UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Antti Möttönen

CREATING A TOOLBOX FOR IOT DEVICE BEHAVIOUR ANALYSIS USING DATA MINING

Master's Thesis
Degree Programme in Computer Science and Engineering
September 2019

Möttönen A. (2019) Creating a toolbox for IoT device behaviour analysis using data mining. University of Oulu, Degree Programme in Computer Science and Engineering. Master's thesis, 46 p.

ABSTRACT

Plenty of devices are connected to the Internet and the number is growing. A lot of data can be extracted from those devices. Using data mining approach that data can be transformed into valuable information.

This work analyses data sources and devices in the Internet of Things ecosystem developed by Arm ltd. The ecosystem includes Mbed OS operating system for embedded devices and Pelion Cloud for device management. Data sources available in the ecosystem are mapped and analysed. A toolbox is created for analysing the data with the goal of separating differently behaving devices into separate clusters. Methods used are machine learning based.

The system utilises events generated by Pelion and memory usage data gathered on devices. Combining the two data sources produces temporal data describing operations of each device. Using Hidden Markov Models that data is transformed into a similarity matrix describing similarity of devices behaviour. The matrix is then analysed using clustering methods with the purpose of separating devices into groups by behaviour. Dimensionality reduction methods are applied to data and the results are visualised.

The test dataset used in this work was small, only 10 devices. The results show some promise and warrant a follow-up study using a larger dataset to further improve the toolbox.

Keywords: Internet of things, data mining, machine learning, big data, device management

Möttönen A. (2019) Työkalun luonti IoT-laitteiden käyttäytymisen analysointiin datan rikastusmenetelmillä. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 46 s.

TIIVISTELMÄ

Internetiin on kytkettynä valtava määrä laitteita ja niitä kytketään jatkuvasti lisää. Kytketyistä laitteista voidaan kerätä suuri määrä dataa. Datan louhintamenetelmillä tuo data voidaan muuntaa arvokkaaksi tiedoksi.

Tässä työssä tutkitaan datalähteitä ja laitteita ohjelmistoyhtiö Arm ltd:n kehittämässä esineiden internetin ekosysteemissä. Ekosysteemiin kuuluu Mbed OS käyttöjärjestelmä sulautetuille laitteille ja Pelion Cloud palvelu laitteiden hallintaan. Ekosysteemissä saatavilla olevat datalähteet kartoitetaan ja analysoidaan. Työssä rakennetaan työkalu, jonka tarkoituksena on tunnistaa laitteet, joiden toiminta eroaa muista vastaavista laitteista. Käytetyt menetelmät ovat koneoppimispohjaisia.

Työkalu hyödyntää tapahtumia, jotka tallennetaan Pelioniin laitteiden elinkaaren aikana ja muistin käyttömääriä, jotka on kerätty laitteilta. Yhdistämällä nämä datalähteet syntyy aikajana, joka kuvaa laitteen toimintaa. Käyttämällä piilotettuja Markovin malleja aikajana muunnetaan matriisiksi, joka kuvaa laitteiden käyttäytymisen samankaltaisuutta. Ryhmittelymenetelmiä käytetään matriisin analysointiin, tavoitteena jakaa laitteet ryhmiin käyttäytymisen samankaltaisuuden perusteella. Datan ulotteisuutta pienennetään siihen soveltuvilla menetelmillä. Tämän jälkeen tulos visualisoidaan.

Testidatan määrä työssä oli pieni, vain 10 laitetta. Tulokset osoittavat jonkin verran lupausta menetelmien toimivuudesta ja oikeuttavat työkalun jatkotutkimuksen isommalla datamäärällä.

Avainsanat: esineiden internet, datalouhinta, koneoppiminen, massadata, laitehallinta

Contents

FOREWORD

This thesis was written while working for Arm Finland Oy. I'm grateful for having the opportunity to work on such an interesting topic for as long as I did. It's interesting to see where the Internet of Things develops and I'm glad to have a workplace like Arm to observe the transformation of communication and computing.

Special thanks are in order for my supervisor Susanna Pirttikangas for her feedback and ideas on the topic and on the research. My second supervisor Ekaterina Gilman provided great feedback as well and I'm grateful for that. I'd also like to thank my technical supervisor at Arm, Yongbeom Pak, for his tough questions that helped shape this thesis. Few other people at Arm also gave me ideas, explained inner workings our systems to me and gave me encouragement. Thank you all.

I couldn't have done this work without enormous amount of support from my friends and family. Thank you for being there for me and listening to my rants.

Oulu, Finland September 19, 2019

Antti Möttönen

ABBREVIATIONS

HMM	Hidden Markov Model
ITU	International Telecommunication Union
IoT	Internet of things
MDS	Multi-dimensional scaling
OMA LwM2M	Open Mobile Alliance Lightweight Machine to machine
OS	Operating system
PDE	Probability density estimation
RFID	Radio Frequency Identification

1. INTRODUCTION

Number of devices connected to the Internet is expected to be anything from 20.1 billion to 30 billion by 2020 [1]. Cheaper hardware, faster networks, and better software solutions make it possible to create entirely new types of products as even simplest and cheapest things can be connected to the Internet. Large amounts of data can be gathered from each connected device and from the cloud software enabling the devices. This thesis explores the first steps in transforming that data into valuable information.

Managing Internet of things (IoT) devices is hard. Many companies offer solutions for device management. Arm, Google, IBM, Amazon and Microsoft are all developing their own device management platforms for the growing IoT market. By utilising the data gathered from the devices and the cloud, hopes are that a lot can be learned about how customers use their devices, how devices behave with different configurations, etc. Potentially, utilising that information translates into value for Arm and Arm's partners. This process is called data mining.

Data mining and Machine Learning are somewhat overlapping research fields. Advances in machine learning are opening new possibilities across many fields. Prototypes of self-driving cars are driving on public roads. AlphaGo beat the best human Go player in the world. In this work analysis is performed with machine learning methods.

Arm has a service for IoT device management, called Pelion Cloud [2]. This work studies whether device management can benefit from the data mining approach on data gathered from IoT devices and software serving the devices. The goal of this thesis is to create a toolbox for analysing the data. The first step is to systemically list different data sources related to Pelion Cloud. The toolbox should then be able to find and visualise interesting information from the data. It is also of interest to find out if there are data sources that should be gathered from the ecosystem to provide useful insights about the connected devices. The operation of the toolbox is laid out in three parts:

1. Study clustering of data gathered from IoT devices connected to Pelion Cloud
2. Reduce dimensionality of gathered data
3. Visualise the results

The target for clustering is to find devices that behave differently from other similar devices. The presumption is that those devices are perhaps broken or otherwise in a state that needs to be inspected. Here, the scope of the information should be about IoT devices even though data from servers is also utilised.

This work is written to be introductory and explorative. The structure of this thesis reflects that by first introducing common concepts, before venturing into how to meet the goals of this work. Chapter 2 presents background information about what the IoT is and how Pelion Cloud is used in IoT device management. Chapter 3 introduces data mining theory and machine learning methods suitable for data mining operations in the environment. Chapter 4 discusses requirements for an implementation, considering the goals of this work and integration into existing services. Chapter 5 dives into analysing data available in Pelion Cloud. Available data is analysed and estimations are made on what data should be gathered in order to meet requirements for the system. Finally, an implementation to meet the goals presented above is made and evaluated in chapters 6

and 7. Chapters 8 and 9 conclude the thesis with analysis. This work is about handling the gathered data and doesn't discuss or show how to implement gathering the data, even though some proposals about useful data that should be gathered are made and are implemented in parallel to this work.

2. THE INTERNET OF THINGS

This section describes shortly how the Internet of Things is defined and what kind of visions there are around the idea. The main challenges regarding implementing the required technical solutions to realise the visions are discussed. Finally, Arm's ecosystem for the IoT, Pelion, is introduced as an implementation of the visions around the IoT.

The term Internet of Things goes back to 2003 [3 p. 2], used then for tracking goods automatically. The IoT is defined as "global infrastructure for the information society" by International Telecommunication Union (ITU) [4]. ITU furthermore continues the definition by adding how the IoT allows anyTHING to communicate. Traditional telecommunication technologies such as mobile phones, or laptops offer separation from time and place constraints regarding communication. Emails for example can be sent at anytime from anywhere and recipients can read them and reply whenever they wish. The IoT adds things as an unrestrained dimension in communication. Information exchange is no longer restricted by what is communicating, as it can be machine-to-machine or humans to things interaction.

The Internet of Things is in its core about connecting things that were not networked before. A device is defined as "a piece equipment that must be able to communicate" [4]. Additionally, a device may be able to sense, utilise data, etc. Furthermore, ITU also adds virtual things alongside physical things as a part of the IoT definition. Applications and multimedia aren't tied to any physical object but can communicate with other things, both physical and virtual in the IoT. Finally, scale is what separates IoT from similar attempts.

All these aspects form the definition of the IoT. There isn't a single system that is called the Internet of Things. Instead there are multiple approaches that focus on aforementioned aspects and can be called an IoT system. Figure 1 shows what might be involved in a such IoT system. Multiple devices are connected to a cloud, an application is analysing a dataset gathered from the devices and users are using an application to interact with the devices. This work is focused on the devices and the cloud software and is implemented as a service computing something from the data. This is highlighted in the figure.

The IoT is made possible by advances and falling costs in networking and hardware [3 p. 2]. Radio Frequency Identification (RFID) for example is a cheap way to identify and track almost anything from animals to passports [5]. ITU envisions a world where devices, applications and networks can be provided by different participants and work seamlessly [4].

IoT ecosystems can manifest in multiple forms. ITU lists five core roles that can be divided between different business players. Roles listed are:

1. Device provider
2. Network provider
3. Platform provider
4. Application provider
5. Application customer

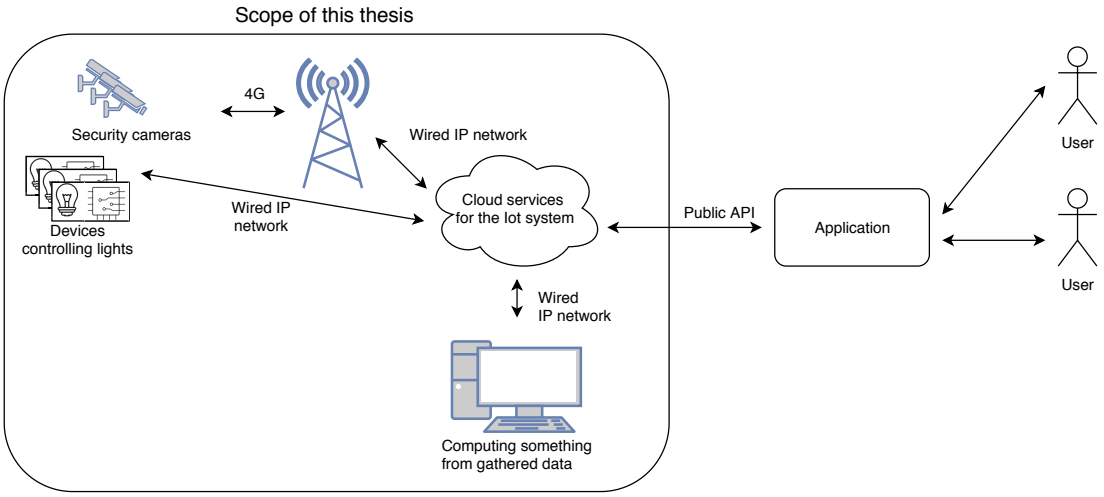

Figure 1. An example of what might be a part of an IoT system.

