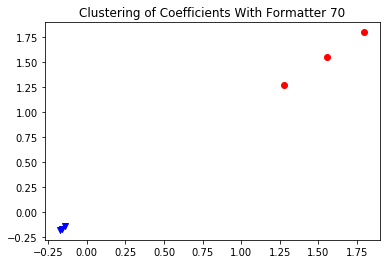
The initial stage of this process was exploratory. There was a strong need to understand the data I had available and the possible pathways I could take to implement this authentication method. Therefore, research played a large role in determining the path that this project would take. During the course of the project, I used many online notebooks as I experimented using different approaches to analyse the data. I was able to share these notebooks easily and gain feedback on the results I was producing with the help of Dr Komninos. The online link to the notebook used for research purposes can be found in the appendix.

The decision was made early on to use polynomial regression to generate one or two coefficients that could represent the large amounts of data that were being generated through communication with the devices. Each data file generated through running the program and authenticating the device ended up being between 280KB and 330KB in size. Therefore, it was essential to be able to condense this down into a few identifying numbers. Thankfully this approach produced coefficients that could be used to accurately differentiate between the devices.

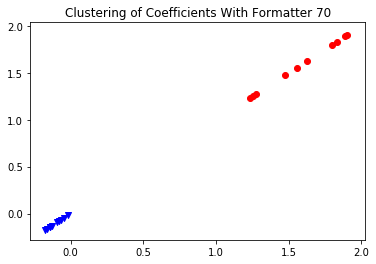
## Formatters

Collecting data using the different formatters in the program was essential to this exploratory research stage. I collected 3 datasets from each Pi, at different times of day and with different background processes running. I then repeated this for each formatter. This gave me a moderate amount of data for each formatter, and I was then able to compare which formatter was going to give me the best classification results.

The graphs below show the results I produced through comparing the different formatters. Each point represents 1 coefficient generated from each dataset through linear regression. The colours represent the clustering groups calculated through k-means clustering, and the different markers represent the actual groups each point belongs to. The full parameters can be found in section 8.4.5 of the appendix.

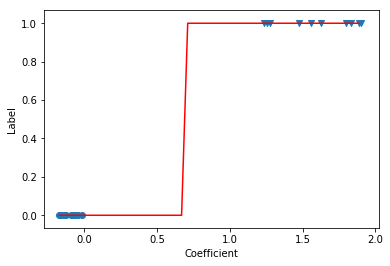


We can see from these graphs that both the 50% formatter and the 70% formatter correctly clustered the data into each device. Despite this, the 70% formatter graph shows a very clear separation between the two devices, which will make it easier to classify the two devices when we have a very large dataset.

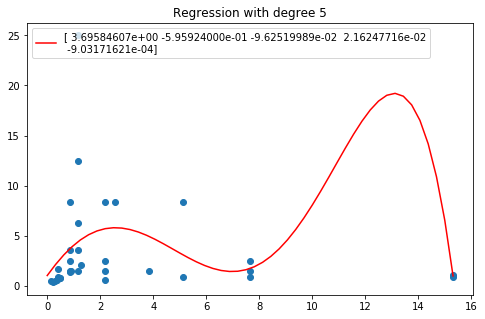
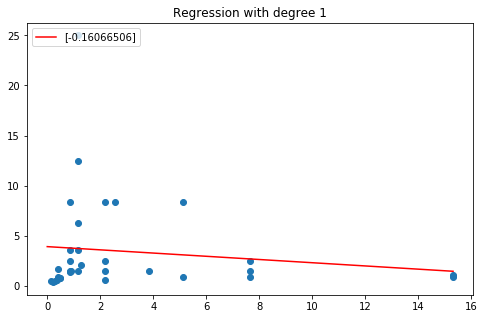
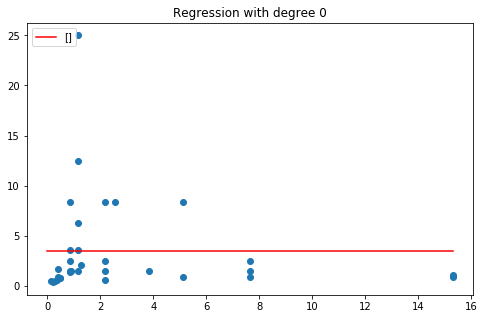
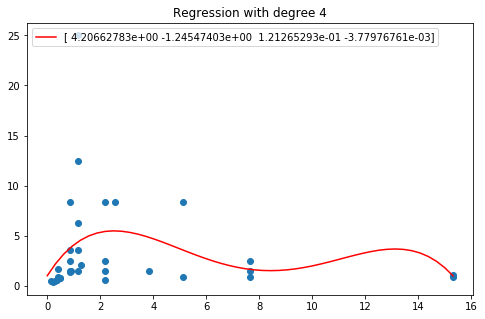
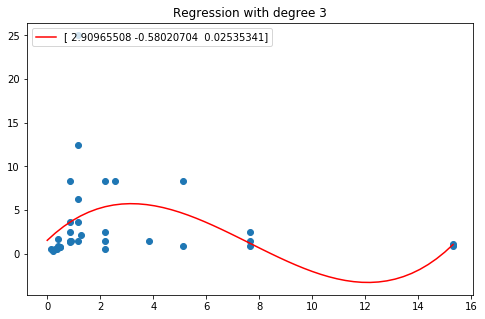
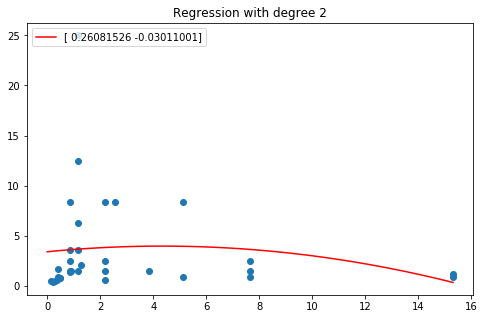
The 80% constant data formatter did not provide a clear separation between the two devices, and as such the clustering algorithm was not able to separate the two devices with any accuracy. The below graph shows just how easily the K-Means clustering Algorithm was able to separate the devices when using the 70% formatter and a dataset of 20 samples.

## Polynomial Regression

I started the project classifying the devices using one coefficient to represent each dataset generated. This worked initially, however one of the drawbacks to using a one-dimensional graph to classify my data was that edge cases from data generated from each device would be at risk of being classified incorrectly.

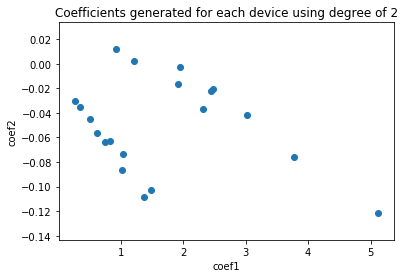
 The graph below shows how the SVM with a degree of 1 performed on our one-dimensional graph, trained with 70% data, scored on 30% data, and displayed using the entire dataset as the input.

We can see that it performs well, and classifies the devices correctly depending on their labels. The issue with this approach is that if we were to measure data from a device under heavy load, the data collected may land much closer to the other device’s coefficients, and therefore be incorrectly classified. To prevent this, I also explored the use of two coefficients as described below.

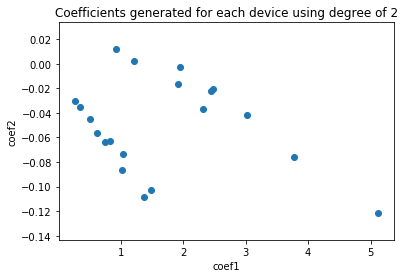
Here we can see different degrees of regression algorithms performed on the same dataset. Each dataset is run through a weighting algorithm to reduce the range within which we are working. The axis here are arbitrary, since each point is representing different information about the device.

Degrees from 3 show that the data is being overfitted. Because of this, I decided to run the regression algorithm with degree 2 going forward.

Since at this stage I knew that I was going to be focussing on the 70% constant formatter, I generated 20 datasets using this system and ran my analysis on this relatively large dataset. For each dataset generated, I ran polynomial regression with a degree of 2. The graph below shows all the datasets from both devices plotted on a graph using each coefficient as the x and y values.



We can see very clearly from this graph that if I were to have used just one coefficient to identify each dataset, there would have been a significant amount of overlap between devices and I would have not been able to classify the devices with any acceptable degree of accuracy. However, since I used two coefficients there is a very clear separation between devices.



Device 1

Device 2

The 2-coefficient approach also meant that edge cases were far less likely to be able to skew the results of my classifier. We can see this in the above graph since there is one data point highlighted in red which may seem like an outlier, however since it falls within the range of coefficients for each device, it would still be classified correctly.

## Exploring Classifiers

When it came to deciding which classification algorithms I would be testing within this project, it became clear that there were many different options that would produce the results that I was looking for. For this project, I decided to review the four main classification algorithms, Naïve Bayes, Random Forest, Logistic Regression, and Support Vector Machine. (Chen, 2011). For a dataset such as the one used in this project with 20 samples, many algorithms would perform similarly. Chen is clear to point out that for larger datasets, it is not always the most accurate classifier that should be chosen. Factors such as speed, ability to interpret results, and the ability to explain the process behind each classifier are important.

