1 2 9 0

UNIVERSIDADE D
COIMBRA

João Miguel Miranda Ferreira

IoT Based Non-Intrusive Elderly Activity Monitoring

VOLUME 1

Dissertation in Integrated Master's in Electrical and Computer Engineering, supervised by Professor Dr. Paulo José Monteiro Peixoto and presented to the Department of Electrical and Computer Engineering.

October 2021

1 2 9 0

UNIVERSIDADE Ð
COIMBRA

João Miguel Miranda Ferreira

IoT based Non-intrusive Elderly Activity Monitoring

Dissertation in Integrated Master's in Electrical and Computer Engineering, supervised by Professor Dr. Paulo José Monteiro Peixoto and presented to the Department of Electrical and Computer Engineering.

Coimbra, October 2021

"*Any fool can know. The point is to understand.*"

Albert Einstein

Acknowledgements

The conclusion of this thesis closed a five-year chapter and opened a new one with different responsibilities. These have been formidable years that allowed me to make new friendships and learn so much with so many people.

I have to express my utmost gratitude to Prof. Paulo Peixoto. When I started developing the thesis, I had no background in machine learning and his guidance helped me understand and gain enthusiasm for the topic. A special thanks to my friends for their support and joy when nothing seemed to work and all what was left to do was throw the computer out the window. Last but not least I want to thank my parents and sister for their support during the past five years.

Resumo

O envelhecimento é um processo natural, que tem tendência para reduzir a mobilidade e aumentar a probabilidade de quedas. Os mais idosos passam a ter necessidade de acompanhamento permanente ora pelos seus familiares, quem nem sempre têm disponibilidade, ora numa unidade de cuidados de saúde, que é economicamente mais exigente. De forma a proporcionar um constante acompanhamento sem necessidade de abandonar as próprias casas, esta tese propõe um sistema de monitorização para idosos baseado em sensores IoT não intrusivos.

Esta tese propõe uma arquitetura de sensores capaz de recolher eventos de um utilizador que, em conjunto com algoritmos de previsão e de análise de semelhanças de atividade, permitem detetar comportamento anormal.

A previsão do comportamento humano é uma tarefa difícil. O ser humano é imprevisível ao ponto de ser capaz de desempenhar uma mesma atividade de formas distintas apenas alterando a ordem nas interações com objetos. Esta diversidade torna dificil estabelecer o modo como atividades são desempenhadas.

Várias abordagens do estado da arte estão mais direcionadas para a identificação de atividades, enquanto esta tese pretende focar-se num método não supervisionado, ao detetar padrões no comportamento de cada utilizador, retirando a necessidade de identificar atividades. Tal será feito através de algoritmos de aprendizagem máquina e um modelo de estrutura em árvore. Para treinar os vários algoritmos, é essencial que exista um número elevado de dados, difíceis de gerar por um único utilizador.

As abordagens de aprendizagem máquina propostas exploram LSTMs, que são o mais recente estado da arte em modelação de sequências, assim como word embeddings para estabelecer correlação entre eventos. O modelo sem perdas implementado avalia o grau de semelhança de sequências de eventos e proporciona um bom método de avaliação para os modelos de aprendizagem máquina desenvolvidos. As várias arquiteturas exploradas demonstraram resultados promissores na descoberta de padrões nos dados.

Keywords: Internet das Coisas, Não Intrusivo, Aprendizagem Máquina, Monitorização de atividade, Previsão de eventos

Abstract

Ageing is a natural process that tends to reduce mobility over time, increasing the probability of falls. The elderly start to require permanent monitoring either by their families, who are not always available or by a caring facility that is economically demanding. To provide constant monitoring without leaving their houses, this thesis proposes an IoT-based non-intrusive elderly monitoring system.

The proposed approach consists of the design of a hardware architecture working side by side with activity prediction and similarity comparison algorithms to detect unusual behaviour.

Predicting human behaviour is a difficult task. The human being is very unpredictable and capable of performing the same activity in various ways just by changing the order of interaction with objects and appliances. This makes it difficult to establish how an activity is performed.

Several state-of-the-art approaches focus on identifying the activities being performed by the elderly. This thesis pretends to address a more unsupervised approach by detecting user behaviour patterns by train several machine learning approaches and a lossless model without daily activity classification. To train such algorithms, a great amount of data is required, which is difficult to acquire by a single user.

The proposed machine learning approaches explore the use of LSTM's, the current state-of-the-art in sequence modelling, and word embeddings to find event correlation. The implemented lossless model evaluates the similarity of event sequences and also provides a good evaluation metric for the machine learning approaches. The various implemented architectures have shown promising results in finding patterns in data.

Keywords: Internet of Things, Non-Intrusive, Machine Learning, Activity monitoring, Event prediction

Contents

Acronyms

ADL Activities of Daily Living. 6

CBOW Continuous Bag Of Words. ix, 10, 11

CNN Convolutional Neural Networks. 8

GRU Gated Recurrent Unit. 6

HMM Hidden Markov Model. ix, 7, 8

IoT Internet of Things. 1

LSTM Long Short-term Memory. 6, 8, 9

NLP Natural Language Processing. 9, 10, 29, 42

NN Neural Networks. 7

RNN Recurrent Neural Networks. 6

List of Figures

List of Tables

1 Introduction

Ageing is a natural process of every human being and with it comes physical and cognitive disabilities [1]. According to Peel *et al.*, the number of falls increases exponentially to people over 65 years old.

In many cases, the lack of family or availability from their families requires elderly people to either to search for a caring facility or someone to take care of them permanently at their houses. They often reject the idea of leaving their houses, not only because they have lived there for their entire lives and their house is of great value to them [1], but also because it can become very expensive.

This thesis pretends to address a way to monitor situations like this and others that otherwise could go unnoticed with the use of non-intrusive IoT sensors. Roy *et al.* [1] appeal for developers to find a solution so that the elderly can have an alternative to this situation.

One key aspect of the proposed work is the collection of data and its analysis in order to find patterns and, consequently, changes in it. This chapter will address the motivation, objectives, and the structure of this thesis.

1.1 Motivation

With the development of new technological systems comes an age that not only connects people to the internet but also devices, the Internet of Things (IoT). The already existing different devices and applications make it possible to monitor a wide range of activities inside a house and even control it if desired. In the end it all comes to which type of devices and activities to monitor and how these can be used in an advantageous way.

Above all, the one thing people value the most is their privacy. They want to feel they are not being watched, monitored. These are the main key aspects this work focuses on by using an IoT system capable of monitoring people's activities in a non-intrusive way, without them realising it with the aim of helping them whenever necessary.

This can be accomplished using non-intrusive sensors of various types, meaning, there is no use of cameras or microphones, for example. For those who still have some movement independence, this can be of great help to assist them in the event of some unfortunate situation. The early detection of such events could trigger prompt actions to mitigate the problems.

The installation of a wide range of sensors that can provide coverage of key appliances and other object interactions makes it possible to establish relationships between the events. The sequences of events that are generated, contain hidden behaviour patterns of each user that this thesis pretends to study and analyse.

The key aspect of this thesis is to take advantage of the data gathered by the sensors to find a way, a method, an algorithm that can detect a pattern and tell whether something is wrong.

1.2 Problem Formulation

There is a wide scope of sensors that can be used in a system like this. The more activities and interactions to cover, the more variety of sensors is required, which results in data heterogeneity. Finding patterns immediately suggests the use of machine learning methodologies, which sets some data requirements to develop an accurate model.

Machine Learning models are all about data. Training a model is trying to find a pattern that fits the data. This is achieved by providing a dataset to train the model, during a training task and depending on the model, this may require less or more data. The devices to be used will generate events that represent several user activities and can be grouped to define sequences of events. This way, the patterns to be found highly depend on the context the events happen, and the more the dependencies, the more data is required.