From these roles a number of different combinations can be formed as a basis for different IoT systems. For example, one business can take care of first three roles, leaving second company to create an application which is used by third party.

2.1. IoT opportunities

Traditionally businesses are based on value of products. According to Uckelmann et al. [3 p. 253] business models in the IoT world will revolve around information sharing instead. Since the IoT visions cannot be achieved by a single provider, there needs to be an incentive to share. Providing information might be paid or otherwise compensated to encourage sharing it with other providers.

Developments in IoT systems will lead to optimised information flow transforming communication [3 p. 22]. Business will at minimum get new information for decision making and might lead to completely new business models. IoT enables intelligent applications through vast scale sensing abilities [6].

Such advances in technology allow new business opportunities in many fields including industry sector, smart cities and healthcare. Goods can be tracked to produce and deliver right amount of needed goods in the right place at the right time. IoT sensor data can enable better traffic control to reduce carbon footprint and time spent in transfer. Personal assistants and health monitoring allow independent living for elders [6].

2.2. Challenges

There are plenty of problems in current IoT systems. IoT needs standards as there is no vendor that can implement the whole system and devices are heterogeneous [4]. As there isn't a clear winner on standardisation front but hardware is already cheap, there are products that are sold with weak or missing security, identity, or management

capabilities. For example, Mirai botnet managed to bring down a large DNS provider by utilising webcams and similar equipment with poor security [7].

Security needs to be taken care of on multiple levels. On application level normal security measures such as authorization, authentication, and data integrity apply. Security also needs to be taken care of on both network and device levels [4].

Devices in the IoT are heterogeneous. Connecting and communicating between heterogeneous devices is challenging. Common options are either to use a gateway through which the devices connect, or to specify a small set of supported protocols that the system supports [8].

Devices need to be managed. Management includes tasks such as activation, deactivation and software updates. Management solutions need to work with heterogeneous and resource constrained devices. Standardised device management solutions developed for devices such as routers do not work with IoT devices because of resource constraints [6].

IoT devices aren't always connected to power grid. Devices running on battery power cannot necessarily use traditional communication protocols because of the high power consumption. New protocols such as Zigbee have been developed especially for IoT [9].

Lack of standards has also led to a fractured world with multiple approaches such as Google Home [10] and Amazon Echo [11] for connected home, both having their own APIs. Interoperability between vendors is missing and devices are tied to one system.

Finally, the IoT solution should scale to billions of devices. This should be considered when designing systems.

2.3. Arm's IoT ecosystem

Arm offers software and services called Pelion IoT Device Platform. It's presented here based on materials from the site mbed.com [2]. Pelion Device Management is a full stack ecosystem solution providing support from an operating system for embedded devices to a device management software in a cloud. Pelion consists of two main parts: a software stack for devices and a cloud service for device management. Examples of claimed benefits by using a pre-built solution are faster time to market and verified security. In ITUs IoT ecosystem model [4] Arm acts as a platform provider and partially as a network and device provider. Following chapters provide more insight into the offering.

2.3.1. Overview of device software

Developing software for embedded devices differs from building software on computers in that resources are constrained and there is more variety on operating systems (OS).

ITU identifies several high-level requirements that affect how software is written for devices [4]. Interoperability is needed since the system is heterogeneous and networking needs to be automated because of the vast scale, for example. As an answer, Arm has developed an operating system called Mbed OS. The OS offers driver support for

multiple boards using Cortex-M processors. It's based on a real time operating system CMSIS-RTOS RTX. Mbed OS helps solve the following problems related to the IoT: security on device level, connectivity and cost. Mbed OS is licensed under Apache license 2.0 [2].

Figure 2 shows components of an IoT device running an application on Mbed OS. Arm provides Mbed OS with driver support for many different boards and hardware modules such as sensors. Additionally Cloud Client library is provided to handle communication with the cloud services. If the device is anything more than a simple sensor, the application is developed by a third party, such as the device or application provider. Hardware can be bought from any supported vendor. Additionally Pelion Cloud can be used without Mbed OS. Therefore the role of device provider is not always clear cut in Pelion ecosystem.

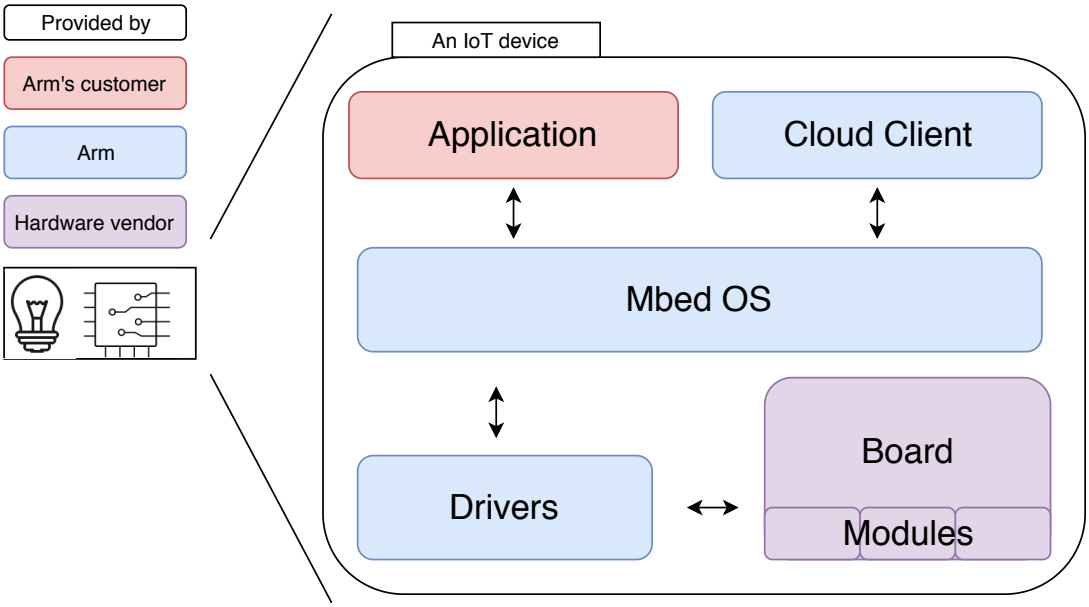

Figure 2. Overview of Mbed OS services.

2.3.2. Overview of Pelion Cloud

Pelion Cloud is a platform for IoT device management. Pelion Cloud can be used to fulfil many requirements identified by ITU [4], such as manageability and connectivity on behalf of users.

The offer includes connectivity libraries and cloud services from which customers may pick ones that fit their needs. Services include bootstrapping devices with security credentials, monitoring client resources and updating devices in the field. Devices can interact with the service using Device Management Client library installed on the device.

Mbed Cloud is built as a cloud offer, consisting of multiple micro-services. Therefore, it is possible to expand it with new features separately from other development work.

Base for communication between most devices and servers is Lightweight M2M by Open Mobile Alliance. Other options are available through gateways but are not explored as part of this work.

From user point of view Pelion Cloud consists of Application Programming Interfaces (API), Software Development Kits (SDK), cloud client for either Mbed OS or Linux and an online portal. Figure 3 shows an example of lifetime of an IoT device connected to Pelion Cloud. After manufacturing the device, an initial OS and software is installed on the device including libraries necessary to communicate with Pelion Cloud. When connecting for the first time the device performs a bootstrap procedure where the client gets an identity and security credentials needed to communicate with the cloud. During the lifetime of the device it monitors sensors and sends that data to cloud, reacts to environment based on sensor input or by instructions from an application communicating via the cloud APIs. When establishing a connection to the cloud after rebooting or connection loss the device performs a registration. Then, at regular intervals, the device sends registration update to signal that it's still on-line. At some point the device is decommissioned.

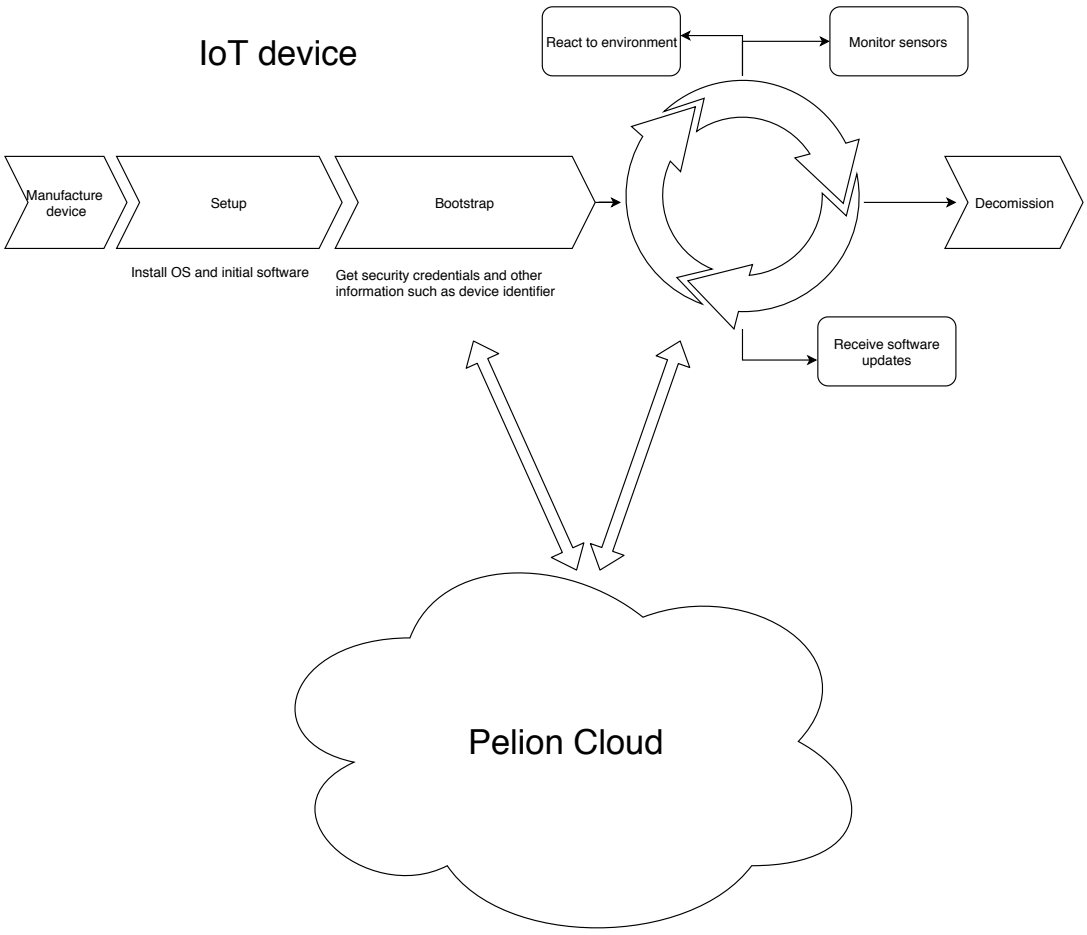

Figure 3. Typical lifetime of an IoT device connected to Pelion Cloud.