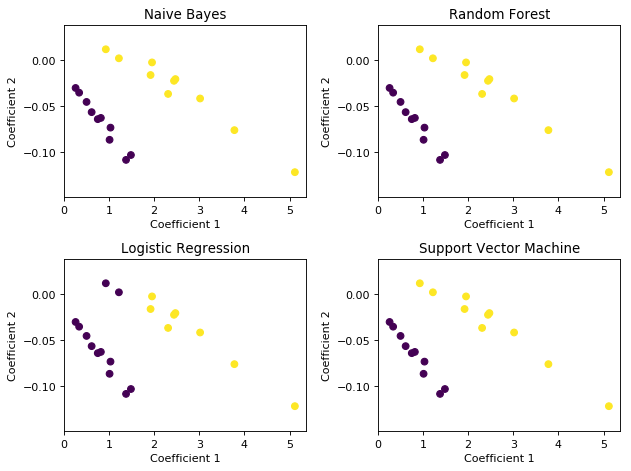
I also found that for the classifiers below, their accuracy would vary greatly on each run of the program. This is because the python code was randomly splitting the data into training and testing sets with a ration of 70:30. The randomness in generating the classifiers then caused the reported accuracy of each classifier to vary by up to 10% on each run of the program. In the deliverable classifier used in the final project, the entire 20 sample dataset was used to train the classifier so that the greatest accuracy could be achieved on the unseen data from the Pis.

The parameters of each classifier can be found in section 8.4 along with the metrics. Each classifier was trained on 70% of the data and scored by testing itself on the remaining 30% test set. The colour groups on each classifier trained on 70% data, and predicting the outcomes of the entire dataset.

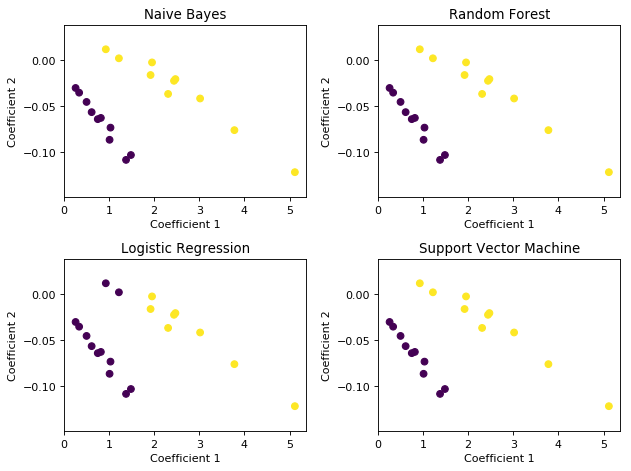
We can see from the below that each classifier scored a perfect 1.00 on the training set. If we had used a larger dataset with which to test our classifiers we would have been able to see larger variance in results.

Since the data collected to train our classifier can quite easily be separated into each device, we get similar results from each classifier. There is also no overlap between devices within our data, so a clean linear line can be drawn between the two groups.

### Naïve Bayes

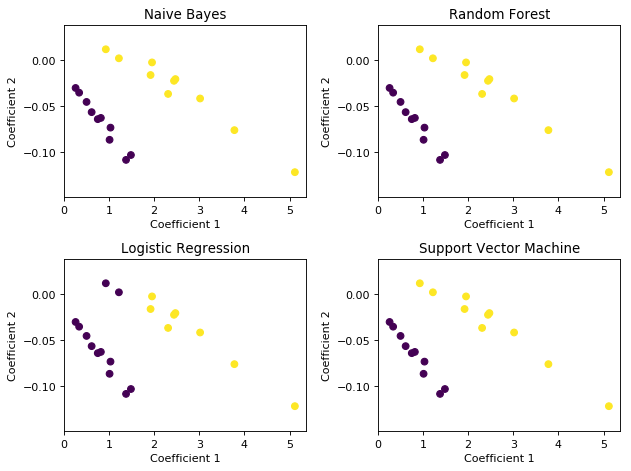
Score: 1.00

### Random Forest

Score: 1.00

### Logistic Regression

Score: 1.00



### Support Vector Machine

Score: 1.00

**Decision Function:**

[-0.97984064 -1.02751492 -1.00024654 -1.15130345 -1.36028516 -1.12546558

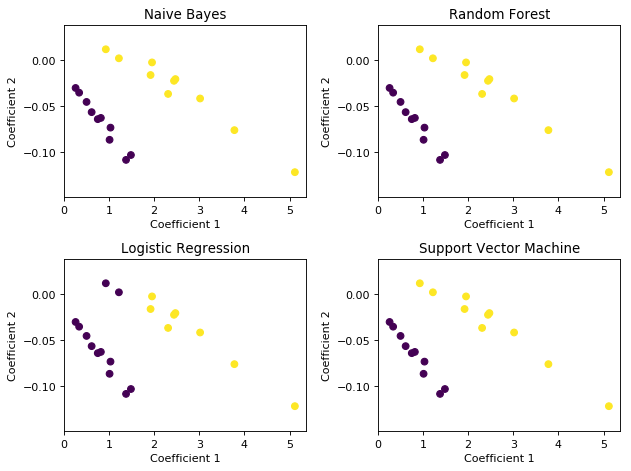
-1.15608185 -1.02797916 -1.40736279 -0.99950887 2.53097575 2.71135733

1.15265689 0.99977084 1.67923071 2.05057578 2.21804555 3.40709608

2.30799621 1.66141698]

**Hinge Loss Value**: 0.0010439825203261532

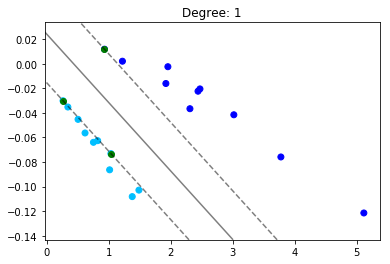
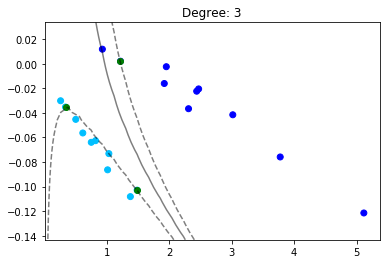
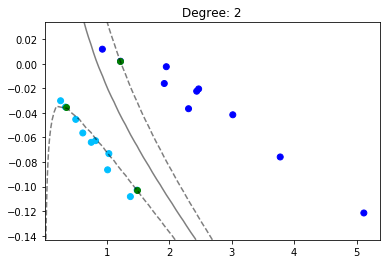
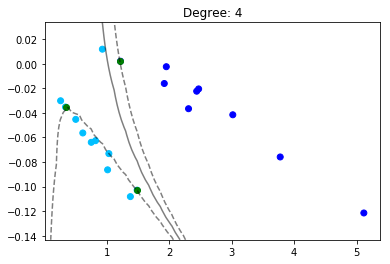
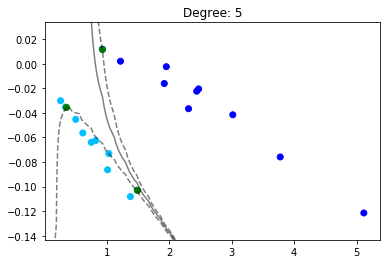
The hinge loss value of this classifier is calculated by using the SciKit-learn library ‘metrics’ with the decision function of this classifier. Therefore, the hinge loss value for this classifier was calculated to be 0.0010 (2 sf) which shows us that the classifier was able to correctly classify our unseen data to a good level of accuracy. The hinge loss value is not quite 0 since some points within our data to the top left of the graph were close to the boundary and caused uncertainty.



The full metrics data for the classifiers used can be found in section 8.4. As seen in this data, many of our classifiers perform without fault on the small dataset provided. The SVM classifier performed with an accuracy of 100% when tested in Python using SciKit-Learn, however just 90% accuracy when testing within Weka using cross validation using 10 folds. I wanted to be able to explore the use of the SVM and see if changing the degree of the classifier would help to produce a more accurate result. The SVM was also explored further since its boundary can be easily visualised and compared within the python code. I believe that in a real-world application of this authentication technique, other classifiers would perform just as well as an SVM. These classifiers need to be tested on a much larger dataset in order to clearly see which would be the best selection.

## Support Vector Machine

I wanted to be able to explore the accuracy of the different degrees of the SVM classifier to see if any increase in performance could be obtained. Overfitting is a natural side effect of choosing a classifier with a high degree, and this can be seen in the graphs below.



From the above classifications we can see that anything above the degree of 1 would give us overfitting of our test set. The coefficients follow a linear line, and therefore the best classifier to match that would be the SVM with a degree of 1. Underfitting was not an issue with our dataset, since the simplest scenario with our classifier would be a linear line drawn to separate our two devices.

Now that we have a model with a reliable classifier, we just need to export this model to a .joblib file and load it into the python script each time we would like to authenticate a new device.

# Device Authentication

Authentication in this project is comprised of two main steps: Data collection, and data analysis. Java code based off of Branstett’s paper was used for collection, and python code was used for the analysis and prediction using our classification model.