Being the events dependent on user activity the data generated will not only be discrete but also sparse, having long periods of the day without any data, like during the night for example.

To sum up, the challenges this thesis faces are the following:

1 **Data heterogeneity** - various types of sensors will result in different data from different devices

2 **Amount of data** - the training of a machine learning model requires great amount of data, difficult for a single user to produce, specially in an IoT environment;

3 **Non-equally distributed data** - time-span between events is highly dependent on user activity and can change at any moment, compromising the dataset quality

4 **Unbalanced data** - a user interacts with objects according to its needs, which might result in a dataset more expressive on some events than others

1.3 Objectives

The goals this thesis pretends to fulfill are:

1 Design a system that allows the collection of data from each users' daily activity using non-intrusive sensors

2 Find a way of representing the collected data

3 Analyse the data to filter events and generate sequences of events

4 Select, develop and implement the desired machine learning algorithms

5 Analyse the routine patterns for all the different users by training machine learning models with the generated data;

6 Evaluate the performance of the approaches developed

1.4 Structure of the Document

This document is structured in the following way:

- **State of the Art and Related Work (Chapter 2)** - Description of similar systems already proposed, the sensors used, methods and ways of overcoming the challenges faced
- **Proposed Approach (Chapter 3)** - The description of methods and planning of the work done, taking into account the state of the art and related work information
- **Proposed Hardware Architecture (Chapter 4)** - Describes all the hardware and framework used to take care of all sensors' communication and where they were placed
- **Data Analysis (Chapter 5)** - Description of the methodologies used to process the data generated by the users. Description of the non-machine learning algorithm developed to detect similarity in user sequences compared with the available dataset. Description of the machine learning algorithms implemented to detect behaviour out of the pattern
- **Experimental Results (Chapter 6)** - results of all implemented methodologies
- **Conclusion and Future Work (Chapter 7)** - final review of the results and improvement suggestions to consider in a future work

2 State of the Art and Related Work

In this chapter different alternative approaches will be analysed, exploring its advantages and disadvantages, methods used and why some of them are useful to the purpose of the work proposed in this thesis. In chapter 3 all that's referred here is taken into account to describe the approach taken.

In this thesis, the main focus will be on the use of non-intrusive IoT sensors to collect data and implement the necessary algorithms that can show themselves capable of dealing with the task. This way, apart from looking into related work, it is also crucial to find a representation that fits the heterogeneous data produced by the sensors, decide which sensors to use and which algorithms to implement.

2.1 Related Work

Bellagente *et al.* [3] proposed an architecture that uses a raspberry pi as a gateway that connects to IoT devices via Z-Wave and Bluetooth. **Motion, door, temperature** and **air quality sensors** are used as well as **emergency push buttons** and a **smartwatch** to ask for assistance and monitor heart rate. In case of emergency, the system alerts the caregiver using his/her smartphone. Medication monitoring is one of the features that include reminders in case of forgetfulness. There is no special algorithm to analyse/model data.

Pinto *et al.* [4] and Hu *et al.*[5] focused their attention to a fall and absence of vital signs detection system. The development of all the hardware was their main focus. Pinto *et al.* [4] achieved this by making their own wristband, that includes body temperature, **pressure, humidity** and **light sensors**, as well as **accelerometers** and push-buttons. Their own service board and the wireless charging dock for the watch made the system easier for the elderly to use. In [5] the system consists of temperature, pulse and accelerometer sensors.

Alcalá *et al.* [6] introduced a "Non-intrusive Load Monitoring System" with the intent of monitoring daily activities by analysing power consumption with **smart plugs**. To model pattern usage they trained a Gaussian Mixture Model which would indicate the distribution of the events for each appliance in a day. The activities were then scored based on the probabilities of an event occurring in a certain time interval.

Hao *et al.* [7] proposed an architecture tested with CASAS dataset [8] that includes motion, temperature, intrumented objects and analog sensors, that pretends to detect activities in a multi-resident home. A model was trained based on formal concept analysis (FCA) and the results were compared with a Hiden Markov Model which it did outperform.

Zhu *et al.* [9] came up with a sequence-to-sequence model built using Gated Recurrent Unit (GRU), which is a type of **Recurrent Neural Networks (RNN)** very similar to **Long Short-term Memory (LSTM)**. They used two datasets which used accelerometer sensors, contact switch sensors placed on cupboards and also on objects. These datasets are for activity recognition, in which the sets of events are labeled with the performed action. Their approach is divided into two steps: Activity reconstruction and Activities of Daily Living (ADL) recognition.

The activity reconstruction i) divides itself into three stages: i) motion state extraction, ii) data interpolation, and iii) activity state sequence. In motion state extraction i), by using a state-of-the-art model in locomotion recognition, they translate motion sensor information to motion states. In the data interpolation step ii), the discrete data is converted to continuous data by applying step-functions to the data; for the continuous values from both types of sensors, data interpolation is used. Finally, in the last step iii), the data is translated to vectors that are concatenated to be fed to the neural network. Both the motion sensor and door sensors are translated to one-hot vectors as they were semantic events.

For the ADL recognition, each column vector with the information from the sensors is fed to the encoder-decoder network. Three experiments were performed to evaluate the proposed approach.

In the first experiment, only motion sensor data was considered which was translated to motion states. Several motion state sequences were processed and fed to the model which has the task of translating those sequence events to the motion states they represented. Each sequence contained more than one motion state. The GRU approach got 71.1% accuracy and the LSTM 66.4%.

The second experiment was similar to the first one but it also took into consideration environment sensors (switch and pressure sensors). The GRU approach got 81.7% accuracy and the

LSTM 79.8%.

In both the first and second experiments, some activities got better accuracy than others, due to not having a fixed activity pattern [9].

The third and final experiment had a different approach. Instead of giving sequences with multiple sensor events that represented multiple activities, only events related to individual activities were fed to the network to evaluate the accuracy in activity labelling. The GRU approach got 77.5% accuracy and the LSTM 75.4%.

For every experiment, the results were also compared with HMM and Naive Bayes architectures which were outperformed, having accuracy values in the range 25-40%.

One problem with approaches that aim to detect daily activities is that the activities are considered to be represented by a fixed set of events. If such a system is to be scalable, this approach narrows the way activities can be performed. Each user may perform the same activity in various ways and approaches similar to this one have little flexibility.

Gueniche *et al.* proposed a more traditional sequence prediction approach. The core of the work developed is in three structures, a prediction tree, an Inverted Index and a Lookup table. They argue that this approach, contrarily to Neural Networks (NN), is lossless as the training of a NN will fail to capture the importance of some features. The prediction tree is a structure that will store every sequence. To make a prediction, given a set of events, a search will be made in the Lookup table for sequences that contain those same events. Their intersection will tell the desired sequences that the Inverted Index allows access. This way, by accessing the sequences of event it is possible to get the next set of possible events. A Count Table holds a score for each possible event and the one with the highest score is most likely the correct prediction.

Fang and Hu [11] proposed a supervised deep learning algorithm to detect human activities in a smart home. The data is also from the CASAS dataset [8]. Their deep learning algorithm consists of multiple Restricted Boltzmann Machines which did outperform both Hiden Markov Model and Naive Bayes Classifier methods.

In [12], long-duration sequence prediction is performed by monitoring movements and interaction with objects. For this, videos from youtube are used to get the movements people make while performing activities. The several actions in each video are then converted to semantic representations. Suffix trees are used to model the long and short-duration sequences and a Variable Markov Model is used to predict activities based on actions.