3. DATA MINING

Data mining is a process in which large amounts of data are analysed to gain interesting knowledge. The whole process is sometimes called knowledge discovery. This thesis uses term data mining for describing both the whole workflow and analysis part of the workflow. Data mining is an interdisciplinary field, including elements for example from database technology, machine learning, and statistics [12 p. 29]. Figure 4 shows the usual workflow in knowledge discovery [12 p. 6-8]. Analysing the data, where the term mining stems from, is just one step in bigger process, which includes gathering and preprocessing data and presenting the results.

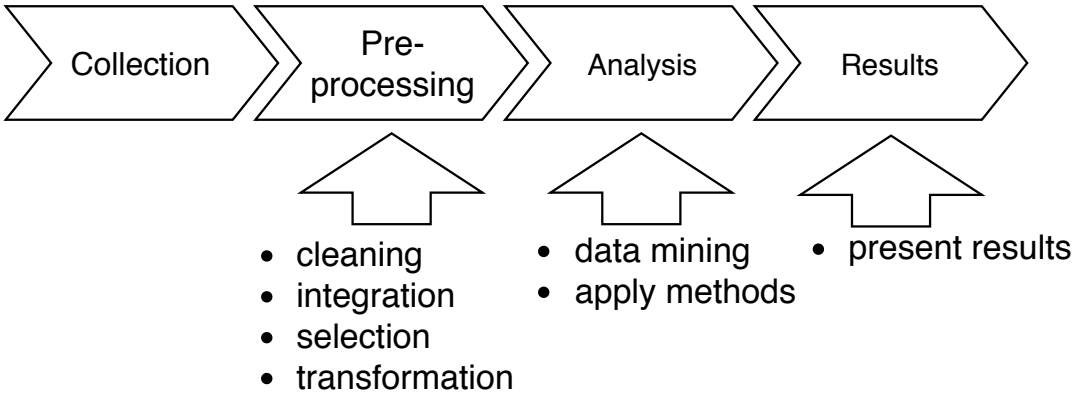

Figure 4. Data mining workflow.

This chapter focuses on preprocessing and analysis steps. Data collection is taken as a pre-requisite for this work and presentation is handled as a part of the implementation and results. Analysis in this work focuses on machine learning methods suitable for data mining. Theoretical background is presented before preprocessing steps to give reader context on why preprocessing is done as is.

3.1. Machine learning in data mining

This section presents information about what machine learning is and how it's used in solving problems usually related to large datasets. The ML methods relevant to this work are presented and discussed.

Learning in a traditional sense can have many semantic meanings as discussed by Witten et al. [13 p. 8]. Machine learning generalises rules from training data so that predictions can be made from unseen samples [14 p. 2]. Usually this means analysing patterns in data humans might not see. Methods to do so may vary from a simple expansion of probability theory like Naïve Bayes network or be inspired by biology like in Multi-layer neural networks but usually it still regards just different ways to find how output can be learned from input data.

Machine learning, as a part of artificial intelligence research, has had an impact on many distinct fields. Whereas humans might have problems with large datasets,

computers can analyse large amounts of data. For example, speech recognition is impossible to solve by using hand crafted rules to teach a computer to recognise each speaker, dialect, and language. However, given guidelines by labelling enough speech data by hand the algorithm can find the patterns necessary to solve the problem.

Machine learning is a wide field with multiple methodologies for different kinds of problems. Classifying flowers of different sub-species is quite different from machine language translation even though machine learning can be used to solve both problems.

Duda et al. conclude that pattern recognition is an empirical subject and selecting the universally best algorithm is impossible [15 p. 499]. For this work, a few different methods for different sub-problems are evaluated and presented. Figure 5 shows classical partitioning of machine learning methods. Following sections describe parts of the figure relevant to this thesis.

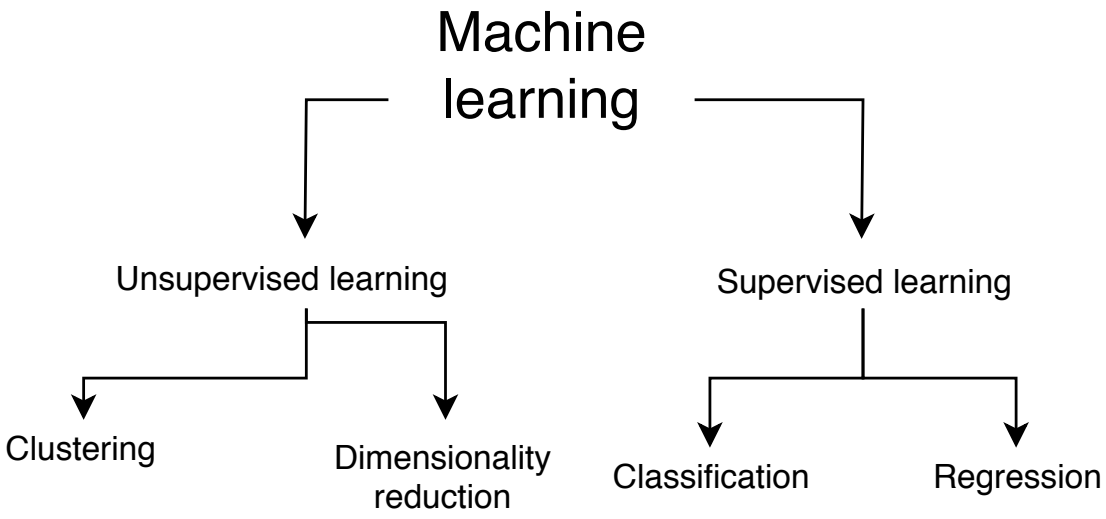

Figure 5. Classical partitioning of machine learning methods.

3.1.1. Supervised and unsupervised learning

Machine learning methods can be split into supervised and unsupervised learning. When using supervised methods, data is labelled and the algorithm learns what features correlate with correct classification. Supervised learning requires teaching datasets to teach the classifier [13 p. 45].

In unsupervised learning, the final number of target classes is not known [15 p. 517]. Unsupervised learning can be used to learn characteristics of the datasets without knowing much of it. For example, finding features that characterise the dataset might be used to reduce the amount of data that needs to be analysed. Shifting of characteristics found by periodically analysing a dataset over time can indicate an ongoing change that might be of interest. As Duda et al. note [15 p. 517-518], many unsupervised methods either assume properties of data and therefore are restricted on what kind of problems they can be used to solve or have no strong theoretical properties. Still, those methods may be beneficial for data mining purposes and once researchers

have better familiarised themselves with the dataset labelling and supervised methods may be used to continue the research.

As the purpose of this thesis is to be an exploratory and to semi-automatically learn about heterogeneous set of devices the methods used are unsupervised. Labelling required for supervised methods is undesired with varied datasets of the IoT world.

3.2. Unsupervised learning methods

This thesis utilises two different kinds of unsupervised learning techniques: clustering and dimensionality reduction. Both serve a purpose in making it easier to understand the dataset. Clustering is about reducing number of samples that need to be inspected when exploring the dataset. After grouping similar samples into one cluster, taking a closer look on one sample can reveal something about the whole cluster. Dimensionality reduction, in turn, reduces number of dimensions a sample has by discarding or calculating combined features [15 p. 580].

3.2.1. Clustering

Clustering is defined as finding similar samples from datasets, i.e. finding natural groups where samples resemble each other more than they resemble other clusters [13 p. 141]. Clustering is part of unsupervised learning as the ground truth is not available in the form of training dataset. Unsupervised methods like clustering, can be used for multiple purposes. Duda et al. [15 p. 517] identify five use cases. First is expanding labelled dataset training model with unlabelled data. Second use case is to find patterns with unlabelled data and label clusters by hand. Third use case is to track changes in data over time. Fourth use case is to find features of unlabelled data. Finally, unsupervised learning can be used to explore unknown datasets. This work focuses on finding patterns and exploring the dataset.

Even though clustering samples in "natural" clusters sounds simple there are two important details that need to be answered before clustering can be meaningfully done:

1. how to define similarity?
2. when to stop clustering?

The first point is concerned with metrics. Are two people similar because they have same number of limbs? Deciding wrong similarity metrics on unknown dataset can lead to poor results. The second point is concerned with what is considered an optimal solution. If the target is the least number of clusters then the optimal solution is one cluster containing all samples. On the other hand, if similarity between samples in cluster is to be maximised then the optimal solution is to put each point in its own cluster. These questions need to be answered in order to define a method for categorisation and in order to evaluate it. In addition, clustering becomes harder when no prior distribution or even number of classes is known.

Samples more like to each other than other samples can be thought to belong in a same logical cluster. Similarity is often defined as Euclidean distance between nodes

[15 p. 538]. Euclidean distance however can be distorted by scaling. It can be replaced with a similarity function returning a value that represents how similar two samples are. That requires however one to be familiar with the dataset to be able to define such function.

Often clustering procedures are iterative. K-means clustering, as presented by Witten et. al. [13 p. 142-144] is shown. It is often a solid and simple way to find clusters. Algorithm 1 explains how it works.

Data: K = number of clusters;
samples to cluster;
Result: Divide samples into K clusters
initialization;
select K random points as cluster centres **while** *Cluster changed since last iteration* **do**
    assign each sample into a cluster with closest centre using Euclidean distance;
    calculate mean of each cluster as a new cluster centre
end

Algorithm 1. K-means clustering pseudo-code.