1. The client Jar file and the server Jar file are initiated using the respective .bat files on each device.
2. A connection is established between the client and the server. This is made over LAN and using the server IP address.
3. The client sends the server it’s device name. Within the scope of this project, this is limited to CityPi or HomePi. This is the name that will be compared to the predicted name of the device using our model.
4. The client starts to send over data about the device. 54 rows of data are sent over, measuring variables described by the 70% constant data formatter.
5. Once 54 rows are received, the server breaks connection with the client.
6. The server then formats the data received, storing it in a .csv file within the project folder. This formatting includes converting data to the same filetype, and ensuring each row is represented on a new line.
7. The server then calls the authenticate.py script and waits for it to finish.
8. The python script then loads in the .csv data file and performs further formatting on the data. Each datapoint is weighted, and duplicates are removed.
9. Polynomial regression with a degree of 2 is run on the data. This generates two coefficients that are then used to represent the data generated from this device.
10. The model is now loaded in and used to predict the identity of the current device using coefficients.
11. The python script then prints to its console the predicted identity of the device and exits.
12. The Java code reads the console output of the python script and compares the claimed identity of the device against the predicted identity of the device. If they are the same, the device is authenticated and this is printed to the console.

# Conclusion

## Limitations

This project was not designed to be a deployable, and the reason for this is mainly that the classification model would have to be entirely retrained for each new situation. Along with this, the amount of data needed to train the model to a good degree of accuracy would take a very long time to generate from each device.

Additionally, the connection between the server and client is only designed to be established over a LAN connection. A connection over the internet would need an intermediary server or require a P2P connection.

As with any classification model, there is a degree of uncertainty with its predictions. Although our model scored 100% accuracy on the data we had available, that’s not to say it would be 100% accurate all the time. Therefore, it’s important to keep in mind that a classification model for device authentication should not be used on anything critical.

## Application

This project has shown that a device authentication model based on classification of hardware features offers many benefits to an IoT network. The ability to offload data processing to the server allows for much lower powered devices to be used in networks, without sacrificing much security. The techniques used in this project could therefore be applied to non-critical infrastructure, such as a home network with many low powered devices.

# Appendix

## Java Code

Credit to Jonathan Branstett. The following code has been modified to suit this project.

### Client.java

package deviceRecognition;  
  
import deviceRecognition.devices.\*;  
import deviceRecognition.formatter.\*;  
  
import java.io.IOException;  
import java.io.ObjectOutputStream;  
import java.io.OutputStream;  
import java.net.Socket;  
*/\*\*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Client {  
  
 static String *ip* = "192.168.0.20";  
 static int *port* = 6789;  
  
 public static void main(String[] args) {  
 try {  
 *test\_successRate\_client*();  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* This method collects data about the device on which it is run 54 times.  
 \* This data is then transferred to the server.  
 \*  
 \** ***@throws*** *IOException  
 \** ***@throws*** *NoSuchFieldException  
 \** ***@throws*** *IllegalAccessException  
 \*/* private static void test\_successRate\_client()  
 throws IOException, NoSuchFieldException, IllegalAccessException {  
 final String[] NAMES = {"device1", "device2", "device3"};  
 final Device[] DEVICES = {  
 new Device(), new Device1\_empty(), new Device1\_full(),  
 new Device1\_approx(), new Device1\_var0(), new Device(),  
 new Device2(), new Device2\_empty(), new Device2\_full(),  
 new Device2\_approx(), new Device2\_var0(), new Device2(),  
 new Device3(), new Device3\_empty(), new Device3\_full(),  
 new Device3\_approx(), new Device3\_var0(), new Device3()  
 };  
  
 Socket s;  
 OutputStream os;  
 ObjectOutputStream oos;  
 Device d;  
 Formatter f;  
 Point[] pts;  
  
 System.*out*.println("Starting write");  
  
 Socket sname = new Socket(*ip*, *port*);  
 OutputStream osname = sname.getOutputStream();  
 oos = new ObjectOutputStream(osname);  
 oos.writeObject("HomePi");  
  
 int j = 0;  
 while (j < 100) {  
 for (String name : NAMES) {  
 for (int i = 0; i < DEVICES.length; i++) {  
 s = new Socket(*ip*, *port*);  
 os = s.getOutputStream();  
 oos = new ObjectOutputStream(os);  
 oos.writeObject(name);  
 d = DEVICES[i];  
 f = new Formatter70(1E15, 1, 0, 8);  
 pts = f.formatData(d, false, false);  
 oos.writeObject(pts);  
 oos.close();  
 os.close();  
 s.close();  
 }  
 }  
 j++;  
 }  
 }  
}

### Server.java

package deviceRecognition;  
  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
*/\*\*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Server {  
  
 static int *port* = 6789;  
 static String *filePath* = "Pi.csv";  
 static String *deviceName*;  
 static String *authenticatedName*;  
  
 public static void main(String[] args) {  
 try {  
 *prepareFile*();  
 *processData*();  
 *authenticate*();  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
  
 */\*\*  
 \* Overwrite the existing data file with a blank file.  
 \*/* public static void prepareFile() {  
  
 try {  
  
 FileWriter wrPts = new FileWriter(*filePath*);  
  
 wrPts.write("");  
  
 wrPts.close();  
  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 }  
  
  
 */\*\*  
 \* This method receives the data from the client and appends it to the data file.  
 \*  
 \** ***@throws*** *IOException  
 \** ***@throws*** *ClassNotFoundException  
 \*/* private static void processData()  
 throws IOException, ClassNotFoundException {  
  
 ServerSocket ss = new ServerSocket(*port*);  
 Socket sname = ss.accept();  
 InputStream isname = sname.getInputStream();  
 ObjectInputStream oisname = new ObjectInputStream(isname);  
 *deviceName* = (String) oisname.readObject();  
  
 System.*out*.println(*deviceName*);  
  
 int i = 0;  
 while (i < 54) {  
 Socket s = ss.accept();  
 InputStream is = s.getInputStream();  
 ObjectInputStream ois = new ObjectInputStream(is);  
 String name = (String) ois.readObject();  
 System.*out*.println("received message from " + *deviceName*);  
 Point[] pts = (Point[]) ois.readObject();  
  
 Save.*savePts*(pts, *filePath*);  
  
 System.*out*.println();  
 is.close();  
 s.close();  
  
 i++;  
 }  
  
 ss.close();  
 }  
  
  
 */\*\*  
 \* This method calls the python file which authenticates the device using the data collected.  
 \* It then reads the output of the python code and outputs the authenticity of the device.  
 \*  
 \** ***@throws*** *IOException  
 \*/* public static void authenticate() throws IOException{  
  
 String s;  
  
 String pyfilepath = "C:\\...\\Device Recognition";  
  
  
 // run the Python file using the cmd:  
 Process p = Runtime.*getRuntime*().exec("cmd /c \"cd " + pyfilepath + " && python authenticate.py\"");  
  
 BufferedReader stdInput = new BufferedReader(new  
 InputStreamReader(p.getInputStream()));  
  
 BufferedReader stdError = new BufferedReader(new  
 InputStreamReader(p.getErrorStream()));  
  
 String lastOutput = "";  
  
 // read the output from the command  
 while ((s = stdInput.readLine()) != null) {  
 System.*out*.println(s);  
 lastOutput = s;  
 }  
  
 *authenticatedName* = lastOutput.split(" ")[1];  
  
 // read any errors from the attempted command  
 while ((s = stdError.readLine()) != null) {  
 System.*out*.println(s);  
 }  
  
 System.*out*.println(*deviceName* + " vs " + *authenticatedName*);  
  
 if (*deviceName*.equals(*authenticatedName*)) {  
 System.*out*.println("Device Authenticated");  
 }  
  
 System.*exit*(0);  
  
 }  
}

### Point.java

package deviceRecognition;  
  
import org.apache.commons.math3.fitting.WeightedObservedPoint;  
  
*/\*\*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Point extends WeightedObservedPoint{  
 private final String xLabel;  
 private final String yLabel;  
 */\*\*  
 \** ***@param*** *weight the weight of the point.  
 \** ***@param*** *xLabel the name of the field used as x coordinate  
 \** ***@param*** *xValue the value of the x coordinate  
 \** ***@param*** *yLabel the name of the field used as y coordinate  
 \** ***@param*** *yValue the value of the y coordinate  
 \*/* public Point(double weight, String xLabel, double xValue,  
 String yLabel, double yValue) {  
 super(weight,xValue,yValue);  
 this.xLabel = xLabel;  
 this.yLabel = yLabel;  
 }  
 public String getXLabel() {  
 return xLabel;  
 }  
 public String getYLabel() {  
 return yLabel;  
 }  
 @Override  
 public String toString() {  
 return "w=" + this.getWeight() + " "  
 + "x=" + this.getX() + " "  
 + "y=" + this.getY() + " "  
 + "(" + this.getXLabel() + " + " + this.getYLabel() + ")";  
 }  
}

### Save.java

package deviceRecognition;  
  
import java.io.\*;  
import java.math.BigDecimal;  
*/\*\*  
 \* The class to perform the saving function  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Save {  
  
 static boolean *firstRun* = true;  
  
 public static void savePts(Point[] pts, String filePath) {  
  
 try {  
  
 PrintStream wr = new PrintStream(new FileOutputStream(filePath, true));  
  
 if (*firstRun*) {  
 *firstRun* = false;  
 for (Point pt : pts) {  
 wr.print(pt.getXLabel() + ",");  
 wr.print(pt.getYLabel() + ",");  
 }  
 wr.println();  
 }  
  
 for (Point pt : pts) {  
  
 BigDecimal Xnum = new BigDecimal(pt.getX());  
 String x = Xnum.toPlainString();  
  