Alhussein *et al.* proposed a model with both Convolutional Neural Networks (CNN) and LSTM to forecast electric load. The CNN is used for feature extraction and the LSTM for sequence learning. The input of this model is a matrix that, apart from the normalized electricity consumption data, also has one-hot encoders for time, day of the week and holiday indicator.

2.2 Sensors

Many of the work described until now is very versatile in terms of sensors used. The main focus of this thesis is to make users as comfortable as they can without feeling they are being watched. This can be accomplished using non-intrusive sensors by avoiding cameras and microphones for example. From the already described work, several devices fit this profile:

- Motion sensors
- Door sensors
- Item sensors (accelerometers)
- Smart plugs
- Light Sensors
- Air quality sensors
- Temperature and humidity sensors
- Smartwatch (heart rate and accelerometer sensors)
- Emergency push buttons

2.3 Algorithms

2.3.1 Hidden Markov Model

Many of the work described so far compared their methods with Hidden Markov Model (HMM). This may suggest that this type of model could be a possible choice in prediction. On the other hand, the fact that it is outperformed several times leads to believe that there are already better algorithms for sequence prediction. Li and Fu [12] argues that the HMM is not suitable for long-term dependencies as it only models 1-order dependencies, meaning, it only makes predictions based on a single event, which isn't the best option when dealing with sequence/activity prediction.

2.3.2 RNN/LSTM

The daily routine of each one of us is no more than a collection of consecutive actions that grouped together can represent activities. The aim of the work developed is not to identify the activities being carried on but to identify if those actions, those events, match the usual pattern of the user. LSTM is the current state-of-the-art in what concerns sequence modeling.

Tax [14] worked on the prediction of human activity in a smart home environment using LSTM architectures. Three prediction approaches were taken: prediction of the next activity, prediction of the timestamp of the next activity, and prediction of a window of multiple activities. Three types of architectures were experimented, all of them LSTM based:

1 Two separate LSTM models (single task layers) to predict the next activity and timestamp of the event

2 One model with multi-task layers, which means the last layer would have two outputs

3 A mixture of the two previous models, starting with multi-task layers and finishing with single-task layers for each of the prediction tasks

The results were compared with the other sequence prediction techniques like Prediction by Partial Matching, Compact Prediction Tree [10] and Hidden Markov Model approaches. The LSTM outperforms these techniques on the prediction of next timestamp as well as the next activity. However it's not capable of outperforming on predicting a window of multiple future tasks. Multi-task models also outperformed the separated LSTM models.

In [15] a basic LSTM was also compared with Naive Bayes on the task of predicting the next activity, which it did outperform.

Long Short-term Memory (LSTM) are also used in many Natural Language Processing (NLP) applications such as sequence to sequence translation ([16] from Google) and sequence prediction, like the work above mentioned.

There are a lot of versions concerning RNN and LSTM all with the aim of improving performance. Li *et al.* [17] and Boyd *et al.* [18] are examples of it.

In [19] a comparison between LSTM architectures and other RNN architectures was done in order to assess its performance. Although LSTM didn't outperform other models in every test, they have proven to be the best in language modeling. They concluded that if there were other architectures better than LSTM they were difficult to find.

LSTM came to overcome the vanishing gradient problem, which could originate prohibitive training times or models that could model not data at all [20]. LSTM's are a set of neurons that are based on three gates: input, output and forget gate. The forget gate is used to filter what is kept and throw away from the previous hidden states. The input gate pretends to protect the memory from irrelevant inputs, by selecting which values get updated and which don't. Finally, the output gate pretends to protect other units from perturbations by deciding the next hidden state.

2.4 Event representation

Typically in a smart home, there are various types of sensors capable of covering most of the activity that takes place there. The higher the diversity of these sensors, the more difficult it is to find a common way of representing this data, making a transition from a heterogeneous representation to a homogeneous one difficult. Some of the above mentioned approaches use suffixes or prefixes to make predictions of actions based on events. By applying this same principle to sensor data, it's possible to turn each of the events into words according to the data they return. However, machine learning models are basically a series of numbers, called weights, and mathematical operations. As so, the inputs must also be numbers in order to perform all the necessary operations. In [16], for example, LSTMs are used in the field of Natural Language Processing which means all the words need to be converted into numbers, or vectors so they can be fed to the network. These vectors are called word embeddings.

2.4.1 Word Embeddings

Before word embeddings, the words were mainly represented by one-hot vectors. This was no less than a dictionary with a size equal to the number of words in both dimensions. Giving as an example a vocabulary with 5000 words, if the word "thesis" is in the index 401, the vector of word "thesis" would be a vector of size 5000 with all values set to zero except at index 401, which would be equal to one. This would happen for every word in the vocabulary. This raised many challenges when trying to find semantic and syntactic relations between words, until word embeddings appeared.

There are two most prominent techniques to train word embeddings, Word2Vec [21], by Google and GloVe [22], by Stanford.

2.4.2 Word2Vec

There are two ways of training these vectors, either by Skip-gram or Continuous Bag Of Words (CBOW). These two architectures have proven to produce high-quality word vectors even in simple architectures. One of the changes from the already existing language models of the time was the removal of a hidden layer from the architectures [21].

CBOW

CBOW learns the embeddings by predicting the word in the middle. Having a set of nine words, in a sentence, for example, the word vectors are trained by providing as input the four words that are placed before and after the middle word. The order by which these words are provided doesn't influence the training accuracy [21]. This approach works better on syntactic tasks than Skip-Gram. The architecture can be seen in Figure 2.1.

Skip-gram

This architecture works the other way round of CBOW. Instead of predicting the current words based on its surroundings, tries to predict a word based on another one. To give an example, let's imagine a set of four words in a sentence in which instead of predicting the middle word, the purpose is now to predict the surrounding words based on the centre one [21]. This architecture has a better performance on the semantic part than syntactic. The architecture can be seen in Figure 2.1.

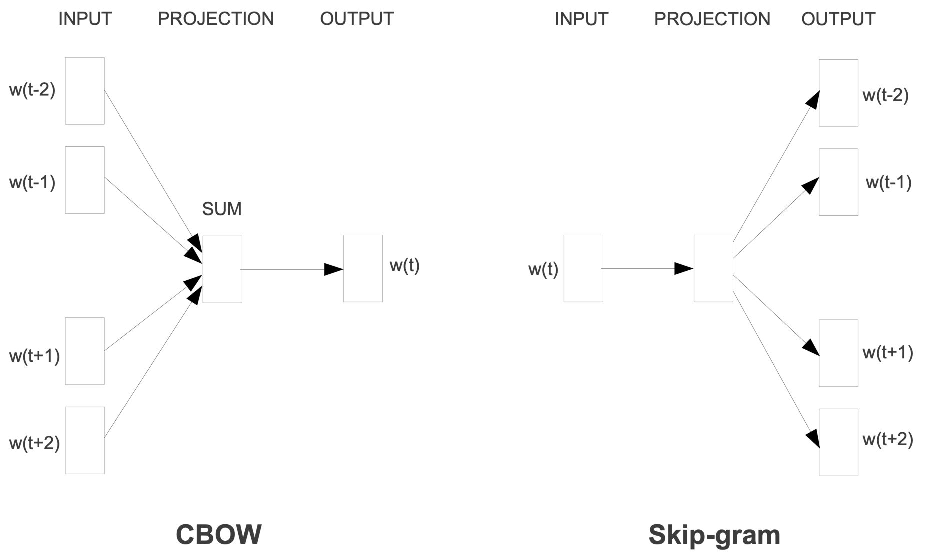


FIGURE 2.1: Architectures of CBOW and Skip-Gram [21]

2.4.3 Glove

GloVe [22] is another word embedding architecture proposed by Stanford. This method is based on a co-occurrence count matrix, which is represented by the number of times a word j occurs in the context of a word i. The probability for each word j, $P(j|i)$, can be obtained from the values of this matrix.

$$P_{ij} = P(j \mid i) = \frac{X_{ij}}{X_i} = \frac{X_{ij}}{\Sigma_k X_{ik}} \tag{2.1}$$

The difference between this model and the word2vec is the way training is performed. While in the GloVe the training is performed by analysing the whole corpus, in the word2vec, both the approaches only neighbouring words are taken into account.