K-means clustering requires knowing the number of clusters beforehand. If the desired number of clusters is not known one option is to try K-means with different numbers of clusters and then compare key metrics to find out natural number of clusters. Then the problem is of course choosing the metric. There are options such as minimum descriptor length which is about finding minimum number of bits to encode clustering information, but they're not covered in this work [15 p. 402].

K-means clustering starts from predefined number of clusters and works by assigning samples to different clusters. Clustering can also be done by starting with one big cluster and checking whether there's a meaningful way to split it further until natural clustering is found. Still, some limit for ceasing clustering process is needed. Otherwise each sample will form its own cluster because such configuration will always have least variance within clusters. One option is just to split the training set into training and validation sets and choose clustering so that error on validation set is minimized.

These kind of naïve clustering methods can be brittle [13 p. 352-353]. K-means clustering has been proven to find a local minimum but not necessarily a global one [13 p. 144]. Different clustering result can be found by changing start positions. If sufficient performance is not gained by K-means clustering another option is to use probability density estimation (PDE). Instead of creating definite clusters each node is given probability for belonging into each cluster. PDE can better consider outlier samples which might be noise and so shouldn't affect clustering too much. Using PDE better clustering performance can be achieved.

3.2.2. Dimensionality reduction

Visualising multidimensional data is hard. However, usually unfiltered datasets contain related, correlated and irrelevant data regarding problem at hand. Dimensionality

reduction is a group of techniques to find out which features contributed most to differentiate samples. Few most important features can then be visualised.

Suitable methods for dimensionality reduction depend on data at hand and goals of dimensionality reduction. For example, if the goal was to reduce the amount of data required to store letters, principal component analysis described below might indeed find a best way to preserve most of letters Q and O as they both have approximately the same shape, but might discard the data required to distinguish Q from O [15 p. 117].

Principal component analysis (PCA) is a transformation, in which data is projected along axis of greatest variance. PCA is helpful in revealing which components explained most of the data. By keeping only axes that explain the most of the variance in the data dimensionality can be reduced without losing too much information. When dimensionality of data is reduced visualisation is easier. [13 p. 305-306]

PCA is based on calculating covariance of the coordinates of the data points and then calculating eigenvalues and eigenvectors of that matrix. Eigenvectors are the axes in the transformed space and vectors with the largest eigenvalues explain most of the data [13 p. 306].

Independent component analysis (ICA) is a method to define which components depend least on other components [15 p. 570]. In real world scenarios it might be difficult to define whether components affect each other and if that should be considered when analysing data.

PCA and ICA are linear projections of the data. They miss non-linear structures. Manifold learning is a way to do non-linear dimensionality reduction. Isometric feature mapping, or isomap for short, is presented as an example of manifold learning. Isomap itself is based on method called Multi-dimensional scaling (MDS). In MDS a projection is generated that respects the distances between the samples in for example Euclidean space. Isomap is similar to MDS, except that distance is geodesic. That means calculating distance along the manifold where the samples lie. The manifold can be constructed with K nearest neighbours and distance can be calculated by traversing the graph constructed from the linked neighbourhoods [14 p. 596] [16] [15 p. 573].

Feature agglomeration is a hierarchical method to merge features together in a way similar to hierarchical clustering. All features start as their own clusters. Most similar features are selected based on some criteria, for example by calculating feature pair with least variance. That pair of features is then merged by for example calculating mean of the features for each sample. The procedure is repeated until desired number of features remain [16].

3.3. Handling temporal data

Methods presented above expect the data to be fixed width matrix of numeral data. The data however is not always just features of samples. Sometimes the data is temporal, a time series of feature values describing samples. For example, events gathered from device registrations are temporal data. Each registration produces a new data point. Each sample has a variable number of data points. Each data point might have multiple features per event and as such methods discussed so far cannot be used on analysing such data. Transforming events into features describing samples is required.

The easiest way is to calculate some statistics that reflect the events. However, this requires quite intimate knowledge of the data. For example, calculating average over all values might miss important fluctuations in the values. Bicego et al. [17] propose a solution based on hidden Markov models (HMM).

Hidden Markov models consider the data to be generated by a model with a sequence of states where the probability of a state is influenced by the previous state only. The states however are not known, only some symbols emitted by the state from which the probability of a state can be calculated. There are three main use cases for HMMs. First is evaluation where model parameters are known, and probability is calculated for a particular sequence of states. The second use case is decoding where the goal is to calculate the most probable sequence of hidden states from observations. Thirdly, in learning a sequence of observations can be used to train an HMM given that the number of states is known [15 p. 130].

The idea in Bicego et al. [17] approach is to fit an HMM for each sample sequence and then compare sequences to find which sequences fit in other models. This similarity metric can then be used in clustering using methods presented above. Algorithm 2 describes the procedure.

Data: Set of events $\{O\}$ for each sample S
Result: Set of features for each sample defining similarity with every other sample
Fit an HMM λ_i for each set of events;
foreach *HMM* λ_i **do**
  foreach *set of events* $\{O\}$ **do**
    Calculate log probability with given HMM divided by length of the sequence

$$\frac{P(\{O\}|\lambda_i)}{len(\{O\})}$$

  end
end
Use the generated distance matrix as features in clustering samples

Algorithm 2. HMM based similarity clustering[17 p. 89-91].

Fitting an HMM is done by Baum-Welch algorithm also known as forward-backward algorithm. It's an iterative approach that will settle on local minima for the best values for the HMM. The log probability is calculated using forward part of the forward-backward algorithm. It gives probability that a sequence of events was generated by the model [15 p. 130-139] and is used in algorithm 2 to measure similarity between samples.

3.4. Validation

In ML the correct result depends on application. Mistakes will happen in classification because of noise and finite number of samples. For example, when screening cancer patients even small change of cancer might warrant more thorough examination, i.e. false negatives are unwanted and false positives might be considered tolerated if false

negative rate is low. Some other use case might see false positives as an equally high cost and therefore algorithm needs to be tuned correspondingly. [14 p. 41]

Usually classifiers can have two sorts of error. Bias happens when learning set is matched too closely and results generalise badly. Variance instead means generalising model too much and missing on important details in the dataset [14 p. 465-468]. As a rule, a test set can't be used for verification in supervised learning.

Usefulness of found cluster depends on the intended use case. In explorative work usefulness of the result is largely subjective attribute.

For empirical methods dimensionality reduction can help in deciding whether results were meaningful.

3.5. Data preprocessing

Data mining often uses data that's available. The data is not perfect. It might have been gathered for another purpose or contain human errors [13 p. 62-65]. Therefore it is important to preprocess data. Furthermore, datasets might include values in non-numeric formats that need to be converted into formats suitable for algorithms presented above. These issues are discussed in this section.

3.5.1. Sanitation

Most ML algorithms require the data needs to be scaled before its used. As shown above, many methods calculate distances between samples in n-dimensional space. If two different features have vastly different scales, one feature will dominate distance calculations distorting the results. Usual options for scaling the data are min-max scaling and standardisation. Min-max scaling is also called normalisation. It is achieved by scaling values to a range of [0, 1].

Standardisation maintains information about outliers and is preferable in many machine learning algorithms. Standardisation is calculated for each feature in each sample with formula 1:

$$x_n^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x} \tag{1}$$

where $x^{(i)}$ is the original value of a feature, μ_x is mean of the feature across all samples and σ_x is standard deviation of the feature [18 p. 141-143].

Nominal attributes, for example whether a device has a specific sensor, must be converted into numeral attributes for algorithms using distances between instances. Easiest way to do so is to set distance to 0 between similar instances and 1 otherwise. This can be achieved with so called one-hot encoding. Each distinct value for nominal feature is made into a new dummy feature where value for the dummy feature is 1 when a sample has that value and 0 otherwise [18 p. 136]. Using domain specific knowledge, a better representation can be achieved [13 p. 85].

3.5.2. Missing values and outliers

In more complex cases the data might be incomplete or be corrupted and require further processing. Changes in a way data is gathered might produce missing values. If data is gathered by hand, there will be human errors in large datasets. Samples that do not appear to belong with the rest for whatever reason are called outliers [13 p. 320].

Easiest way to work with outliers is to ignore features or samples with missing data. However much valuable data might be lost doing so [18 p. 313]. By replacing missing values with mean, median or most frequent value will lessen the impact. However, as Witten et al. note [13 p. 62] understanding reasons for missing data might be crucial.

Outlier detection is a common name for methods that try to detect samples that are different from the rest. Outlier detection is usually used in preprocessing to find out samples that are polluting the data. A few outlier detection methods implemented in scikit-learn framework are presented here shortly [16].

Elliptic envelope fitting assumes data comes from a known distribution. Samples that do not fit near the distribution are deemed outliers. The problem obviously is that if the distribution is not known then the method might not produce meaningful results.

Isolation Forest uses technique called random forests. A feature is selected randomly, and splitting value is chosen also randomly between minimum and maximum values of that feature. By using these splits samples can be stored into a tree structure. Then the tree can be traversed until the sample is isolated and length of the path traversed is calculated. Anomalies have shorter path lengths than with samples like each other.

Local Outlier Factor is a method which measures the density of samples. Samples in lower density areas are outliers. Average density of selected number of neighbours is compared to density of the sample. If the sample has lower density than that of its neighbours, it's considered to be an outlier.

The downside of many outlier detection methods is that the outliers shouldn't form tight clusters, or the samples will not be detected as outliers. If the samples with errors are clustered, perhaps due to systematic collection error, outlier detection might not be effective.

3.5.3. Data formats and transformations

Sometimes raw data itself is not interesting before preprocessing. For example, if a device sends events throughout its lifetime the absolute time of the event might not be useful. Time from device boot or average time between events might however reveal something.

Another option would be for example day of the time if it's known that devices have some sort of cycle related to a day. Street lights, for example, have a natural 24h cycle and so events on midday might indicate something.

HMMs can treat sequential data. Preprocessing methods shown above can be used to transform the data for HMM to process. To preprocess the data a decision must be made on how to combine different samples. Simplest approach is to combine all of the events across all the samples as if the events were individual samples.

4. REQUIREMENTS

Software developed as a part of this thesis, could be run as a part of the Pelion Cloud introduced in Chapter 2. The goals of this thesis and the requirements from existing software and infrastructure are combined in this section into requirements for the implementation.

4.1. Product

The goal of this thesis is to uncover interesting information from the data available. The goal is to find devices behaving differently from other devices with similar hardware and software. The software will have two main functions. The first step is to use data mining and machine learning methods to analyse the dataset. The second step is to visualise the results. In the first, step samples are divided into groups and data is transformed into a form where it's possible to visualise it. The second part is to visualise the data for users to explore.