 BigDecimal Ynum = new BigDecimal(pt.getY());  
 String y = Ynum.toPlainString();  
  
 wr.print(x + ",");  
 wr.print(y + ",");  
 }  
  
 wr.println();  
  
 wr.close();  
  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

### Device.java

package deviceRecognition.devices;  
import com.sun.management.OperatingSystemMXBean;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.lang.management.ManagementFactory;  
import java.lang.reflect.Field;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
*/\*\*  
 \* The Device class is used to collect data on the Raspberry PI. All data are made public so they can  
 \* be called by the Formatter classes using reflection.  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Device {  
 // Data collected using Java OperatingSystemMXBean class  
 public String architecture;  
 public int availableProcessors;  
 public long committedVirtualMemorySize;  
 public long freePhysicalMemorySize;  
 public long freeSwapSpaceSize;  
 public String osName;  
 public String osVersion;  
 public double processCpuLoad; //recent CPU usage for the JVM process  
 public long processCpuTime; //CPU time used by the JVM process  
 public double systemCpuLoad; //recent CPU usage in the system  
 public double systemLoadAverage; //System load average for the last minute  
 public long totalPhysicalMemorySize;  
 public long totalSwapSpaceSize;  
 //Data collected using the Raspberry PI vcgencmd command  
 public double clock\_arm;  
 public double clock\_core;  
 public double clock\_h264;  
 public double clock\_isp;  
 public double clock\_v3d;  
 public double clock\_uart;  
 public double clock\_pwm;  
 public double clock\_emmc;  
 public double clock\_pixel;  
 public double clock\_vec;  
 public double clock\_hdmi;  
 public double clock\_dpi;  
 public double volts\_core;  
 public double volts\_sdram\_c;  
 public double volts\_sdram\_i;  
 public double volts\_sdram\_p;  
 public double temperature;  
 public boolean isCodecEnabled\_H264;  
 public boolean isCodecEnabled\_MPG2;  
 public boolean isCodecEnabled\_WVC1;  
 public boolean isCodecEnabled\_MPG4;  
 public boolean isCodecEnabled\_MJPG;  
 public boolean isCodecEnabled\_WMV9;  
 public double memorySize\_arm;  
 public double memorySize\_gpu;  
 public String firmwareVersion;  
 // SD card data collected using the OS commands  
 public String cid; // Card ID (contains all of the following data)  
 public String manfid; // Manufacturer ID  
 public String oemid; // OEM/Application ID  
 public String sdName; // Product name  
 public String hwrev; // Product revision  
 public String fwrev; // Product revision  
 public String sdSerial; // Serial number  
 public String date; // Manufacture Date Code  
 //Other data collected using the OS commands  
 public double bogoMips;  
 public String hardware; // Raspberry PI model  
 public String revision; // Raspberry PI revision  
 public String mac1; // ethernet MAC address  
 public String mac2; // Wi-Fi MAC address  
 public String cpuSerial;  
 /\*  
 \* Private attributes needed for collection purposes.  
 \* Because these are unique instances provided by Java, they are defined as attributes of the  
 \* class so we only have to get them once.  
 \*/  
 protected final OperatingSystemMXBean os;  
 protected final Runtime r;  
 */\*\*  
 \* Construct a collector and collect data. The collect method can be called again later to update  
 \* the given instance with new values.  
 \*/* public Device() {  
 this.os = ManagementFactory.*getPlatformMXBean*(OperatingSystemMXBean.class);  
 this.r = Runtime.*getRuntime*();  
 }  
 */\*\*  
 \* Updates the fields of the class with new measurements.  
 \*/* public void collect() {  
 //Collect data using Java OperatingSystemMXBean class  
 this.architecture = os.getArch();  
 this.availableProcessors = os.getAvailableProcessors();  
 this.committedVirtualMemorySize = os.getCommittedVirtualMemorySize();  
 this.freePhysicalMemorySize = os.getFreePhysicalMemorySize();  
 this.freeSwapSpaceSize = os.getFreeSwapSpaceSize();  
 this.osName = os.getName();  
 this.processCpuLoad = os.getProcessCpuLoad();  
 this.processCpuTime = os.getProcessCpuTime();  
 this.systemCpuLoad = os.getSystemCpuLoad();  
 this.systemLoadAverage = os.getSystemLoadAverage();  
 this.totalPhysicalMemorySize = os.getTotalPhysicalMemorySize();  
 this.totalSwapSpaceSize = os.getTotalSwapSpaceSize();  
 this.osVersion = os.getVersion();  
 /\*  
 \* When retrieving values using the OS terminal, the output is given as a String so we first  
 \* need to select the part containing the value we are interested in (split) and then convert it  
 \* into the right type (parse)  
 \*/  
 //Collect data using Raspberry PI vcgencmd command  
 this.clock\_arm = Double.*parseDouble*(run("vcgencmd measure\_clock arm").split("=")[1]);  
 this.clock\_h264 = Double.*parseDouble*(run("vcgencmd measure\_clock h264").split("=")[1]);  
 this.clock\_isp = Double.*parseDouble*(run("vcgencmd measure\_clock isp").split("=")[1]);  
 this.clock\_v3d = Double.*parseDouble*(run("vcgencmd measure\_clock v3d").split("=")[1]);  
 this.clock\_uart = Double.*parseDouble*(run("vcgencmd measure\_clock uart").split("=")[1]);  
 this.clock\_pwm = Double.*parseDouble*(run("vcgencmd measure\_clock pwm").split("=")[1]);  
 this.clock\_emmc = Double.*parseDouble*(run("vcgencmd measure\_clock emmc").split("=")[1]);  
 this.clock\_pixel = Double.*parseDouble*(run("vcgencmd measure\_clock pixel").split("=")[1]);  
 this.clock\_vec = Double.*parseDouble*(run("vcgencmd measure\_clock vec").split("=")[1]);  
 this.clock\_hdmi = Double.*parseDouble*(run("vcgencmd measure\_clock hdmi").split("=")[1]);  
 this.clock\_dpi = Double.*parseDouble*(run("vcgencmd measure\_clock dpi").split("=")[1]);  
 this.temperature = Double.*parseDouble*(run("vcgencmd measure\_temp").split("[=']")[1]);  
 this.volts\_core = Double.*parseDouble*(run("vcgencmd measure\_volts core").split("[=V]")[1]);  
  
 this.volts\_sdram\_c = Double.*parseDouble*(  
 run("vcgencmd measure\_volts sdram\_c").split("[=V]")[1]  
 );  
 this.volts\_sdram\_i = Double.*parseDouble*(  
 run("vcgencmd measure\_volts sdram\_i").split("[=V]")[1]  
 );  
 this.volts\_sdram\_p = Double.*parseDouble*(  
 run("vcgencmd measure\_volts sdram\_p").split("[=V]")[1]  
 );  
 this.memorySize\_arm = Double.*parseDouble*(  
 run("vcgencmd get\_mem arm").split("[=M]")[1]  
 );  
 this.memorySize\_gpu = Double.*parseDouble*(  
 run("vcgencmd get\_mem gpu").split("[=M]")[1]  
 );  
 this.isCodecEnabled\_H264 = run("vcgencmd codec\_enabled H264")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_MPG2 = run("vcgencmd codec\_enabled MPG2")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_WVC1 = run("vcgencmd codec\_enabled WVC1")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_MPG4 = run("vcgencmd codec\_enabled MPG4")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_MJPG = run("vcgencmd codec\_enabled MJPG")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_WMV9 = run("vcgencmd codec\_enabled WMV9")  
 .split("=")[1].equals("enabled");  
 this.firmwareVersion = run("vcgencmd version").split(" ")[8];  
 /\*  
 \* Collect data using OS commands  
 \*/  
 this.cid = run("cat /sys/block/mmcblk0/device/cid");  
 this.manfid = run("cat /sys/block/mmcblk0/device/manfid").split("x")[1];  
 this.oemid = run("cat /sys/block/mmcblk0/device/oemid").split("x")[1];  
 this.sdName = run("cat /sys/block/mmcblk0/device/name");  
 this.hwrev = run("cat /sys/block/mmcblk0/device/hwrev").split("x")[1];  
 this.fwrev = run("cat /sys/block/mmcblk0/device/fwrev").split("x")[1];  
 this.sdSerial = run("cat /sys/block/mmcblk0/device/serial").split("x")[1];  
 this.date = run("cat /sys/block/mmcblk0/device/date");  
 this.hardware = run("grep Hardware /proc/cpuinfo").split(": ")[1];  
 this.revision = run("grep Revision /proc/cpuinfo").split(": ")[1];  
 this.cpuSerial = run("grep Serial /proc/cpuinfo").split(": ")[1];  
 this.mac1 = run("cat /sys/class/net/eth0/address");  
 this.mac2 = run("cat /sys/class/net/wlan0/address");  
 this.bogoMips = Double.*parseDouble*(run("grep BogoMIPS /proc/cpuinfo").split("[ \n]")[1]);  
 }  
  