3 Proposed Approach

Most of the state-of-the-art literature focuses on detecting and identifying activities. Such a method would require sequences of events to be label to specific activities, which is not only highly time-consuming but also very difficult that such activities could be equally performed by different users in case the system was scaled. To try a more generic approach, this thesis will focus on the detection of out of the context events/ event prediction. With the data collected from the use of non-intrusive sensors, several machine learning models will be trained to detect the context in which the events occur. This means that instead of knowing the activity the user is performing, we get the events that usually occur with the input given to the model. To explore other approaches, models to predict the next events will also be tested. With this in mind, the sensors were chosen to cover a wide range of possible activities that are carried in a house. The sensors are:

- Smart plugs
- Vibration Sensors

These devices will be connected to a Raspberry Pi with a Zigbee receiver, acting as a gateway. The Raspberry will receive all the data from the sensors and store it locally to be trained with a machine learning approach when enough data is available.

These sensors were placed in different key locations that the users most interact with, trying to get as much data as possible from the activities performed.

One way of detecting most home activity is by monitoring the power consumption of appliances that are frequently used. This can be achieved using smart plugs, that given are small and portable, and so they can be placed in any appliance that meets the maximum power consumption requirements.

Due to the challenges faced by this thesis, in the beginning, it took us some time to figure out which could be the best machine learning approach. Until the decision was made, a more traditional non-machine learning model was built to make some sequence similarity analysis in real-time. The approach proposed by [10] was implemented with a few adjustments that will be further discussed.

Due to the existence of heterogeneity in the collected data, it was necessary to find a way to standardise the data in order to train the model. As so, each of the events will be first translated to words, having different names depending on the appliances and states (provided by the consumption values). In order to train the machine learning model, these word events should be converted to vectors through the training of word embeddings.

The first implemented traditional method was later used to try and generate more data hopping to improve the accuracy of the implemented machine learning algorithm.

Taking all the state of the art and related work into consideration, this was the approach designed to try to fulfil the objectives and challenges described in chapter 1.

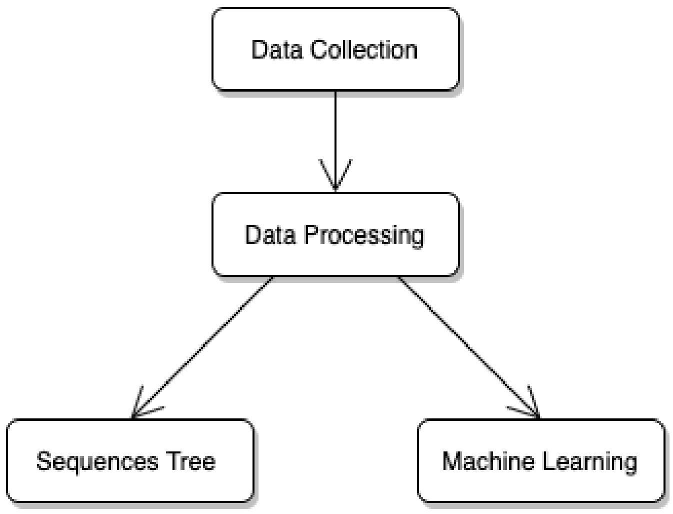


FIGURE 3.1: General Pipeline of the proposed approach

3.1 Tools

Python has a great amount of libraries that simplify data analysis, which came in really handy with the JSON messages from the sensors. The entire algorithm was developed in python, as well as the machine learning approach with PyTorch. To fine tune the hyper-parameters of the models the API "Weights and Bias" was used to automatically test the model to a variety of values in order to check which were the parameters that could provide the best model performance.

The traditional method was developed in C++ as it is a performs much faster due to being a compiled language.

4 Proposed Hardware Architecture

4.1 Sensors

As previously mentioned, two types of sensors were used, vibration sensors and smart plugs. The vibration sensors consist of accelerometers, capable of detecting sudden movements like opening drawers or doors. The smart plugs can provide us the instant power values, from which is possible to detect ON and OFF states of an appliance.

Other types of sensors were considered like motion and temperature/humidity sensors. Motion sensors could be used to provide the location of the user in the house and the humidity/temperature sensors could be used to detect the steam when having a shower, where the environment is more propitious to falls.

The sensors were placed in a house where four people live. Given this environment, the location of a user using motion sensors would be pointless for obvious reasons. In an environment like this, it's most appropriate to group and analyse the sensors by room so that the activities from different users can be better distinguished. As so, if the temperature/humidity sensor was to be the only sensor in the bathroom, no event correlation inside this room would exist as it would be the only device available in that room.

For these reasons, the decision was to focus mainly on those two sensors and in one particular room, the kitchen. There are vibration sensors (VS) and smart plugs (SP) installed in:

1 Fridge door (VS)

2 Coffee cups' cupboard (VS)

3 Trash bin door (VS)

4 Pans door (VS)

5 Kitchen tools drawer (VS)

6 Herbs and spices cupboard

7 Cutlery drawer (VS)

8 Platters drawer (VS)

9 Dishes drawer (VS)

10 Glasses cupboard (VS)

11 Oven door (VS)

12 Coffee Machine (SP)

13 Toaster (SP)

14 Sandwich Maker (SP)

15 Cooking robot (SP)

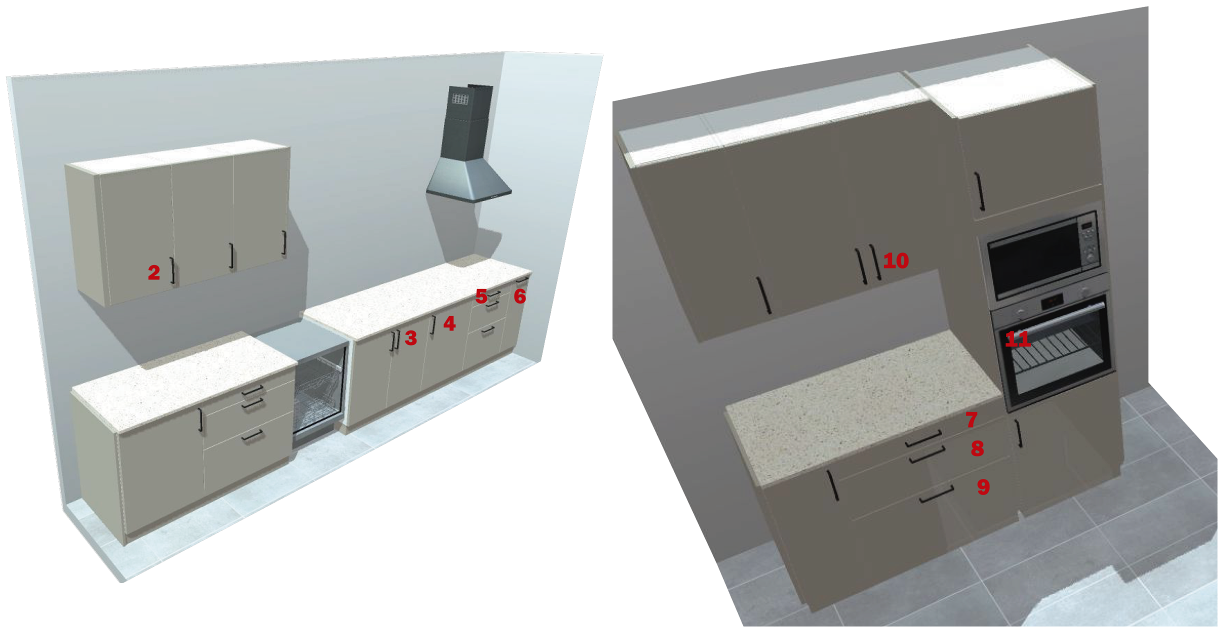


FIGURE 4.1: 3D sketch of the kitchen with numbers indicating the location of the vibration sensors

In the living room, two smart plugs were also installed in the television and reading lamp. The last smart plug was installed in the bedroom, where the bedside lamp was connected. The data collected from these sensors was used in the traditional approach although it wasn't used to train the machine learning model due to the lack of event dependencies and therefore context. In these cases probably a simpler method like measure the mean usability time or the number of occurrences during the day would be enough, if not better.