4.1.1. Functional requirements

The service must combine data from any suitable sources Arm has at the moment. Analysed data should be selected so that the scope of data is on device behaviour, so for example server performance, although interesting, is not in the scope of this thesis. Gathering the data should not, if possible, require modifications on device or cloud software.

Data aggregation must be done in a way that does not overburden other services or devices. Aggregated data could be stored or cached in order to reduce the costs of long-term operations and minimise distribution to other services.

Data is analysed with three types of methods, dimensionality reduction, clustering, and outlier detection. The goal of dimensionality reduction is to make it possible to visualise devices consisting of maybe tens of features and to perhaps improve performance of clustering. Clustering and outlier detection are done to help users visualise what kinds of natural groupings devices might have and find out devices misbehaving.

Clustering is done with K-means. As this work is about exploring and visualising the data, clustering can be replaced by outlier detection methods instead to fill the same goal of finding devices behaving unlike the rest. Such methods include elliptic envelope fitting, isolation forest, or local outlier factor. Dimensionality reduction is done with algorithms presented, i.e with PCA, MDS, Isomap, or Feature Agglomeration.

Visualisation should be two- or three-dimensional scatter plots with additional dimensions such as clusters shown with colours and/or shapes.

As stated, the service could be run as a part of Pelion Cloud. It should be implemented as a HTTP/JSON API and a web page utilising the API. Visualisation must not require any browser plugins or extensions. Authentication is not required at this point as the product will be running in a closed environment.

To make it possible to run the service as a part of the Pelion Cloud the program should be runnable inside docker container and be stateless. Any data should be in a separate database.

The service should not affect normal operation of the devices. Ideally any device connected to Pelion can be analysed without requiring any modifications on the device software and without using the devices resources such as network or battery. For now, the results will be available for internal use only. As the service will not be exposed to anyone outside Arm it will not be necessary to further anonymise data.

5. DATA ANALYSIS

Different data sources available currently are explored. Finally, modifications needed to existing data sources in order to improve results of this thesis are estimated.

5.1. Requirements for the data

The goal of this thesis is to help in gaining insights about IoT devices. To be useful, data gathered about devices must be identifiable on per device level granularity. Therefore, some data that is not about devices itself, like server CPU usage won't be useful in this work even though it could be interesting if scope were to be changed to service monitoring for example.

Since not all users use all features of the cloud and have different devices, some data may be missing and needs to be taken into account when selecting the features. Most useful features would be those used by as many users as possible or those that could be added with minimal work by users and operate without significant overhead on device resources.

There is no way to identify any personal details. Ideally, device data should not be compared between accounts to make it possible to offer the service to users of Pelion Cloud at some point and mixing up information from multiple accounts would be a security risk.

5.2. Current data sources

In this thesis data source, is defined as a database, API, service or any other type of solution that has data about devices connected to Pelion Cloud. Currently there are multiple data sources available that could be used in analysing devices. This section goes through the sources explaining the data.

5.2.1. Device Directory

Device Directory is a service that has the official data about the device such as its name, current state and owning account. Table 1 illustrates what is stored about a device. Device Directory data could be extended with new fields for analysis if needed.

Device Directory data is quite static and does not reveal much of device behaviour. Some fields like state can be used to detect devices that aren't functioning as expected in simple cases, but the dataset offers not much for automated learning and rather calls for hand crafted monitoring rules.

Device Directory also provides an API with events related to device operation. Every time a device registers, re-registers or goes offline an event is created describing what happened. Furthermore, whenever Device Directory data about a device changes an event is also created describing the change. Especially registrations could be used to detect how a device behaves, as it is automatically gathered, is available on all devices and is directly linked to device life cycle and behaviour.

Table 1. Device Directory data about a device

Field name	Notes
Account ID	Account of the owner of the device. Useful for filtering results
Device execution mode	Separates production and development devices
Bootstrap expiration data	
Bootstrapped timestamp	
CA id	Which certificate authority created credentials for the device
Connector expiration date	The device requires rebootstrapping after this date
Created at	
Custom attributes	User can define attributes for the device
Description	
Device class	Model and hardware version of the device
Id	Unique identifier for the device
Device key	Fingerprint of the device certificate
Endpoint name	Unique customer given name for the device
Endpoint type	
Etag	
Firmware checksum	
Host gateway	Name of the gateway device if used
Manifest timestamp	
Mechanism	Either connector or direct
Mechanism url	Address of the connector
Name	User defined name for the device
Object	
Serial number	
State	
Updated at	Timestamp of last update of the device info
Vendor id	Identifies device vendor
Enrolment list timestamp	

However, event data requires some preprocessing to be meaningful. A decision must be made on how temporal data is handled. Should timestamps of events be treated as calendar time, time from device boot or perhaps by time of the day the event occurred? Additional challenges arise from the fact that the number of events is not fixed. One device might have several registrations while another registers just once. Table 2 lists fields that can be found from a single event.

Table 2. Device event

Field name	Notes
Changes	
Data	
Date and time	
Description	
ID	
Device ID	Used to link to a device
Event type	
Event type description	
State change	Whether the event included a state change

5.2.2. Querying devices directly

OMA LwM2M specification allows querying data from devices directly. Data is queried as objects, some of which are defined in the specification. Table 3 shows objects that are specified as a must have for a specification compliant device [19].

Table 3. Different object types that can be queried through LwM2M interface

Object type name
LwM2M security
LwM2M server
Device

There are also other object types defined in the specification and the specification allows for the device to provide new data types as well. Devices might or might not implement those resources additional resources. Querying this information requires knowledge about what resources the devices implement. Gathering data on devices might be slow if the devices spend most of their time in low power mode. Because of these limitations gathering data on device will be dependent on use case and is not suitable for a general-purpose solution.

5.2.3. Prometheus

Prometheus is a monitoring tool used to gather data from various sources. Each service in Pelion Cloud provides data to be gathered by Prometheus. Specific metrics are individually defined for each service and there are no cloud-wide rules on what to collect. A service might gather for example its memory usage, queue sizes and average latencies. Unfortunately, the data gathered in Prometheus are on service level granularity and not usable when inspecting single account and devices tied to it.

Each service maintainer gathers Prometheus data as they see fit. Creating guidelines on how to gather data would allow more general approach to analysis.

5.2.4. Statistics and Billing

Statistics is a service which gathers events such as bootstraps, registrations and registration updates. The data is gathered on per account granularity and as such is not useful in learning about behaviour of individual devices. Modifying statistics and billing services to collect data per device would provide additional information about device life cycle with no disturbance on a device i.e. it's collected on server side based on what the device is doing without any modification needed from device software developers.

Statistics has timestamps on device events and to be meaningful it needs to be correlated with device lifetime like with Device Directory events.

Billing service is similar to statistics. Events are stored for bootstraps, transactions and firmware updates.

5.3. Notifications

Pelion Cloud has APIs for notifications that can be utilised. Notifications allow monitoring device status through events that are sent to subscribers. Notifications can follow device life cycle events such as registrations or can follow LwM2M resources on the device. Notifications are delivered through callbacks or by querying the cloud APIs.

5.4. Additional data

This section describes possible data sources that might be interesting. Statistics and Billing gather data on the cloud with no cost on device perspective. If it were to be changed to be gathered on per device granularity it could be used on analysis like device events. The added benefit with billing is that the data is directly related to operating costs of devices. Non-critical devices with suddenly increasing billing costs could be automatically suspended for example. Of course, care needs to be taken not to over engineer the solution. Billing costs can probably be calculated without machine learning.

Devices have a lot information, such as hardware configuration, software version, etc. that might be useful, but is not gathered. Creating a library for gathering data on device would allow easily querying that data. Additionally, if added costs (power consumption, processing power and network usage, etc.) are accepted, device runtime data could be gathered. Devices could monitor memory, CPU and network usage for example and provide time series for analysis.

A prototype of such project exists. Device Monitoring (DM) project within Arm collects heap and stack memory usage of devices and exposes them as LwM2M resources. Those resources are monitored, and the results are stored in a MongoDB database. The database also contains event data gathered through notification service.

Low-level network monitoring could identify activity that's not logged on service level, such as TCP keep alive messages. Identifying devices on network level and storing the data for analysis might require quite significant processing power and a great deal of storage so in addition to possibly significant implementation costs of such data gathering. Therefore, it should be investigated only if there's a clear use case for the data and value in its analysis.

Pelion Cloud is also available as an on-premise setup. In such case, all devices might belong to one customer. Under such circumstances it might be possible to utilise cloud-wide data if combined with other information, such as device update. Updating half of the devices might show changes in service behaviour, revealing a problem in the new device software. Machine learning could be utilised in detecting whether the increase of activity related to updates is normal or not.

6. IMPLEMENTATION

This section describes how algorithms in Chapter 3 were used in implementing a platform for separating devices into groups based on behaviour of the devices over time. The hypothesis is that correctly working devices with the same software behave similarly and a faulty software or hardware changes the behaviour in a way that can be detected by analysing gathered data. The hidden Markov models combined with unsupervised learning methods are tested for this analysis. Finally, the results are visualised after reducing the number of features in the dataset with dimensionality reduction. Figure 6 shows overview of the process overlaid with corresponding data mining steps. Details of the process are explained in following sections.

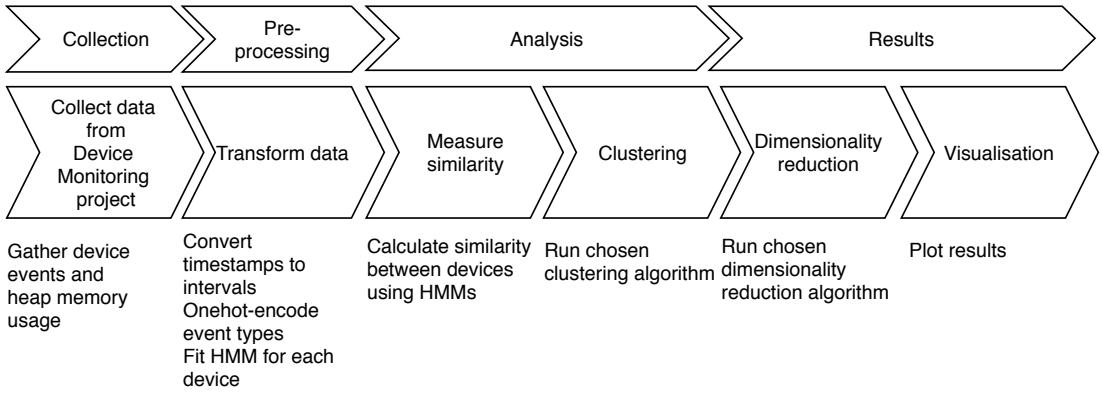

Figure 6. Overview of the process overlaid with corresponding data mining steps.