 @Override  
 public String toString() {  
 Field[] fields = this.getClass().getFields();  
 String s = this.getClass().getName() + " Object {\n";  
 for (Field f : fields) {  
 s += "\t";  
 try {  
 s += f.getName() + ": " + f.get(this) + "\n";  
 } catch (IllegalAccessException ex) {  
 Logger.*getLogger*(Device.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 }  
 s += "}";  
 return s;  
 }  
 */\*\*  
 \* Run an OS command.  
 \** ***@param*** *command The command to execute in the Runtime  
 \** ***@return*** *The output of the command as a String  
 \*/* private String run(String command) {  
 String result = "";  
 try {  
 boolean isFirstLine = true;  
 String line;  
 Process p = this.r.exec(command);  
 BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));  
 while ((line = br.readLine()) != null) {  
 // If there are more than one line to output we add a new line to the output String  
 if (!isFirstLine) {  
 result = result.concat("\n");  
 }  
 result = result.concat(line);  
 isFirstLine = false;  
 }  
 p.waitFor();  
 p.destroy();  
 } catch (IOException | InterruptedException ex) {  
 Logger.*getLogger*(Device.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 return result;  
 }  
}

### Formatter.java

package deviceRecognition.formatter;  
  
import deviceRecognition.Point;  
import deviceRecognition.devices.Device;  
import java.lang.reflect.Field;  
import java.util.ArrayList;  
*/\*\*  
 \* The Formatter class is used to convert the data retrieved by a Device in a format that can be  
 \* used by the linear regression. This abstract class provides the basic methods to format the data  
 \* and subclasses represents the different set of values to be used  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public abstract class Formatter {  
 // Lower bound for the exponent  
 protected int expFrom;  
 // Upper bound for exponent  
 protected int expTo;  
 // Name of fields for the constant data to use in the regression  
 protected String[] namesCons;  
 // Name of fields for the variable data to use in the regression  
 protected String[] namesVar;  
 // Weight given to constant data for the linear regression  
 protected double weightCons;  
 // Weight given to variable data for the linear regression  
 protected double weightVar;  
 */\*\*  
 \* Construct a Formatter with the given parameters  
 \*  
 \** ***@param*** *cWeight weight for constant data  
 \** ***@param*** *vWeight weight for variable data  
 \** ***@param*** *expFrom lower bound for exponents. If this value is equal to the upper bound, the  
 \* original data values will be used  
 \** ***@param*** *expTo upper bounds for exponents. If the value is equal to the lower bound, the  
 \* original data values will be used  
 \*/* public Formatter(double cWeight, double vWeight, int expFrom, int expTo) {  
 this.weightCons = cWeight;  
 this.weightVar = vWeight;  
 this.expFrom = expFrom;  
 this.expTo = expTo;  
 }  
 */\*\*  
 \* Format data to create Point objects from them.  
 \*  
 \** ***@param*** *dev The device containing the data to format.  
 \** ***@param*** *noVar If set to true, the Formatter will ignore variable data.  
 \** ***@param*** *mixed If set to true, the Point objects will mix constant and variable coordinates. If  
 \* set to false, the Point objects will contain two constant or two variable coordinates.  
 \** ***@return*** *array of Point objects.  
 \** ***@throws*** *java.lang.NoSuchFieldException  
 \** ***@throws*** *java.lang.IllegalAccessException  
 \*/* public Point[] formatData(Device dev, boolean noVar, boolean mixed)  
 throws NoSuchFieldException, IllegalAccessException {  
 dev.collect();  
 // Creating list for constants  
 ArrayList<String> constantNames = new ArrayList<>();  
 ArrayList<Double> constantValues = new ArrayList<>();  
 for (String name : this.namesCons) {  
 Field f = dev.getClass().getField(name);  
 if (f.getType().getSimpleName().equals("String")) {  
 int[] info = this.getStringInfo(f);  
 double[] subValues = this.formatString((String) f.get(dev), info[0], info[1]);  
 for (double val : subValues) {  
 constantNames.add(name);  
 if (this.expFrom != this.expTo) {  
 val = modifyExponentInRange(val, this.expFrom, this.expTo);  
 }  
 constantValues.add(val);  
 }  
 } else {  
 constantNames.add(name);  
 double val = this.formatField(dev, f);  
 if (this.expFrom != this.expTo) {  
 val = modifyExponentInRange(val, this.expFrom, this.expTo);  
 }  
 constantValues.add(val);  
 }  
 }  
 if (noVar) {  
 int pointsNbr = constantNames.size() / 2;  
 Point[] points = new Point[pointsNbr];  
 for (int i = 0; i < pointsNbr; i++) {  
 points[i] = new Point(  
 this.weightCons,  
 constantNames.get(i \* 2),  
 constantValues.get(i \* 2),  
 constantNames.get(i \* 2 + 1),  
 constantValues.get(i \* 2 + 1)  
 );  
 }  
 return points;  
 } else {  
 //Creating list for variables  
 ArrayList<String> varNames = new ArrayList<>();  
 ArrayList<Double> varValues = new ArrayList<>();  
 for (String name : this.namesVar) {  
 Field f = dev.getClass().getField(name);  
 if (f.getType().getSimpleName().equals("String")) {  
 int[] info = this.getStringInfo(f);  
 double[] subValues = this.formatString((String) f.get(dev), info[0], info[1]);  
 for (double val : subValues) {  
 varNames.add(name);  
 if (this.expFrom != this.expTo) {  
 val = modifyExponentInRange(val, this.expFrom, this.expTo);  
 }  
 varValues.add(val);  
 }  
 } else {  
 varNames.add(name);  
 double val = this.formatField(dev, f);  
 if (this.expFrom != this.expTo) {  
 val = modifyExponentInRange(val, this.expFrom, this.expTo);  
 }  
 varValues.add(val);  
 }  
 }  
 if (mixed) {  
 int pointsNbr = (constantNames.size() + varNames.size()) / 2;  
 int limit = Math.*min*(constantNames.size(), varNames.size());  
 Point[] points = new Point[pointsNbr];  
 for (int i = 0; i < pointsNbr; i++) {  
 if (i < limit) {  
 points[i] = new Point(  
 this.weightVar,  
 constantNames.get(i),  
 constantValues.get(i),  
 varNames.get(i),  
 varValues.get(i)  
 );  
 } else if (varNames.size() == limit) {  
 points[i] = new Point(  
 this.weightCons,  
 constantNames.get(i \* 2 - limit),  
 constantValues.get(i \* 2 - limit),  
 constantNames.get(i \* 2 - limit + 1),  
 constantValues.get(i \* 2 - limit + 1)  
 );  
 } else {  
 points[i] = new Point(  
 this.weightVar,  
 varNames.get(i \* 2 - limit),  
 varValues.get(i \* 2 - limit),  
 varNames.get(i \* 2 - limit + 1),  
 varValues.get(i \* 2 - limit + 1)  
 );  
 }  
 }  
 return points;  
 } else {  
 int constantPointsNbr = constantNames.size() / 2;  
 int variablePointsNbr = varNames.size() / 2;  
 int pointsNbr = constantPointsNbr + variablePointsNbr;  
 Point[] points = new Point[pointsNbr];  
 for (int i = 0; i < constantPointsNbr; i++) {  
 points[i] = new Point(  
 this.weightCons,  
 constantNames.get(i \* 2),  
 constantValues.get(i \* 2),  
 constantNames.get(i \* 2 + 1),  
 constantValues.get(i \* 2 + 1)  
 );  
 }  
 for (int i = 0; i < variablePointsNbr; i++) {  
 points[i + constantPointsNbr] = new Point(  
 this.weightVar,  
 varNames.get(i \* 2),  
 varValues.get(i \* 2),  
 varNames.get(i \* 2 + 1),  
 varValues.get(i \* 2 + 1)  
 );  
 }  
 return points;  
 }  
 }  
 }  
  