4.2 Gateway

All these sensors worked with the Zigbee specification and therefore needed a gateway to be connected to. At the centre of this was a Raspberry Pi to which was connected a Zigbee receiver and a hard drive to store the data. Due to the limited range of Zigbee devices and the existence of walls and furniture, there were some connection problems between the sensors and the gateway. To avoid the loss of data, another Zigbee receiver was installed in the kitchen to route the information to the main gateway. As this receiver only needed a power supply, it was directly connected to a USB phone charger. To make this all work, a framework called Zigbee2MQTT [23] was used. This framework allows the connection of a variety of sensors having access to the data via MQTT and log txt files of every sensor event. After pairing all the devices to the Raspberry Pi, the configuration file was edited in order to separate the devices into groups, which were named according to the rooms they were in. This configuration file was then provided to the data analysis algorithm so that it could easily and correctly identify the events. A schematic of the system architecture is represented in figure 4.2

The generated log files with all the events were analysed to get the sequences of events representative of the user activity. This analysis is described in chapter 5.

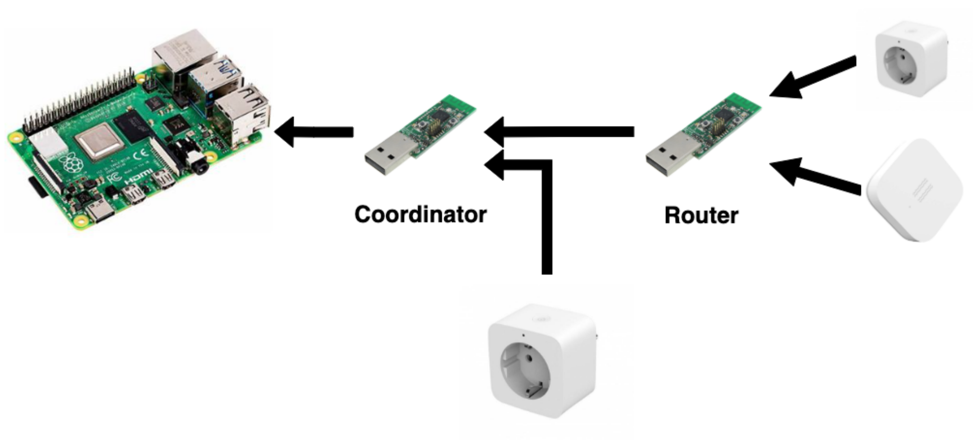


FIGURE 4.2: Hardware Schematic

MQTT is a messaging protocol capable of communicating even in a limited bandwidth scenario. It is built on top of TCP/IP protocol which guaranties the delivery of messages. The publish-subscribe pattern allows for every subscriber of a topic to get the message published, even remotely, making it suitable for an IoT monitoring system. This feature allows real-time data analysis and will be taken to advantage in the proposed system.

4.3 Problems faced

To train machine learning models one of the things that is of most importance is to have as much data as possible. However, some problems emerged that ended up in the loss of data. First of all, this thesis was developed in the context of the Covid-19 pandemic. The purchase of the sensors got affected by several delays due to lockdown periods that only allowed the devices to arrive three and half months after starting working on the thesis.

Apart from the delay, there was a limit to the amount of log information the framework can save. The log files are generated by folders that are named with the date and time the framework starts logging after an interruption. So, if the raspberry pi is running uninterruptedly only one folder is created that can store up to three log files. After this file limit is reached, the following files will overwrite the existing ones. This wasn't referred anywhere in the website, or any help forums, resulting in the loss of one and a half months of data. After this limitation was discovered, backups started to be made every week.

Considering all the delays, until the end of the thesis, only three months of data were successfully collected from a total of eight months.

5 Data Analysis

As introduced in chapter 1, there are several challenges this thesis faces: data heterogeneity, low amount of available data and the unpredictability of the user.

When training machine learning algorithms, particular care should be put on the training data. All the features need to have the same structure. Although, as mentioned, one of the problems is precisely data heterogeneity, which means that as each sensor has its own parameters, there is no data structure on which they can all rely. Finding a common data type or structure that could be applied to every sensor data became a priority, not only for the machine learning approach but also to make the traditional method simpler.

To overcome this challenge the solution that ended up being implemented was converting all the data events to semantic representations. This means that the values returned by the sensors were converted to words capable of having a meaning and purpose for the problem in hands. The parameters that were of great interest were the power consumption for the smart plugs and the trigger action for the vibration sensors.

5.1 Power Consumption

The smart plugs have the capacity of measuring instant power consumption. For each appliance, after collecting a wide range of data for several weeks, the minimum, maximum, mean and standard deviation of all the consumption values were calculated for each appliance. The goal was to try finding a way of getting the ON/OFF thresholds automatically just by analysing the data. For some of the appliances, the calculated values were perfectly fine but for others, like the coffee machine, the thresholds were causing too many ON/OFF states. These devices don't have a fixed power consumption, the values change according to its needs. For example, the coffee machine starts working from the moment it starts grinding the coffee beans, although, the power consumption at this stage is much lower than when it needs to pump and heat the water. The same happens with the television. The power consumption changes with the image colours as well as the changing of a channel for example, which can lead to the detecting of false ON and OFF states. To tackle this issue, all the values were evaluated and some of the devices, these two included, got their thresholds overwritten by more suitable values in order to reduce false events and obtain a more balanced dataset.

5.2 Trigger action

The vibration sensors return a set of position parameters as well as three possible action values: tilt, drop and vibration. These action values represent the action that triggered the event. The values themselves are not relevant to the thesis but as the vibration sensor sends regular messages, even if no interaction with the user happened, these action values allow distinguishing the triggered events from the others. However in future work, due to their versatility and small size, these sensors could have a wide range of use cases in which these parameters could come in really handy.

5.3 Analysis by room

The analysis of the sequences is divided into rooms. This decision came from the fact that in this particular environment, in which the devices were installed in a house where four people live, there would be occasions in which two or more people would be interacting with different sensors in different locations at the same time. Logically, these couldn't be considered of the same user sequence as they were not performed by the same person.

5.4 Pipeline

All the data generated by the sensors is logged into a txt file by the Zigbee2MQTT framework [23]. Many of the log information doesn't represent event data or if it does, it is not an event generated by user interaction. It is required a thorough data analysis to identify the events triggered by the user and the sequences they represent. A simpler diagram is presented in figure 5.1 followed by an explanation of each step's.

1 Read every line of the log file. Only those that represent events (with and without user interaction) are considered and stored. Values like time and the room the event took place, are stored for further analysis.