6.1. Data

Device Monitoring project mentioned in Chapter 5 was utilised in this work. In the project there are some devices running, gathering sensor data. For the devices in the DM project, all OMA LwM2M resources are monitored, and registrations and deregistrations are recorded using notification channels. That database from that project was selected in this work as a basis for the analysis as it contains sequential data from real devices. DM project was also altered to gather device registration updates which were not gathered before.

Only sequential data was used in implementing the segregation procedure. Device connectivity events mentioned in chapter 5 were combined with device heap memory usage. This resulted in three parameters that were used for the analysis. Device events provide series of event types with timestamps providing first two parameters. Events were selected because they're available on all devices and are directly related to device behaviour. Table 4 shows an example of what the unprocessed event data looks like on one sample.

Third parameter used was device heap memory usage. Memory usage was extracted by using added software library on the devices available for testing. Heap usage was gathered separately from the device events using LWM2M resource monitoring. Table 5 shows an example of memory data collected.

Table 4. Example of event data used in analysis

Timestamp	Event type
2018-07-30T09:16:12+00:00	registered
2018-07-30T09:23:43+00:00	re-registered
2018-07-30T09:31:15+00:00	re-registered
2018-07-30T09:38:47+00:00	re-registered
2018-07-30T09:39:17+00:00	dropped
2018-07-30T09:59:12+00:00	registered

Table 5. Example of heap memory usage values

Timestamp	Heap usage
2018-07-30T09:11:12+00:00	57000
2018-07-30T09:18:42+00:00	55123
2018-07-30T09:25:12+00:00	57035
2018-07-30T09:17:12+00:00	57091
2018-07-30T09:19:12+00:00	57781
2018-07-30T09:21:12+00:00	57651

6.2. Data preprocessing

As explained in section 5 data is often not suitable for use before some preprocessing. In this case timestamps of events aren't in itself interesting or comparable. Two same kind of devices started on different times should still show up as similar which is impossible if the timestamps are used as is. As the devices should re-register on known intervals, inconsistent or larger than expected delays between events might indicate problems. Therefore, timestamps for re-registrations were converted into seconds from the last event. For registrations following dropped connections timestamp values were set to 0 since the devices might be offline for a reason not related to problems with the device as the devices were used in the DM project. Inconsistent values with the downtime might cause device to be labelled as a malfunctioning device incorrectly.

Event types are a set of predefined values. There were four different values found in the database: registered, re-registered, dropped, and expired. Dropped and expired were combined into one category to indicate connection issue. To make it possible to use event types with HMM, values were converted with one-hot encoding into three distinct features.

The entries in memory usage table do not match exactly with device lifetime events. For each entry in the device events the nearest heap usage value for that device was used. The memory value was in bytes and was left as is. Table 6 shows an example of what combined data for one device might look like after preprocessing.

Table 6. Example of event data after preprocessing for a single device

Time since last event	Flags indicating event types			Heap usage
0	1	0	0	57000
451.00	0	1	0	55123
452.00	0	1	0	57035
452.00	0	1	0	57091
30.00	0	0	1	57781
0	1	0	0	57651

6.3. Similarity Measurement

Device behaviour similarity was measured by using Hidden Markov Models. For each timeline constructed from device events and heap memory usage an individual HMM was fitted. Only parameter given for the HMM was the number of states for the model. Otherwise defaults from "hmmlearn" library were used. Then probability of each timeline was calculated for each HMM using forward algorithm described in Chapter 3. The output of the HMM gives a likelihood on the similarity of the devices, effectively forming a distance matrix describing how far the devices are from each other measured by likeness of devices behaviour. These similarity numbers were used as features in the next section. Data was not normalised for this step since mean and standard deviation of observations are what HMM uses in describing the hidden states.

6.4. Clustering and outlier detection

Clustering and outlier detection were used to find groups of devices behaving unlike the rest. Similarity metrics obtained in the previous step were used as features describing the devices, replacing original dataset. First the dataset was standardised using algorithm 1. K-means was selected as a clustering algorithm and outlier detection methods local outlier factor, isolation forest and elliptic envelope were used to separate devices into inliers and outliers. K-means takes the number of clusters as a parameter whereas the rest use contamination percentage to sort out fraction of the devices to classify as outliers. Otherwise parameters were defaults from "scikit-learn". Algorithms evaluation details are presented in Chapter 7.

The selected algorithm was run with the similarity dataset. Each sample was given a cluster by the algorithm. The cluster is also given a confidence value ranging from 0 to 1 to support PDE if such algorithm were added. For algorithms providing just one cluster the confidence is 1. Outlier detection algorithms provide the result as outlier or inlier. Those values were then used as clusters with confidence value of 1.

6.5. Dimensionality reduction

Dimensionality reduction was implemented to enable visualising multidimensional dataset. Similarity data from first step was used as the input, like in clustering. PCA, MDS, Isomap and Feature Agglomeration were used as dimensionality reduction methods. A desired number of components to keep or form is the only parameter required. The result was a matrix of the similarity data with only desired number of features or dimensions per device.

6.6. Architecture and implementation

The service was built as a HTTP Server providing both a backend with a JSON API for querying different data sets and a simple front end for visualisation purposes. An option was added to API to switch the order of clustering and dimensionality reduction operations. It has two uses. First, if the clustering is slow because of too many dimensions, the DR algorithm can reduce the performance impact. Second, it is used to analyse if using DR loses information in a way that impacts data presentation reliability. Figure 7 shows the architecture including the Device Management project.

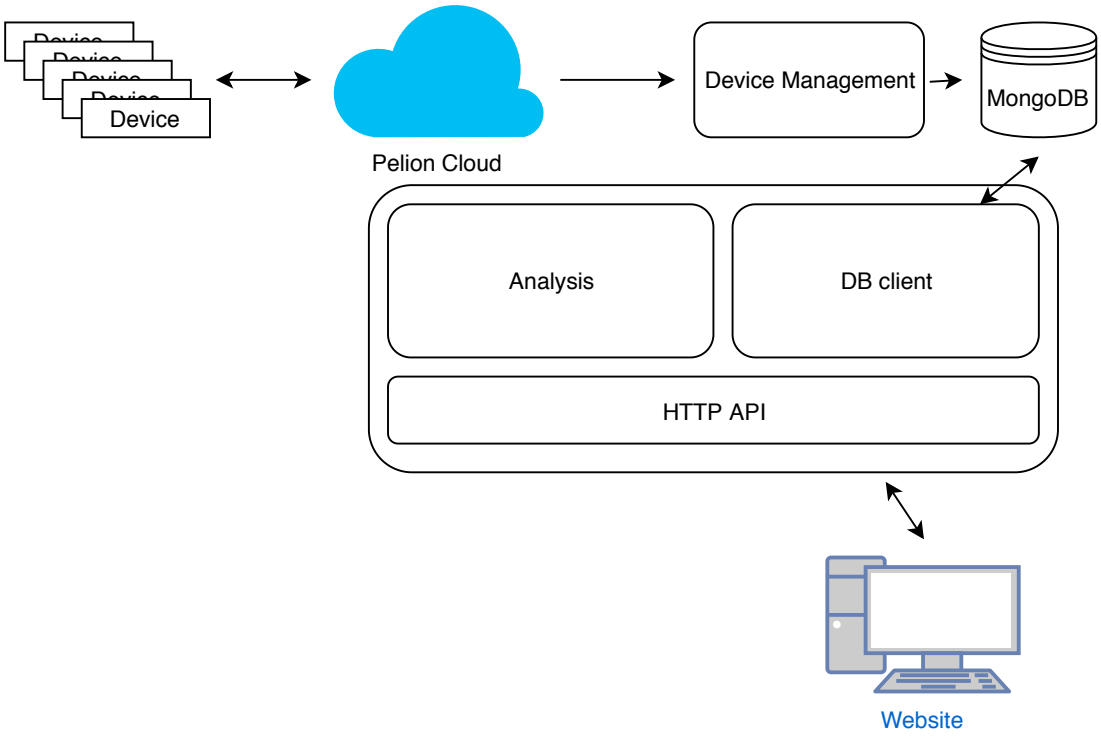

Figure 7. Architecture of the implementation

The devices connected to Pelion Cloud provide the data. The data is gathered on the MongoDB outside the cloud. The service is running as a web application, in this work offline.

Python was selected as a versatile mid-ground used in both machine learning and web services. Django framework was used as a basis for the web service. Scikit-learn library provided most of the required machine learning and data mining functionalities. Hmm-learn, a forked version of scikit-learn was used for HMM computations. Numpy was heavily used for data structures and manipulation.

The frontend functionality was written in Javascript. D3 library was used for visualisation.

7. TESTING

This section introduces how the implementation was tested and presents the test results. All tests were run against the HTTP API the server exposed using Python and requests library. Tests, which are explained below, were run 50 times, as some of the algorithms used are sensitive to randomised setup conditions. 50 runs might be too much for such a small amount of devices but it's configurable and calculating results over multiple runs was implemented if tests are continued on a larger dataset. Resulting averages, medians and standard deviations calculated from combined results are demonstrated.

7.1. Test data

The testing setup consisted of 10 devices used in the Device Management project within Arm. The devices used were Freedom-K64Fs running Mbed OS and Pelion Cloud Client and the library used for collecting memory usage data. Four of the devices were rigged with a bug in the device software causing memory usage to increase over time. The memory leak was done by allocating memory on random intervals 256B-1kB at a time. After approximately 4-6 hours the device will reboot when heap memory usage goes too high. The test devices were not dedicated to this thesis and some had for example downtime unrelated to this work, possibly degrading quality of the results.

The goal of the testing is to separate the devices into clusters based on their behaviour, i.e. to separate correctly working devices from the ones with buggy software.

7.2. Data preprocessing and similarity measurement

Performance of event data loading from the database and the time spent on preprocessing and HMM fitting and similarity measurement was recorded. Table 7 shows the results. First row shows average and standard deviation for time spent on querying device event and memory usage data from the database and combining those into the event series data used in the similarity analysis. The second row shows average and standard deviation for time spent on HMM fitting and similarity estimation. In addition, it contains the preprocessing steps described in the previous chapter i.e. one-hot encoding for status events.

HMM used two states for the devices as the presumption was that the devices are either offline or working as expected. The accuracy and usefulness of similarity data is measured by clustering accuracy.