 */\*\*  
 \* Return the content of the Field casted as a double  
 \*  
 \** ***@param*** *dev The Device owning the Field  
 \** ***@param*** *f The Field whose value needs to be casted as double  
 \** ***@return*** *The double value representing the content of the Field  
 \** ***@throws*** *IllegalAccessException  
 \*/* private double formatField(Device dev, Field f) throws IllegalAccessException {  
 switch (f.getType().getSimpleName()) {  
 case "int":  
 return (int) f.get(dev);  
 case "long":  
 return (long) f.get(dev);  
 case "boolean":  
 return (boolean) f.get(dev) ? 1 : 0;  
 default:  
 return (double) f.get(dev);  
 }  
 }  
 */\*\*  
 \* Split Strings that are too long to be converted into smaller substrings  
 \* that will be converted.  
 \*  
 \** ***@param*** *data The String to be converted  
 \** ***@param*** *radix The radix to use during the conversion. Must be 36 for alphanumerical Strings  
 \* (A-Za-z0-9) or 16 for hexadecimal (0-9a-f)  
 \** ***@param*** *size The size of the substrings that will be extracted from the given String. The last  
 \* substring will be smaller if there are not enoughcharacters.  
 \** ***@return*** *An array of double which are the values gotten from the converted substrings.  
 \*/* private double[] formatString(String data, int radix, int size) {  
 data = data.replaceAll("[^A-Za-z0-9]", "");  
 if (data.length() > size) {  
 int nb = (data.length() - 1) / size + 1;  
 double[] l = new double[nb];  
 for (int i = 0; i < nb; i++) {  
 int limit = (i + 1) \* size < data.length() ? (i + 1) \* size : data.length();  
 l[i] = Long.*parseLong*(data.substring(i \* size, limit), radix);  
 }  
 return l;  
 } else {  
 return new double[]{Long.*parseLong*(data, radix)};  
 }  
 }  
 */\*\*  
 \* Get the information needed to convert the String of a given Field into a double. These  
 \* information are the radix and size of substrings.  
 \*  
 \** ***@param*** *f The Field to get information from.  
 \** ***@return*** *An int array with two values: the radix to use as index 0 and the size of substrings to  
 \* use as 1.  
 \*/* private int[] getStringInfo(Field f) {  
 int[] info = new int[2]; // [Radix, Substrings size]  
 switch (f.getName()) {  
// Alphanum Strings  
 case "hardware":  
 info[0] = 36;  
 info[1] = 4;  
 break;  
 case "architecture":  
 case "osName":  
 case "osVersion":  
 case "sdName":  
 info[0] = 36;  
 info[1] = 6;  
 break;  
// Hexadecimal Strings  
 case "sdSerial":  
 case "cpuSerial":  
 info[0] = 16;  
 info[1] = 4;  
 break;  
 case "mac1":  
 case "mac2":  
 case "revision":  
 info[0] = 16;  
 info[1] = 6;  
 break;  
 case "firmwareVersion":  
 info[0] = 16;  
 info[1] = 7;  
 break;  
 case "cid":  
 case "manfid":  
 case "oemid":  
 case "hwrev":  
 case "fwrev":  
 info[0] = 16;  
 info[1] = 8;  
 break;  
// Decimal Strings  
 case "date":  
 info[0] = 11;  
 info[1] = 10;  
 break;  
 default:  
 info[0] = 0;  
 info[1] = 0;  
 }  
 return info;  
 }  
 */\*\*  
 \* Modify a double exponent in order to be in the given range  
 \** ***@param*** *val The double value to modify  
 \** ***@param*** *from The lower bound for the exponent  
 \** ***@param*** *to The upper bound for the exponent  
 \** ***@return*** *The value with its exponent changed  
 \*/* private double modifyExponentInRange(double val, int from, int to) {  
 if (val != 0) {  
 int exp = (int) Math.*floor*(Math.*log10*(Math.*abs*(val)));  
 int newExp = Math.*floorMod*(exp - from, to - from) + from;  
 return val \* Math.*pow*(10, newExp - exp);  
 } else {  
 return val;  
 }  
 }  
}

### Formatter50.java

package deviceRecognition.formatter;  
  
*/\*\*  
 \* The Formatter50 class is used to convert the data retrieved by a Device in a format that can be  
 \* used by the linear regression. It formats data using a set of values with a ratio of 50/50  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Formatter50 extends Formatter {  
 */\*\*  
 \* Construct a Formatter with the given parameters  
 \** ***@param*** *cWeight weight for constant data  
 \** ***@param*** *vWeight weight for variable data  
 \** ***@param*** *expFrom lower bound for exponents. If this value is equal to the upper bound, the  
 \* original data values will be used  
 \** ***@param*** *expTo upper bounds for exponents. If the value is equal to the lower bound, the  
 \* original data values will be used  
 \*/* public Formatter50(  
 double cWeight, double vWeight, int expFrom, int expTo  
 ) {  
 super(cWeight, vWeight, expFrom, expTo);  
 this.namesCons = new String[]{  
 "memorySize\_arm", "memorySize\_gpu", "cpuSerial",  
 "firmwareVersion", "cid", "mac1", "mac2", "hardware"  
 };  
 this.namesVar = new String[]{  
 "freePhysicalMemorySize", "freeSwapSpaceSize",  
 "systemLoadAverage", "temperature", "volts\_core",  
 "volts\_sdram\_c", "volts\_sdram\_i", "volts\_sdram\_p",  
 "clock\_arm", "clock\_core", "clock\_h264", "clock\_isp",  
 "clock\_v3d", "clock\_uart", "clock\_pwm", "clock\_emmc",  
 "clock\_pixel", "clock\_vec", "clock\_hdmi", "clock\_dpi",  
 "processCpuLoad", "systemCpuLoad"  
 };  
 }  
}

### Formatter60.java

package deviceRecognition.formatter;  
*/\*\*  
 \* The Formatter60 class is used to convert the data retrieved by a Device in a format that can be  
 \* used by the linear regression. It formats data using a set of values with a ratio of 60/40  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Formatter60 extends Formatter {  
 */\*\*  
 \* Construct a Formatter with the given parameters  
 \** ***@param*** *cWeight weight for constant data  
 \** ***@param*** *vWeight weight for variable data  
 \** ***@param*** *expFrom lower bound for exponents. If this value is equal to the upper bound, the  
 \* original data values will be used  
 \** ***@param*** *expTo upper bounds for exponents. If the value is equal to the lower bound, the  
 \* original data values will be used  
 \*/* public Formatter60(  
 double cWeight, double vWeight, int expFrom, int expTo  
 ) {  
 super(cWeight, vWeight, expFrom, expTo);  
 this.namesCons = new String[]{  
 "memorySize\_arm", "memorySize\_gpu", "cpuSerial",  
 "firmwareVersion", "cid", "mac1", "mac2"  
 };  
 this.namesVar = new String[]{  
 "freePhysicalMemorySize", "freeSwapSpaceSize",  
 "systemLoadAverage", "temperature", "volts\_core",  
 "volts\_sdram\_c", "volts\_sdram\_i", "volts\_sdram\_p",  
 "clock\_arm", "clock\_uart", "processCpuLoad",  
 "systemCpuLoad"  
 };  
 }  
}

### Formatter70.java

package deviceRecognition.formatter;  
  
*/\*\*  
 \* The Formatter70 class is used to convert the data retrieved by a Device in a format that can be  
 \* used by the linear regression. It formats data using a set of values with a ratio of 70/30  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Formatter70 extends Formatter {  
 */\*\*  
 \* Construct a Formatter with the given parameters  
 \** ***@param*** *cWeight weight for constant data  
 \** ***@param*** *vWeight weight for variable data  
 \** ***@param*** *expFrom lower bound for exponents. If this value is equal to the upper bound, the  
 \* original data values will be used  
 \** ***@param*** *expTo upper bounds for exponents. If the value is equal to the lower bound, the  
 \* original data values will be used  
 \*/* public Formatter70(double cWeight, double vWeight, int expFrom, int expTo) {  
 super(cWeight, vWeight, expFrom, expTo);  
 this.namesCons = new String[]{  
 "memorySize\_arm", "memorySize\_gpu", "cpuSerial",  
 "firmwareVersion", "cid", "mac1", "mac2", "hardware"  
 };  
 this.namesVar = new String[]{  
 "freePhysicalMemorySize", "freeSwapSpaceSize",  
 "systemLoadAverage", "temperature", "volts\_core",  
 "volts\_sdram\_c", "clock\_arm", "clock\_uart"  
 };  
 }  
}

### Formatter80.java

package deviceRecognition.formatter;  
  
*/\*\*  
 \* The Formatter80 class is used to convert the data retrieved by a Device in a format that can be  
 \* used by the linear regression. It formats data using a set of values with a ratio of 80/20  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Formatter80 extends Formatter {  
 */\*\*  
 \* Construct a Formatter with the given parameters  
 \** ***@param*** *cWeight weight for constant data  
 \** ***@param*** *vWeight weight for variable data  
 \** ***@param*** *expFrom lower bound for exponents. If this value is equal to the upper bound, the  
 \* original data values will be used  
 \** ***@param*** *expTo upper bounds for exponents. If the value is equal to the lower bound, the  
 \* original data values will be used  
 \*/* public Formatter80(  
 double cWeight, double vWeight, int expFrom, int expTo  
 ) {  
 super(cWeight, vWeight, expFrom, expTo);  
 this.namesCons = new String[]{  
 "memorySize\_arm", "memorySize\_gpu", "cpuSerial",  
 "firmwareVersion", "cid", "mac1", "mac2", "hardware"  
 };  
 this.namesVar = new String[]{  
 "freePhysicalMemorySize", "freeSwapSpaceSize",  
 "volts\_core", "clock\_arm"  
 };  
 }  
}