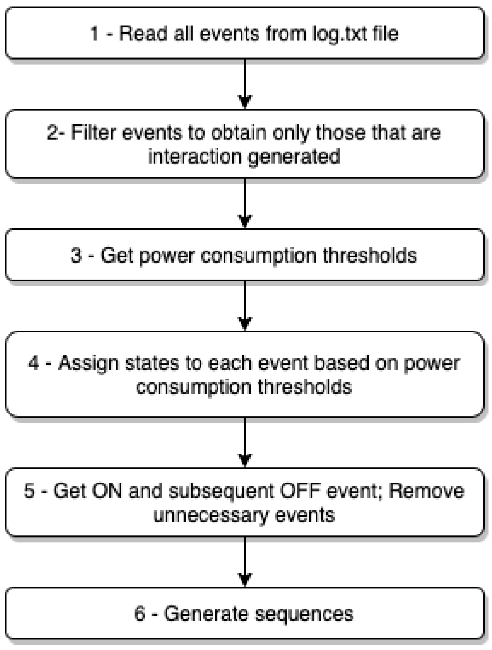

1 - Read all events from log.txt file
2- Filter events to obtain only those that are interaction generated
3 - Get power consumption thresholds
4 - Assign states to each event based on power consumption thresholds
5 - Get ON and subsequent OFF event; Remove unnecessary events
6 - Generate sequences

FIGURE 5.1: Data analysis pipeline

2 As the current list of events includes every event, it is essential to remove those that weren't generated by the user. As only smart plugs and vibration sensors are being used, the important parameters are power and action, respectively. All the events that have values in these two parameters are attributed a device type and all the others are removed from the dataset. To recall, if the vibration sensor events have a value in the action parameter they represent an event by interaction.

3 Calculate the mean and standard deviation of the power consumption for each appliance in order to automatically define power consumption thresholds.

4 With the consumption thresholds assign a state to every event in the dataset. These can be of 3 types: ON, OFF and PING. The last state applies if it's an event from a vibration sensor, as it has no duration.

5 Search for each ON and subsequent OFF event of the same appliance and calculate the time the appliance was being used and remove all the unnecessary ON and OFF events detected in the middle: remove all the ON events between an ON state until the next OFF event; remove all the OFF events between an OFF event until the next ON appears.

6 Finally, generate the sequences by aggregating the different events taking into consideration a time limit after which a new sequence is generated. This time limit was set to 10 minutes, which was considered a reasonable value for a user to perform an action, without sensor coverage, between two events, to allow the sequence to continue. The time limit is only considered after the last expected OFF event, which means that it is ignored until the completion of an ON event.

5.5 Event vocabulary

As described, there are three possible states, ON, OFF and PING. The first two states belong to the smart plugs and depend on the power consumption. The last one is the only state that a vibration sensor possesses. As they only report the action and not the total time the interaction took place PING was the chosen word to address this situation. As so, the possible generated event vocabulary list is the following:

- coffeeMachineON/ coffeeMachineOFF
- sandwichMakerON/ sandwichMakerOFF
- toasterON/ toasterOFF
- cookingRobotON/ cookingRobotOFF
- microwaveON/ microwaveOFF
- dishesPING
- glassesPING
- plattersPING
- cutleryPING
- spicesPING
- pansPING
- trashPING
- fridgePING
- ovenPING
- toolsPING
- coffeeCupsPING

These are the events that will make part of each sequence generated by the data analysis algorithm. Those will be provided to both the traditional and machine learning methods that will rely on them in order to obtain the desired performance. Additionally, for the traditional method also the events that take place in the living room and the bedroom are considered as it is a lossless approach that doesn't use machine learning and thus the amount of data and event correlation is not critical. The following vocabulary is also possible:

- televisionON/ televisionOFF
- readingLampON/ readingLampOFF
- bedsideLampON/ bedsideLampOFF

5.6 Data collected

As already mentioned in chapter 4, there were problems concerning the data collection that caused the only allowed for the collection of three months of data.

From these three months several data was gathered. The data statistics is represented in Table 5.1

	No. Events	**%**
Coffee Machine	2080	8.38%
Cutlery	2356	9.50%
Toaster	530	2.14%
Sandwich Maker	144	0.58%
Cooking Robot	438	1.77%
Microwave	724	2.92%
Fridge	5208	20.99%
Oven	943	3.80%
Herbs and Spices	1450	5.84%
Glasses	1356	5.47%
Tools	272	1.10%
Dishes	1454	5.86%
Pans	2358	9.50%
Coffee Cups	1040	4.19%
Trash	1809	7.29%
Platters	2649	10.68%
Total:	24811	
No. Sequences generated:	3114	
No. Days:	80	

Table 5.1: Statistics of the data collected for 3 months

Analysing the dataset it is noticeable that it's not accurately balanced as there are events that have a higher number of occurrences than others. The fridge is the event with the highest number of occurrences, occupying 22% of the dataset and on the other end are events from the sandwich maker with only 0,5% of the dataset. Such an unbalanced dataset can highly influence the results when training a neural network model, although it's normal that the users interact more with some objects than with others. This situation emphasizes the challenges introduced in chapter 1.

5.7 Sequences Tree

While still trying to find the best machine learning approach possible to detect the pattern in the user activity, a more traditional non-machine learning approach was implemented to serve as baseline method.

This algorithm is truly based on the sequences generated on the data analysis procedure. To briefly explain how it all works, it consists of a prefix tree that stores all these sequences and

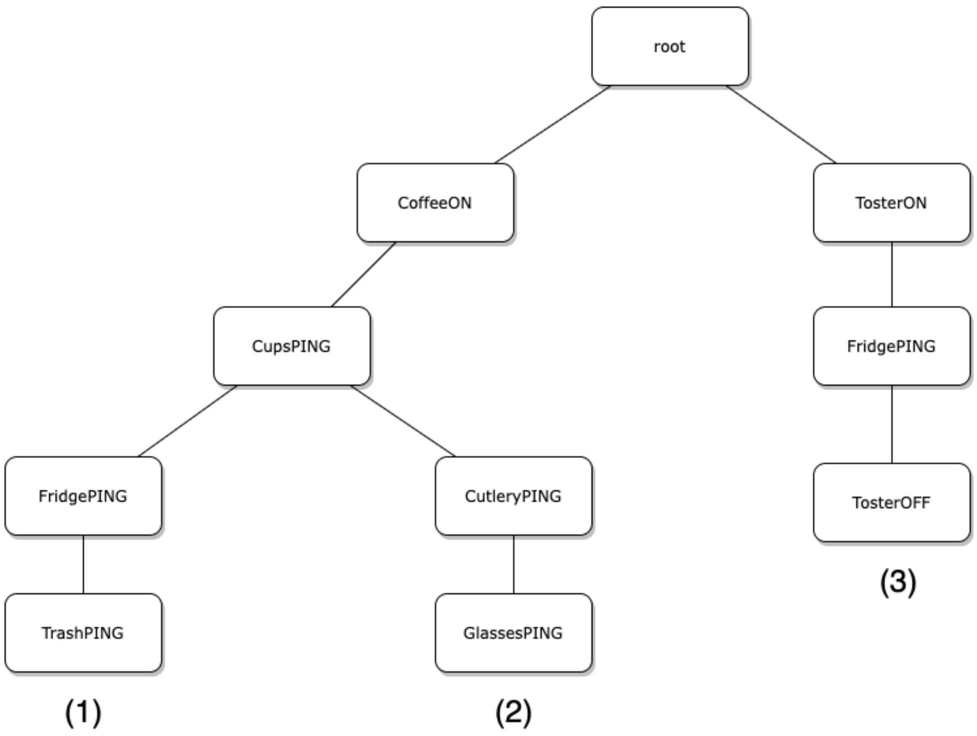


Figure 5.2: Prefix Tree

analyses the similarity of a user sequence by getting the most similar sequences stored in the tree, comparing it with all of them by matching the subsequences both contain.