Table 7. Time spent on data handling steps in

Step	Average time (ms)	std deviation (ms)
Preprocessing	20612.33	445.23
HMM	3999.49	148.68

7.3. Clustering and outlier detection

Clustering algorithms used require either the number of clusters or percentage of outliers as parameter. Clustering was performed so that there were as many clusters as categories so two clusters were formed. Correspondingly percentage of outliers was set to 40% as there were four malfunctioning devices out of ten. As clustering and outlier detection were both used to fill the same function of separating malfunctioning devices from working ones, both are referred here as clustering methods.

The clustering is an unsupervised process and the clusters returned by the clustering algorithm aren't named, but are instead given arbitrary labels "1" and "2" by the clustering algorithm. The optimal clustering algorithm would divide the devices in two clusters with 6 (working) devices in one cluster and 4 (malfunctioning) devices in the other. To measure performance of each clustering method, clusters were renamed using information not available to clustering algorithm with the presumption that most of the devices were clustered correctly. To do so clusters were labelled working and malfunctioning in a way that greatest sum of devices (working + malfunctioning) were in the correct cluster. I.e. if cluster one contained five working and two malfunctioning devices and cluster two contained one working and two malfunctioning devices cluster one was labelled working and cluster two malfunctioning so that the sum of correctly labelled devices was the highest, seven devices.

Then true positives, true negatives, false positives and false negatives were calculated. It should be noted that strictly speaking this action takes an optimistic approach as the labelling is done on outside knowledge, not available in a real scenario. However, since this work is not about making decisions but about providing data to analyse this approach was deemed suitable as it's a familiar tool in analysing results in scenarios like this. Tables 8 - 11 present the results for each algorithm used. The results are calculated over 50 runs. An optimal algorithm would cluster 60% of devices as working (upper left corner of the table) and 40% of devices as malfunctioning (lower right corner of the table) with 0% of false positives and false negatives (lower left and upper right respectively). Results interpretation is given in Section 8.2.

Table 8. Results of clustering with K-means

		Classified as working	Classified as broken
Working devices	Average	59.0%	1.0%
	Median	60.0%	0.0%
	Std deviation	3.0%	3.0%
Malfunctioning devices	Average	27.4%	12.6%
	Median	30.0%	10.0%
	Std deviation	4.4%	4.4%

Run time for each clustering algorithm was measured on the server. Table 12 shows execution times in milliseconds.

Table 9. Results of clustering with local outlier factor

		Classified as working	Classified as broken
Working devices	Average	40.0%	20.0%
	Median	40.0%	20.0%
	Std deviation	0.0%	0.0%
Malfunctioning devices	Average	20.0%	20.0%
	Median	20.0%	20.0%
	Std deviation	0.0%	0.0%

Table 10. Results of clustering with isolation forest

		Classified as working	Classified as broken
Working devices	Average	50.0%	10.0%
	Median	50.0%	10.0%
	Std deviation	0.0%	0.0%
Malfunctioning devices	Average	10.0%	30.0%
	Median	10.0%	30.0%
	Std deviation	0.0%	0.0%

Table 11. Results of clustering with elliptic envelope clusterer

		Classified as working	Classified as broken
Working devices	Average	51.4%	8.6%
	Median	50.0%	10.0%
	Std deviation	10.4%	10.4%
Malfunctioning devices	Average	28.8%	11.2%
	Median	30.0%	10.0%
	Std deviation	12.1%	12.1%

Table 12. Time spent on clustering

Algorithm	Average time (ms)	std deviation (ms)
K-means	21.12	3.13
LocalOutlierFactor	3.52	2.16
IsolationForest	261.76	43.08
EllipticEnvelope	81.43	27.70

7.4. Dimension reduction

The dataset, which at this point has 10 attributes measuring similarity to each device including itself, was reduced into two or three dimensions as two or three dimensions

are still possible to visualise on two-dimensional screen quite easily. Dimensionality reduction was tested by first doing a clustering run without doing dimensionality reduction. Results for that run were saved. Then the service was run so that dimensionality reduction was done on the similarity dataset before clustering algorithm was run. Results for this run were also saved. Clusters for both runs were named independently as with clustering results. The results of the former run were compared with results of the latter run. Any devices that changed clusters between the two runs indicate dimensionality reduction lost information in a way that altered the results. Tables 13 - 16 show the results. As with the clustering the tests were run 50 times and numbers are calculated from all runs. Results interpretation is given in Section 8.3.

Table 13. Percentage of devices that changed category while using K-means

		PCA	MDS	Isomap	FeatureAgglomeration
3 dimensions	Average	21.6%	9.4%	18.0%	13.6%
	Median	20.0%	10.0%	20.0%	20.0%
	Std deviation	4.2%	9.5%	6.3%	7.7%
2 dimensions	Average	21.6%	10.6%	14.4%	16.8%
	Median	20.0%	10.0%	15.0%	20.0%
	Std deviation	4.2%	6.1%	8.3%	6.5%

Table 14. Percentage of devices that changed category while using LocalOutlierFactor

		PCA	MDS	Isomap	FeatureAgglomeration
3 dimensions	Average	58.8%	14.0%	49.2%	1.2%
	Median	60.0%	0.0%	40.0%	0.0%
	Std deviation	4.7%	21.3%	10.0%	4.7%
2 dimensions	Average	36.4%	29.6%	50.0%	28.4%
	Median	0.0%	20.0%	80.0%	0.0%
	Std deviation	38.9%	30.1%	37.8%	37.3%

Table 15. Percentage of devices that changed category while using IsolationForest

		PCA	MDS	Isomap	FeatureAgglomeration
3 dimensions	Average	2.8%	0.4%	0.0%	1.2%
	Median	0.0%	0.0%	0.0%	0.0%
	Std deviation	6.9%	2.8%	0.0%	4.7%
2 dimensions	Average	12.4%	2.0%	2.4%	4.4%
	Median	20.0%	0.0%	0.0%	0.0%
	Std deviation	9.7%	6.0%	6.5%	8.3%

Table 16. Percentage of devices that changed category while using EllipticEnvelope

		PCA	MDS	Isomap	FeatureAgglomeration
3 dimensions	Average	29.8%	29.0%	29.8%	29.8%
	Median	20.0%	20.0%	20.0%	20.0%
	Std deviation	19.3%	19.9%	19.3%	19.3%
2 dimensions	Average	29.8%	27.4%	29.8%	29.8%
	Median	20.0%	20.0%	20.0%	20.0%
	Std deviation	19.3%	17.4%	19.3%	19.3%

Like in clustering, run time for each dimensionality reduction algorithm was measured on the server. Execution times were measured for both three- and two-dimensional cases. Tables 17 and 18 shows execution times in seconds for each dimensionality reduction algorithm in three and two dimensions respectively.

Table 17. Time spent on dimensionality reduction with three dimensions

DR algorithm	Average time (ms)	std deviation (ms)
PCA	1.88	0.94
MDS	64.91	19.03
Isomap	5.64	2.79
FeatureAgglomeration	2.49	2.15

Table 18. Time spent on dimensionality reduction with two dimensions

DR algorithm	Average time (ms)	std deviation (ms)
PCA	1.91	0.83
MDS	56.37	17.23
Isomap	5.37	2.81
FeatureAgglomeration	2.30	1.06

7.5. Visualisation

One of the goals was to make it possible to visualise the device behaviour. Visualisation performance or usefulness was not measured as part of this thesis. Figure 8 is presented as an example of what the results look like. On the left side is user interface for selecting which algorithms to use. The right side shows the results plotted in 2d-space. Each circle is one device. Outer ring shows whether the device is malfunctioning or not. Black indicates correctly working device and pink is malfunctioning. The inner circle indicates cluster on which the device was assigned. Remember clusters have no

meaningful labels as this is unsupervised learning. The axis have no meaningful labels either as DR algorithms can transform the data. Visualisation can hopefully be used to assist in analysing clustering results.

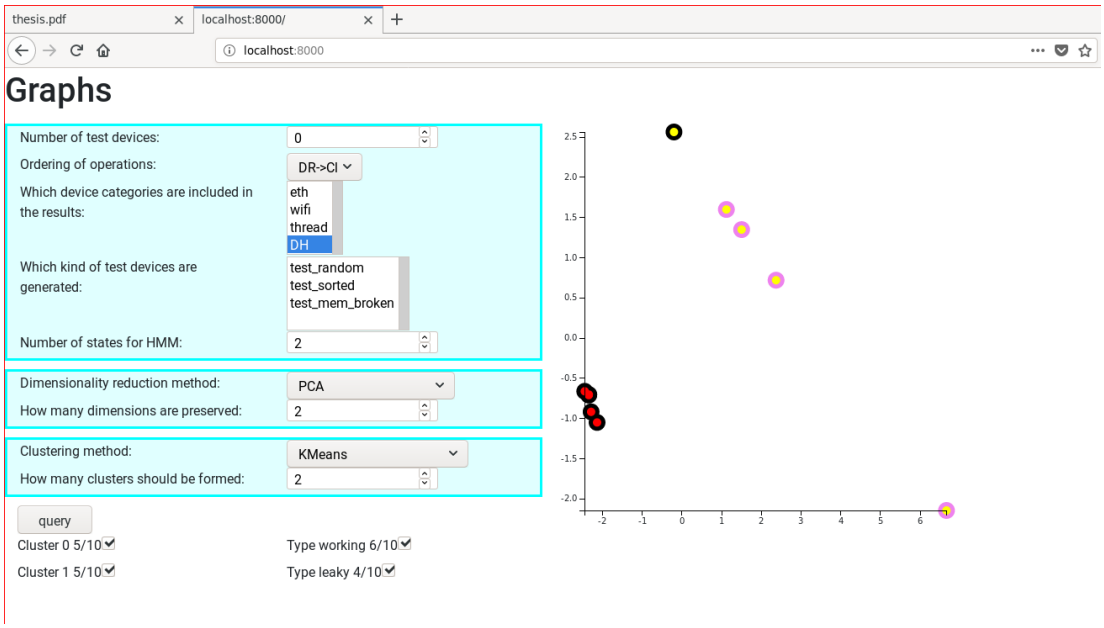

Figure 8. Example of how the results can be visualised.

8. RESULTS

Tests showed a scenario that is possible in a real world: in a set of similar devices some can differently, and the devices misbehaviour should be identified. This section analyses the findings.

8.1. Feasibility

The work was done on a basis that it would be adaptable for as many Pelion Cloud users and use cases as possible. The only thing required from the devices, apart from standard Pelion Cloud Client library, is the ability to monitor heap usage data. That might cause a penalty for battery usage on the most constrained devices. On plugged devices or devices that are charged easily that shouldn't be a problem. The rest of the software is on server side and therefore imposes no special requirements regardless the use case.