### Device.java

package deviceRecognition.devices;  
import com.sun.management.OperatingSystemMXBean;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.lang.management.ManagementFactory;  
import java.lang.reflect.Field;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
*/\*\*  
 \* The Device class is used to collect data on the Raspberry PI. All data are made public so they can  
 \* be called by the Formatter classes using reflection.  
 \*  
 \** ***@author*** *Jonathan Branstett  
 \*/*public class Device {  
 // Data collected using Java OperatingSystemMXBean class  
 public String architecture;  
 public int availableProcessors;  
 public long committedVirtualMemorySize;  
 public long freePhysicalMemorySize;  
 public long freeSwapSpaceSize;  
 public String osName;  
 public String osVersion;  
 public double processCpuLoad; //recent CPU usage for the JVM process  
 public long processCpuTime; //CPU time used by the JVM process  
 public double systemCpuLoad; //recent CPU usage in the system  
 public double systemLoadAverage; //System load average for the last minute  
 public long totalPhysicalMemorySize;  
 public long totalSwapSpaceSize;  
 //Data collected using the Raspberry PI vcgencmd command  
 public double clock\_arm;  
 public double clock\_core;  
 public double clock\_h264;  
 public double clock\_isp;  
 public double clock\_v3d;  
 public double clock\_uart;  
 public double clock\_pwm;  
 public double clock\_emmc;  
 public double clock\_pixel;  
 public double clock\_vec;  
 public double clock\_hdmi;  
 public double clock\_dpi;  
 public double volts\_core;  
 public double volts\_sdram\_c;  
 public double volts\_sdram\_i;  
 public double volts\_sdram\_p;  
 public double temperature;  
 public boolean isCodecEnabled\_H264;  
 public boolean isCodecEnabled\_MPG2;  
 public boolean isCodecEnabled\_WVC1;  
 public boolean isCodecEnabled\_MPG4;  
 public boolean isCodecEnabled\_MJPG;  
 public boolean isCodecEnabled\_WMV9;  
 public double memorySize\_arm;  
 public double memorySize\_gpu;  
 public String firmwareVersion;  
 // SD card data collected using the OS commands  
 public String cid; // Card ID (contains all of the following data)  
 public String manfid; // Manufacturer ID  
 public String oemid; // OEM/Application ID  
 public String sdName; // Product name  
 public String hwrev; // Product revision  
 public String fwrev; // Product revision  
 public String sdSerial; // Serial number  
 public String date; // Manufacture Date Code  
 //Other data collected using the OS commands  
 public double bogoMips;  
 public String hardware; // Raspberry PI model  
 public String revision; // Raspberry PI revision  
 public String mac1; // ethernet MAC address  
 public String mac2; // Wi-Fi MAC address  
 public String cpuSerial;  
 /\*  
 \* Private attributes needed for collection purposes.  
 \* Because these are unique instances provided by Java, they are defined as attributes of the  
 \* class so we only have to get them once.  
 \*/  
 protected final OperatingSystemMXBean os;  
 protected final Runtime r;  
 */\*\*  
 \* Construct a collector and collect data. The collect method can be called again later to update  
 \* the given instance with new values.  
 \*/* public Device() {  
 this.os = ManagementFactory.*getPlatformMXBean*(OperatingSystemMXBean.class);  
 this.r = Runtime.*getRuntime*();  
 }  
 */\*\*  
 \* Updates the fields of the class with new measurements.  
 \*/* public void collect() {  
 //Collect data using Java OperatingSystemMXBean class  
 this.architecture = os.getArch();  
 this.availableProcessors = os.getAvailableProcessors();  
 this.committedVirtualMemorySize = os.getCommittedVirtualMemorySize();  
 this.freePhysicalMemorySize = os.getFreePhysicalMemorySize();  
 this.freeSwapSpaceSize = os.getFreeSwapSpaceSize();  
 this.osName = os.getName();  
 this.processCpuLoad = os.getProcessCpuLoad();  
 this.processCpuTime = os.getProcessCpuTime();  
 this.systemCpuLoad = os.getSystemCpuLoad();  
 this.systemLoadAverage = os.getSystemLoadAverage();  
 this.totalPhysicalMemorySize = os.getTotalPhysicalMemorySize();  
 this.totalSwapSpaceSize = os.getTotalSwapSpaceSize();  
 this.osVersion = os.getVersion();  
 /\*  
 \* When retrieving values using the OS terminal, the output is given as a String so we first  
 \* need to select the part containing the value we are interested in (split) and then convert it  
 \* into the right type (parse)  
 \*/  
 //Collect data using Raspberry PI vcgencmd command  
 this.clock\_arm = Double.*parseDouble*(run("vcgencmd measure\_clock arm").split("=")[1]);  
 this.clock\_h264 = Double.*parseDouble*(run("vcgencmd measure\_clock h264").split("=")[1]);  
 this.clock\_isp = Double.*parseDouble*(run("vcgencmd measure\_clock isp").split("=")[1]);  
 this.clock\_v3d = Double.*parseDouble*(run("vcgencmd measure\_clock v3d").split("=")[1]);  
 this.clock\_uart = Double.*parseDouble*(run("vcgencmd measure\_clock uart").split("=")[1]);  
 this.clock\_pwm = Double.*parseDouble*(run("vcgencmd measure\_clock pwm").split("=")[1]);  
 this.clock\_emmc = Double.*parseDouble*(run("vcgencmd measure\_clock emmc").split("=")[1]);  
 this.clock\_pixel = Double.*parseDouble*(run("vcgencmd measure\_clock pixel").split("=")[1]);  
 this.clock\_vec = Double.*parseDouble*(run("vcgencmd measure\_clock vec").split("=")[1]);  
 this.clock\_hdmi = Double.*parseDouble*(run("vcgencmd measure\_clock hdmi").split("=")[1]);  
 this.clock\_dpi = Double.*parseDouble*(run("vcgencmd measure\_clock dpi").split("=")[1]);  
 this.temperature = Double.*parseDouble*(run("vcgencmd measure\_temp").split("[=']")[1]);  
 this.volts\_core = Double.*parseDouble*(run("vcgencmd measure\_volts core").split("[=V]")[1]);  
  
 this.volts\_sdram\_c = Double.*parseDouble*(  
 run("vcgencmd measure\_volts sdram\_c").split("[=V]")[1]  
 );  
 this.volts\_sdram\_i = Double.*parseDouble*(  
 run("vcgencmd measure\_volts sdram\_i").split("[=V]")[1]  
 );  
 this.volts\_sdram\_p = Double.*parseDouble*(  
 run("vcgencmd measure\_volts sdram\_p").split("[=V]")[1]  
 );  
 this.memorySize\_arm = Double.*parseDouble*(  
 run("vcgencmd get\_mem arm").split("[=M]")[1]  
 );  
 this.memorySize\_gpu = Double.*parseDouble*(  
 run("vcgencmd get\_mem gpu").split("[=M]")[1]  
 );  
 this.isCodecEnabled\_H264 = run("vcgencmd codec\_enabled H264")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_MPG2 = run("vcgencmd codec\_enabled MPG2")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_WVC1 = run("vcgencmd codec\_enabled WVC1")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_MPG4 = run("vcgencmd codec\_enabled MPG4")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_MJPG = run("vcgencmd codec\_enabled MJPG")  
 .split("=")[1].equals("enabled");  
 this.isCodecEnabled\_WMV9 = run("vcgencmd codec\_enabled WMV9")  
 .split("=")[1].equals("enabled");  
 this.firmwareVersion = run("vcgencmd version").split(" ")[8];  
 /\*  
 \* Collect data using OS commands  
 \*/  
 this.cid = run("cat /sys/block/mmcblk0/device/cid");  
 this.manfid = run("cat /sys/block/mmcblk0/device/manfid").split("x")[1];  
 this.oemid = run("cat /sys/block/mmcblk0/device/oemid").split("x")[1];  
 this.sdName = run("cat /sys/block/mmcblk0/device/name");  
 this.hwrev = run("cat /sys/block/mmcblk0/device/hwrev").split("x")[1];  
 this.fwrev = run("cat /sys/block/mmcblk0/device/fwrev").split("x")[1];  
 this.sdSerial = run("cat /sys/block/mmcblk0/device/serial").split("x")[1];  
 this.date = run("cat /sys/block/mmcblk0/device/date");  
 this.hardware = run("grep Hardware /proc/cpuinfo").split(": ")[1];  
 this.revision = run("grep Revision /proc/cpuinfo").split(": ")[1];  
 this.cpuSerial = run("grep Serial /proc/cpuinfo").split(": ")[1];  
 this.mac1 = run("cat /sys/class/net/eth0/address");  
 this.mac2 = run("cat /sys/class/net/wlan0/address");  
 this.bogoMips = Double.*parseDouble*(run("grep BogoMIPS /proc/cpuinfo").split("[ \n]")[1]);  
 }  
  
 @Override  
 public String toString() {  
 Field[] fields = this.getClass().getFields();  
 String s = this.getClass().getName() + " Object {\n";  
 for (Field f : fields) {  
 s += "\t";  
 try {  
 s += f.getName() + ": " + f.get(this) + "\n";  
 } catch (IllegalAccessException ex) {  
 Logger.*getLogger*(Device.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 }  
 s += "}";  
 return s;  
 }  
 */\*\*  
 \* Run an OS command.  
 \** ***@param*** *command The command to execute in the Runtime  
 \** ***@return*** *The output of the command as a String  
 \*/* private String run(String command) {  
 String result = "";  
 try {  
 boolean isFirstLine = true;  
 String line;  
 Process p = this.r.exec(command);  
 BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));  
 while ((line = br.readLine()) != null) {  
 // If there are more than one line to output we add a new line to the output String  
 if (!isFirstLine) {  
 result = result.concat("\n");  
 }  
 result = result.concat(line);  
 isFirstLine = false;  
 }  
 p.waitFor();  
 p.destroy();  
 } catch (IOException | InterruptedException ex) {  
 Logger.*getLogger*(Device.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 return result;  
 }  
}

## Python Code

import numpy as np

from sklearn import linear\_model, utils, cluster

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.linear\_model import Ridge

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make\_pipeline

from joblib import dump, load

def \_\_iter\_\_(self): return 0

print("Libraries Loaded Successfully")