5.7.1 Main Structures

Being a traditional non-machine learning model, there is a lot to take into account and it comes as a cost of many data structures. The following structures are the ones in charge of storing the analysed sequences on which any other structure will rely to get the necessary data.

Sequences Tree

This is the main structure that stores the sequences by which the similarity will be evaluated. After the data analysis described in chapter 5, the data is added to this structure. A file containing all the sequences is read and the prefix tree is created. This tree avoids redundancy by taking to its advantage equal parts of different sequences. For example, if two sequences have three events at the beginning that are the same, only three nodes will be created that will be common for both the sequences. On the third node, the sequences diverge resulting in two children for this node. This only works if equal events are placed at the beginning of sequences. The address of the last node of the sequence is stored in a lookup table, making it possible to access a given sequence without iterating through the whole tree.

Lookup table

The sub sequence similarity algorithm requires sequences with which the comparison can be performed and therefore it is essential a quick access to a sequence stored in the tree. The lookup table stores the address of the last node of each sequence in the prefix tree, with a different ID making it possible to obtain access to a particular series of events. Taking the example in figure 5.10, for each of the sequences (1), (2) and (3) there would be an entry that has the address for the last node of the sequence.

Occurrences Index

Each event is registered in the occurrences index. This structure stores the ID of every sequence where that particular event appears. When a user sequence arrives to be compared with the ones existing in the prefix tree, one of the steps is to get all the sequence IDs of every event and get their intersection. The remaining IDs are the ones whose sequences have all the events in common. These IDs are represented by the numbers below each sequence in figure 5.10.

5.7.2 Similarity Algorithm

Several structures are used to process the algorithm. For each room, in this case, bedroom, living room and kitchen, exists a structure that takes place in that room. This means that the similarity analysis is performed independently for each room.

The framework used to collect the data from the sensors [23] is based in MQTT. This feature allowed the reception of events in real-time, by subscribing to a topic. Whenever an event is received, a set of tasks is performed in order to evaluate the similarity of the sequence received until that moment:

1 Receive event JSON message and process the information to check if it's an interaction event and in that case if the event is expected. An event is expected if it is a PING, an OFF event after an ON event or a OFF event after an ON event, for each appliance. NOTE: after an ON/OFF event, repeated events may arrive with similar information, which is not desirable to consider as part of a user sequence.

2 Search the occurrences index for the sequences ID where the events have taken place

3 Get the intersection of all the sequence IDs to find the only sequences that match all the events

4 Get the address of the last node of each matched sequence and iterate through each one of them to get the whole set of events

5 Evaluate the user sequence for each matched tree sequence by analysing the subsequence similarity, meaning, searching for sets of following events in both sequences

6 Whenever a new event arrives, it is added to the vector the stores the user sequence currently active

7 If after the last PING or OFF event, a 10 minute interval has elapsed, a new sequence will be generated

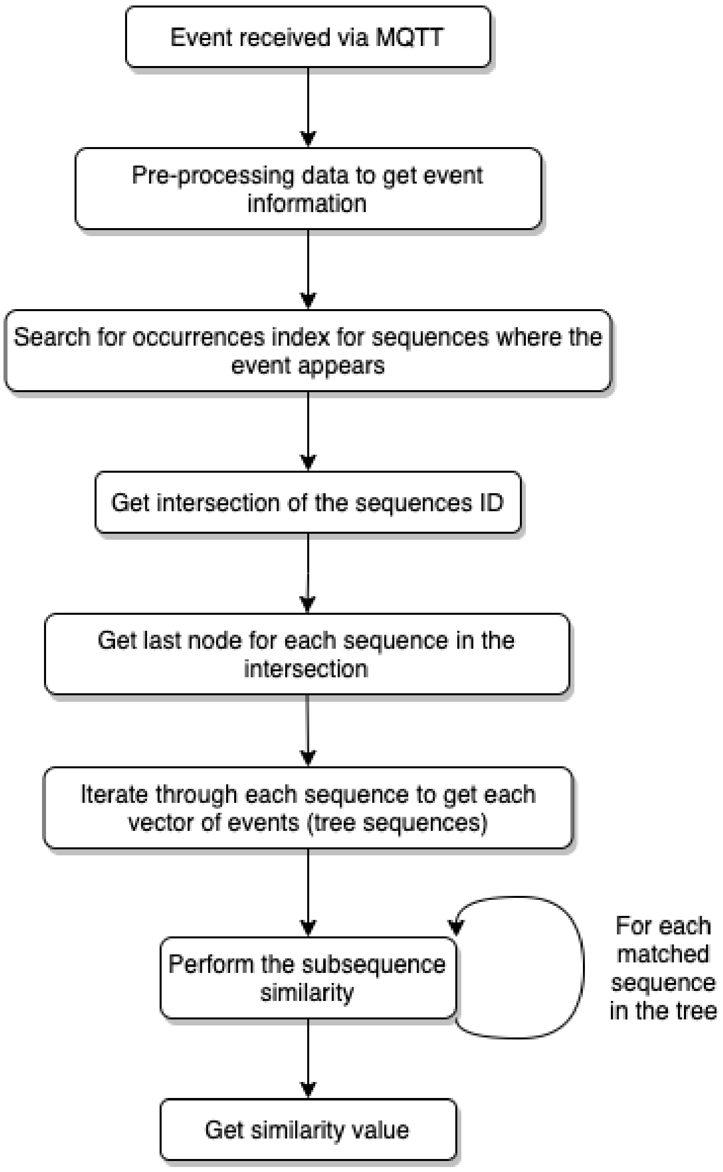

Event received via MQTT
Pre-processing data to get event information
Search for occurrences index for sequences where the event appears
Get intersection of the sequences ID
Get last node for each sequence in the intersection
Iterate through each sequence to get each vector of events (tree sequences)
Perform the subsequence similarity
For each matched sequence in the tree
Get similarity value

FIGURE 5.3: Pipeline of the traditional approach algorithm

Subsequence similarity

Let's consider a user sequence (US) of size m and a tree sequence (TS) of size n. They will form a matrix of size $m \times n$ which will be the voting system of this similarity comparison.

All the values of the matrix are initialised to zero. By iterating through each of the sequence vectors, a comparison between each of the selected events will be made. If both the events at position i and j are the same, the value in the matrix at location $[i][j]$ will be equal to the sum of one to the value of the its upper diagonal value, meaning, $matrix[i][j] = matrix[i-1][j-1]+1$. If this condition doesn't verify, the value remains the same, which means, zero.

If there is a match of a subsequence between the user and tree sequences, a diagonal of incremental values will exist (Table 5.4).