Because of constant data generation the growing database size probably restricts the service so that it's not feasible to run it automatically for all accounts.

8.2. Clustering

Different clustering algorithms were executed on the similarity data calculated with HMMs in order to separate non-working devices from correctly functioning ones. Isolation forest was the best method with this dataset with 10% median false negative and false positive rates. Local outlier factor was middle ground with a 20% median rates for both false negatives and false positives. Both isolation forest and local outlier factors were also stable having 0% standard deviation. On the other hand, Elliptic Envelope wasn't at all suitable with 30% false negative and 10% false negative rates. K-means wasn't much better with 30% median false negative rate, although median false positive rate was 0%. See tables 8 - 11 for the results.

Outlier detection is not strictly speaking clustering. If a lot of devices have similar faults, they will not be outliers, but a cluster of their own. Therefore, it's expected that outlier detection methods will perform poorly in such circumstances. This was the case with the current test setup.

8.3. Dimensionality Reduction

Dimensionality reduction was implemented to compress the similarity data calculated with HMMs into a form that can be visualised. As mentioned in Chapter 3 dimensionality reduction can be lossy. The effect must be checked to ensure that visualisations are not misleading or totally wrong. Devices were clustered with and without dimensionality reduction and the number of devices that changed a cluster between the runs was measured.

Test data shows that most combinations of dimensionality reduction and clustering changed the results quite dramatically. Isolation forest seemed most resistant to

changes with most dimensionality reduction methods changing the results only little or not at all. The other clustering methods did not perform as well.

It seems that visualising the results is possible without giving misleading picture of the situation, but it requires more careful approach. See tables 13 - 16 for the results.

8.4. Performance

Most of the execution time was spent on database related operations and event pre-processing. Forming event data series took on average over 20 seconds per request. HMM fitting and calculating similarity scores took about 4 seconds which is also a quite significant portion of total processing time. Compared to that, the rest of the execution times are negligible and well in line to be executed for each web request synchronously even on synchronous APIs. For clustering algorithms isolation forest was clearly slowest with over 260ms average execution time (see table 12). In dimensionality reduction MDS was slowest of the options but still with execution time in tens of milliseconds is not meaningful in compared to event processing times (see tables 17 and 18).

One of the hopes for dimensionality reduction step was to improve clustering performance. As clustering was already quite fast and the usefulness of dimensionality reduction is still questionable in regards to speeding up the clustering process. Due to limitations of the test data, effect dimensionality reduction on clustering performance was not measured

The event pre-processing implementation is naive and suffers from non-optimal database for this work. First of all, there are unrelated events stored such as resource value changes that are not used for this work. Used events are parsed in Python code instead of in the database which is optimised for that kind of workloads. In addition, heap usage value is stored separately from lifecycle events and corresponding values are searched once again with Python.

9. CONCLUSIONS

This chapter analyses the results from previous the chapters, discusses their significance regarding the research goal of this work and finally presents ideas on how to further develop the toolbox on from there.

9.1. Achievements

This thesis explored on how to use machine learning methods in data mining in order to separate malfunctioning IoT devices from correctly working ones. The goal was to create a toolbox for analysing the data found in Pelion ecosystem. The test setup was a group of devices where some of the devices malfunctioned.

One implementation goal for the toolbox was to make it as general as possible. The results require an additional library installed on the device but use standard communication methods available. The results can be achieved without significant overhead on devices especially if the data gathering can be combined with other device activity on resource constrained devices. Memory data was utilised only from when the device was already active so gathering that shouldn't affect battery usage or other resources significantly. Other events were gathered from cloud side and therefore could be done on any device.

Clustering results were promising. Most of the correctly working devices were found with each tested algorithm. Some malfunctioning devices were missed on each setup but this could be related to problems in the test setup. Analysis of the data was fast after the preprocessing, although scaling of the system was not measured.

9.2. Limitations

The test setup was small. Although the results were promising the statistical significance of the results is uncertain until a larger test set is used. Therefore, the result tables are only relevant for showing how larger sets can be validated. Unlike clustering, dimensionality reduction results weren't so promising. The dataset might have been too small for proper measurements or it might be that the approach is not suitable.

Assigning labels to groups requires outside information about the number of clusters or percentage of outliers in the dataset. As such the implementation is not yet usable as an automatic analysis tool since it requires outside knowledge of the dataset. Provably reliable visualisation would help if the device owner can set number of clusters or try different methods based on visualisation.

The test setup was done with two clusters only. For multiple types of problems, or heterogeneous device pool the results need further development and testing.

Current implementation does not separate devices into working and broken devices, only into differently behaving groups. Learning whether a device is malfunctioning or not requires supervised learning algorithms.

Parameters of the analysis were not decided automatically. The number of clusters and percentage of outliers were fixed. Amount of HMM states is also based on approximation of device behaviour.

9.3. Future Work

First of all, the results should be verified on a larger dataset. One option would be using Linux clients which can be scaled by an automated setup. Another way could be a simulated data set for really large-scale experiments but this approach suffers from detachment from a real-world scenario. Either way could be used to verify the results and provide clearer picture into what are the strengths or weaknesses of each algorithm used. If the results are promising there are multiple approaches. Some ideas from both research, technical and monetisation point of view are presented below.

9.3.1. Research

The number of clusters was fixed in this case. Finding out the number of clusters automatically would allow researching more diverse datasets. For example, there might be multiple different problems with devices, each showing up as a separate cluster. Similarly, outlier detection had predetermined percentage of outliers expected from the dataset. That too should be ideally found out automatically or through supervised learning to make the results useful in a generic case.

Current implementation takes into account all the events available from the device. Automatically analysing different parts of the timeline to find out if device behaviour has changed would open new insights into device life cycle. This approach could be used, for example, analysing behaviour before and after device software update or to detect hardware failures.

This work took quite exploratory research direction. Supervised learning could be used in implementing a similar service for identifying malfunctioning devices.

Current implementation uses Gaussian HMMs. Distributions of the data should be studied further.

Testing could be done without using memory data to explore whether the service would generalise to all devices connected to Pelion Cloud.

9.3.2. Technical

To demonstrate possible usage and to spark interest, the product could be added to Pelion Cloud, either as an internal or an external service. That would require either documented API and polished web user interface to interact with or preferably both. Additionally authentication would be needed.

Many Pelion Cloud APIs offer an option to filter devices. Option to pass filters available in Device Directory would allow users to select one of their device group for analysis.

Scaling of the performance was not tested. The solution might not scale to be usable with a synchronous API. Using a subset of representative devices in analysis or using random sampling should be investigated. The performance could be boosted by using purpose-built database which doesn't collect unnecessary data and stores memory usage with the event data so that querying is faster. That would also make it easier to collect data on larger set of devices. On the other, hand reading the data and fitting

the HMMs can be done in parallel for each device. This would cut preprocessing and HMM fitting times significantly. Another option is to make the API asynchronous so that long processing times would not be a problem.

The analysis could be run periodically to monitor if devices behaviour changes. This could be used to observe whether software update had undesired consequences for example. Especially with optimised database, it might be possible to, for example, gather data of devices during update campaign to see if there are changes in device behaviour.

More data sources presented in Chapter 5 could be used. One option is to build a more specific solution with careful analysis. Another option is to leave the responsibility of selecting interesting features to user.

9.3.3. Monetisation

The market for this kind of toolbox is easy to imagine. For example, if soon to be broken devices can be detected, multiple devices can be fixed by one technician on one trip instead of reactive approach where each device is fixed separately as soon as they break. If the service could be added as a part of the Pelion Cloud APIs, customers could use it to monitor their devices.

Automatic supervision of accounts could be implemented. For example, if devices start behaving differently from their customary patterns that could mean devices have been compromised.

The service implemented alongside DM like application could be deployed as part of the cloud offer or offered as a software package for users.

10. REFERENCES

[1] *Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated*, https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated, Accessed: 2017-01-02.

[2] *Mbed Cloud | IoT Device Management Platform*, https://mbed.com, Accessed: 2017-12-28.

[3] B. Scholz-Reiter, Ed., *Architecting the internet of things*. Heidelberg: Springer, 2011.

[4] *ITU-T Rec. Y.2060 (06/2012) Overview of the Internet of things*, http://handle.itu.int/11.1002/1000/11559-en?locatt=format:pdf&auth, (Accessed on 01/02/2018).

[5] C. M. Roberts, "Radio frequency identification (RFID)", *Computers & security*, vol. 25, no. 1, pp. 18–26, 2006.

[6] E. Borgia, "The Internet of Things vision: Key features, applications and open issues", *Computer Communications*, vol. 54, pp. 1–31, 2014.

[7] *Today the web was broken by countless hacked devices – your 60-second summary • The Register*, https://www.theregister.co.uk/2016/10/21/dyn_dns_ddos_explained, (Accessed on 12/06/2018).

[8] J. Mineraud *et al.*, "A gap analysis of Internet-of-Things platforms", *Computer Communications*, vol. 89-90, pp. 5–16, 2016.

[9] M. S. Mahmoud and A. A. Mohamad, "A study of efficient power consumption wireless communication techniques/modules for internet of things (IoT) applications", 2016.

[10] *Actions on Google | Actions on Google | Google Developers*, https://developers.google.com/actions/, (Accessed on 03/31/2019).

[11] *Amazon Echo, Echo Plus, and Echo Dot*, https://developer.amazon.com/echo, (Accessed on 03/31/2019).

[12] J. Han, *Data mining : concepts and techniques*, 2nd ed, M. Kamber, Ed., ser. Morgan Kaufmann series in data management systems. Amsterdam ; Boston : San Francisco, CA: Elsevier ; Morgan Kaufmann, 2006. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=193647.

[13] C. J. Pal, Ed., *Data Mining : Practical Machine Learning Tools and Techniques, Fourth Edition*, Fourth Edition. Place of publication not identified: Elsevier Science and Technology Books, Inc, 2017. [Online]. Available: http://www.books24x7.com/marc.asp?bookid=120122.

[14] k. Bishop Christopher M., *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006.

[15] R. O. Duda, *Pattern classification*, 2. ed, D. G. Stork, Ed. New York: Wiley, 2001.

[16] *scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation*, `https://scikit-learn.org/stable/`, (Accessed on 03/04/2019).

[17] M. Bicego *et al.*, "Similarity-based clustering of sequences using hidden Markov models", in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, Springer, 2003, pp. 86–95.

[18] S. Raschka, *Python machine learning : unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics*. Birmingham, UK: Packt Publishing, 2015, ISBN: 978-1-78355-513-0.

[19] *OMA Specification*, `http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf`, (Accessed on 04/03/2018).