# The Data class provides an organised structure to store the data and the analysis performed on said data.

# This class allows easy access to the coefficients and the polynomial regression model for each dataset.

class Data:

def \_\_init\_\_(self, df):

self.df = formatDF(df)

self.combinedDF = combineDF(self.df)

self.weightedDF = weightedDF(self.combinedDF)

def getModel(self, degree):

return calculateModel(self.weightedDF, degree)

def getCoefs(self, degree):

return self.getModel(degree).steps[1][1].coef\_[1:degree+1]

# This method performs preliminary cleaning on the dataset.

def formatDF(df):

df = df.drop\_duplicates()

df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

return df

# This method formats the dataset from having multiple columns into a dataset that has just two columns.

def combineDF(df):

dfCombined = pd.DataFrame()

for i in range(int(df.shape[1]/2)):

col1 = df.iloc[:,i\*2]

col2 = df.iloc[:,(i\*2)+1]

dfCombined = dfCombined.append(pd.DataFrame.from\_dict({'0':col1, '1':col2}), ignore\_index=True)

dfCombined.dropna(inplace=True)

return dfCombined

# This dataset weights the datapoints depending on their uniqueness.

def weightedDF(df):

df['0'] = uniqueModelX = utils.class\_weight.compute\_class\_weight('balanced', df['0'], df['0'])

df['1'] = uniqueModelY = utils.class\_weight.compute\_class\_weight('balanced', df['1'], df['1'])

df.drop\_duplicates(inplace=True)

df.reset\_index(inplace=True, drop=True)

return df

# This method calculates the polynomial regression model for each dataset using Ridge regression.

def calculateModel(data, degree):

model = make\_pipeline(PolynomialFeatures(degree=degree), Ridge())

model.fit(data['0'].values.reshape(-1,1), data['1'])

return model

#This method loads in the data file generated by the Java code.

def loadData():

global dataList

dataList = {}

dataList['Pi'] = Data(pd.read\_csv('Pi.csv'))

print('CSV file loaded')

'''

From this point the data analysis starts.

1. Data is loaded in

2. The coefficients are calculated and printed to the console

3. The classification model is loaded in

4. The identity of the device is predicted and printed

'''

loadData()

coefs = dataList.get("Pi").getCoefs(2)

piDF = pd.DataFrame(columns=['coef1', 'coef2'], data=list([coefs]))

print(piDF.head())

clf = load('AuthenticationModel.joblib')

y\_pred = clf.predict(piDF)

if y\_pred == 1:

classification = "CityPi"

else:

classification = "HomePi"

print("Classification: " + classification)

## Formatter Variables

### Formatter 50

Constant:  
"memorySize\_arm", "memorySize\_gpu", "cpuSerial",

"firmwareVersion", "cid", "mac1", "mac2", "hardware"

Variable:

"freePhysicalMemorySize", "freeSwapSpaceSize",

"systemLoadAverage", "temperature", "volts\_core",

"volts\_sdram\_c", "volts\_sdram\_i", "volts\_sdram\_p",

"clock\_arm", "clock\_core", "clock\_h264", "clock\_isp",

"clock\_v3d", "clock\_uart", "clock\_pwm", "clock\_emmc",

"clock\_pixel", "clock\_vec", "clock\_hdmi", "clock\_dpi",

"processCpuLoad", "systemCpuLoad"

### Formatter 60

Constant:

"memorySize\_arm", "memorySize\_gpu", "cpuSerial",

"firmwareVersion", "cid", "mac1", "mac2"

Variable:

"freePhysicalMemorySize", "freeSwapSpaceSize",

"systemLoadAverage", "temperature", "volts\_core",

"volts\_sdram\_c", "volts\_sdram\_i", "volts\_sdram\_p",

"clock\_arm", "clock\_uart", "processCpuLoad",

"systemCpuLoad"

### Formatter 70

Constant:

"memorySize\_arm", "memorySize\_gpu", "cpuSerial",

"firmwareVersion", "cid", "mac1", "mac2", "hardware"

Variable:

"freePhysicalMemorySize", "freeSwapSpaceSize",

"systemLoadAverage", "temperature", "volts\_core",

"volts\_sdram\_c", "clock\_arm", "clock\_uart"

### Formatter 80

Constant:

"memorySize\_arm", "memorySize\_gpu", "cpuSerial",

"firmwareVersion", "cid", "mac1", "mac2", "hardware"

Variable:

"freePhysicalMemorySize", "freeSwapSpaceSize",

"volts\_core", "clock\_arm"

## Classification Metrics

The classifiers below were trained with 20 data points, using cross validation. The below data was calculated using the Weka metrics tool. (Eibe Frank, 2016)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Support Vector Machine | Logistic Regression | Naïve Bayes | Random Forest |
| Correctly Classified Instances | 90% | 100% | 100% | 95% |
| Incorrectly Classified Instances | 10% | 0% | 0% | 5% |
| Kappa statistic | 0.8 | 1 | 1 | 0.9 |
| Mean absolute error | 0.1 | 0 | 0.1011 | 0.136 |
| Root mean squared error | 0.3162 | 0.0001 | 0.1587 | 0.2213 |
| Relative absolute error | 20% | 0.0027% | 20.226% | 27.2% |
| Root relative squared error | 63.2456% | 0.0122% | 31.745% | 44.2538% |

### Support Vector Machine

#### Metrics

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
| 0 | 1.000 | 0.200 | 0.833 | 1.000 | 0.909 | 0.816 | 0.900 | 0.833 |
| 1 | 0.800 | 0.000 | 1.000 | 0.800 | 0.889 | 0.816 | 0.900 | 0.900 |
| Weighted Avg. | 0.900 | 0.100 | 0.917 | 0.900 | 0.899 | 0.816 | 0.900 | 0.867 |

#### Parameters

|  |  |
| --- | --- |
| C | 1.0 |
| class\_weight | None |
| dual | False |
| fit\_intercept | True |
| intercept\_scaling | 1 |
| max\_iter | 100 |
| multi\_class | multinomial |
| n\_jobs | None |
| penalty | l2 |
| random\_state | 0 |
| solver | lbfgs |
| tol | 0.0001 |
| verbose | 0 |
| warm\_start | False |

### Logistic Regression

#### Metrics

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
| 0 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 1 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Weighted Avg. | 1.000 | 0.00 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

#### Parameters

|  |  |
| --- | --- |
| bootstrap | True |
| class\_weight | None |
| criterion | gini |
| max\_depth | 2 |
| max\_features | auto |
| max\_leaf\_nodes | None |
| min\_impurity\_decrease | 0.0 |
| min\_impurity\_split | None |
| min\_samples\_leaf | 1 |
| min\_samples\_split | 2 |
| min\_weight\_fraction\_leaf | 0.0 |
| n\_estimators | 100 |
| n\_jobs | None |
| oob\_score | False |
| random\_state | 0 |
| Verbose | 0 |
| warm\_start | False |

### Naïve Bayes

#### Metrics

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
| 0 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 1 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Weighted Avg. | 1.000 | 0.00 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

#### Parameters

|  |  |
| --- | --- |
| C | 10000000000.0 |
| cache\_size | 200 |
| class\_weight | None |
| coef0 | 0.0 |
| decision\_function\_shape | ovr |
| degree | 1 |
| gamma | auto |
| kernel | poly |
| max\_iter | -1 |
| probability | False |
| random\_state | None |
| shrinking | True |
| tol | 0.001 |
| verbose | False |

### Random Forest

#### Metrics

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
| 0 | 1.000 | 0.100 | 0.909 | 1.000 | 0.952 | 0.905 | 0.995 | 0.991 |
| 1 | 0.900 | 0.000 | 1.000 | 0.900 | 0.947 | 0.905 | 0.995 | 0.991 |
| Weighted Avg. | 0.950 | 0.050 | 0.955 | 0.950 | 0.950 | 0.905 | 0.995 | 0.991 |

#### Parameters

|  |  |
| --- | --- |
| priors | None |
| var\_smoothing | 1e-09 |

### K-Means Clustering

#### Parameters

|  |  |
| --- | --- |
| algorithm: | auto |
| copy\_x: | True |
| init: | k-means++ |
| max\_iter: | 300 |
| n\_clusters: | 2 |
| n\_init: | 10 |
| n\_jobs: | None |
| random\_state: | None |
| precompute\_distances: | auto |
| tol: | 0.0001 |
| verbose: | 0 |

# Bibliography

Chen, E., 2011. *Choosing a Machine Learning Classifier.* [Online]   
Available at: https://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/  
[Accessed 10 September 2019].

Eibe Frank, M. A. H. a. I. H. W., 2016. *Data Mining: Practical Machine Learning Tools and Techniques.* [Online]   
Available at: https://www.cs.waikato.ac.nz/~ml/weka/index.html  
[Accessed 10 September 2019].

O'DWYER, M., 2018. *Internet Of Things 101 – IoT Device Authentication Explained.* [Online]   
Available at: https://blog.ipswitch.com/internet-of-things-101-iot-device-authentication-explained  
[Accessed 18 August 2019].

Rouse, M., 2018. *IoT devices (internet of things devices).* [Online]   
Available at: https://internetofthingsagenda.techtarget.com/definition/IoT-device  
[Accessed 18 August 2019].