		Tree Sequence						
		ToasterON	FridgePING	CoffeeON	CupsPING	ToasterOFF	CoffeeOFF	DishesPING
User Sequence	ToasterON	1	0	0	0	0	0	0
	FridgePING	0	2	0	0	0	0	0
	CoffeeON	0	0	3	0	0	0	0
	CupsPING	0	0	0	4	0	0	0
	CoffeeOFF	0	0	0	0	0	1	0
	DishesPING	0	0	0	0	0	0	2
	ToasterOFF	0	0	0	0	1	0	0

TABLE 5.2: Subsequence similarity: The orange diagonal represents an existing subsequence between the user sequence and one of matched sequences store in the prefix tree. The numbers in yellow represent single event matches

		Tree Sequence							
		FridgePING	CoffeeON	CupsPING	ToasterON	FridgePING	CoffeeON	CupsPING	ToasterOFF
User Sequence	ToasterON	0	0	0	1	0	0	0	0
	FridgePING	1	0	0	0	2	0	0	0
	CoffeeON	0	2	0	0	0	3	0	0
	CupsPING	0	0	3	0	0	0	4	0
	ToasterOFF	0	0	0	0	0	0	0	5
	DishesPING	0	0	0	0	0	0	0	0

TABLE 5.3: Subsequence similarity: The darker line outlines the repeated subsequence events. To avoid double counting, only the orange subsequence, that start first, is considered

The higher the number of subsequences found in the matrix, the higher is the similarity. Although, if a subsequence repeats itself more than once along the tree sequence, there will be more than one diagonal in the same lines (Table 5.5). As the algorithm is supposed to compare the similarity of the user sequence with the already registered ones in the tree, this isn't a desirable behaviour. This would cause the algorithm to double count the same subsequence in the user sequence. If two or more diagonals overlap, the algorithm will only consider the one that starts first in the similarity equation. Let K be the sum of the sizes of all subsequences found, the similarity equation is as follows:

$$\textit{Similarity} = \frac{2 \times K}{\text{size (user sequence)} \ + \ \text{size (tree sequence)}} \tag{5.1}$$

This evaluates the user sequence in two different ways:

1 Considers all the common events, generating a general similarity ratio

2 Only considers sub-sequences (with size greater than 1) and, in this case, the order by which the events occur matters

The values returned by the similarity function can have different meanings:

- If **Similatiry = 0** the user sequence has no subsequence matches with the tree sequence
- If **0 < Similarity < 1** the user sequence has that amount of similarity with the tree sequence
- If **Similarity = 1** the user sequence and the tree sequence are a perfect match
- If **Similarity > 1** the user sequence and the tree sequence have subsequence match but the user sequence is already bigger than the tree sequence and there are matches of the tree sequence that appear more than once in the user sequence

5.7.3 Challenges

The human being is capable of performing the same activity in an infinite number of ways. In this particular case, it means that the same activity, let's say lunch, can have a wide range of possible events, order, length, duration, and so on. The unpredictability of the human being makes this method, not the best approach as it can only work based on the stored information, not having the ability to generalise for unseen scenarios. This is a role where machine learning models can have a better performance. Those methods will be explored in the next chapter.

5.8 Machine Learning

There are various ways and architectures to deal with the task at hand. All of them were evaluated and some of them were tested in order to evaluate which one could better identify the events and the context in which they happen with the best accuracy possible. The following sections will describe all the steps taken until the final result and why some types of architectures perform better than others. But before that, there is something that all of the models have in common, which is the dataset.

5.8.1 Dataset

There are various challenges to overcome when training a neural network. In this case, as already mentioned, the dataset can introduce a high level of bias into the model. As mentioned in the Introduction, there are three main challenges that can influence the results: (1) the amount of data, (2) an unbalanced dataset and (3) data heterogeneity.

The last of the mentioned challenges was mitigated when the events were coded as words. From the data analysis described in chapter 5, the result is a file with sequences of events generated by the user when performing his daily activities. These sequences are like sentences in a text in which the text is the whole set of sequences available. Most of the NLP models that exist are trained with large datasets from websites like "Wikipedia" that are capable of providing text corpus with millions of words, sentences in which there is a logic to the order in which they appear, there is a context which makes it possible to find patterns and define rules to model sequence data.

However, all the data collected through the sensors comes not only from a restricted dataset but the order in which these events appear in a sequence doesn't follow a specific set of rules. To better explain, lets imagine a user is taking his breakfast and generating the following sequence:

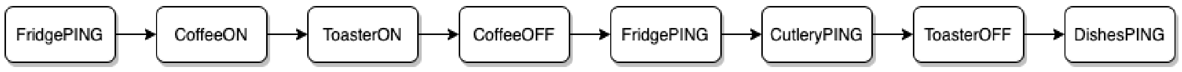


FIGURE 5.4: Example of a sequence of events representing a user having breakfast

This is possible to understand that very probably breakfast is being prepared. Now lets change the order of these events:

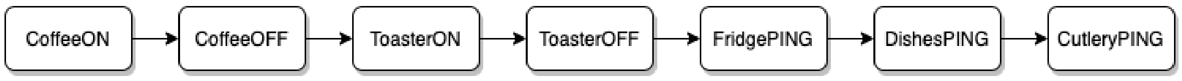


FIGURE 5.5: Example of another sequence of events of a user taking his breakfast that contain the same events but in a different order

This sequence of events represents exactly the same activity, although the order of the events has changed. It's possible to noticed that for every ON event there is always an OFF event, but apart from that, the order could be any.

In a normal sentence, from a book for example, there are nouns, verbs, adjectives and so on. The order in which those words appear has to follow a certain set of rules, otherwise the sentence would make no sense. With this type of events there is always space for randomness, which is not common in a corpus of text. The small size of the dataset, with the already presented twenty seven words, may make it difficult to find relations between them as the context in which some of the appliances are used can change drastically and the sequence still makes sense.

To try to find whether this may influence accuracy, several model architectures were tested. But one of the things all models have in common is the word representation.

5.8.2 Word Embeddings

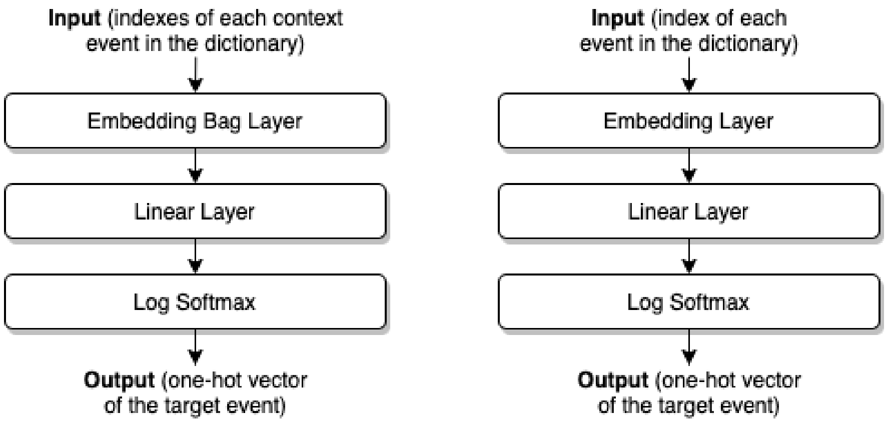
As introduced, there are two state-of-the-art word embeddings architectures, Word2Vec (CBOW and Skip-gram) and Glove. Each of them requires different input shapes thus they will be addressed separately. The initial sequence recognition is equal in every approach. In a first step, the CSV file generated by the data analysis algorithm is read. This file contains all the events divided into sequences identified by a sequence ID. To train the different architectures it's probably a better approach do deal with each sequence individually and therefore these events need to be aggregated into sequences, which means turning a dataset of events into a dataset of sequences.

Algorithm 1 Data setup

```
1: $data \leftarrow$ read data from file
2: $dataset \leftarrow$ create empty list to store sequence dataset
3: $aux\_sequence \leftarrow$ auxiliar structure to temporarily store a sequence of events
4: $sequenceID = 0$
5: for $i < length(data)$ do
6:     if $sequenceID == data[i].sequenceID$ then
7:         $aux\_sequence \leftarrow$ append current event data[i]
8:     else
9:         $dataset \leftarrow$ append aux_sequence
10:        sequenceID = data[i].sequenceID
11:        $aux\_sequence \leftarrow clear$
12:        $aux\_sequence \leftarrow$ append current event data[i]
13:    end if
14: end for
```

Word2Vec

Both the architectures used to train the Skip-gram and CBOW are very simple. They consist only of a hidden layer, that contains the embeddings which will then be fed to a linear layer, in order to convert the embeddings to an output with the size of the event vocabulary and finally a log softmax layer. For each event received as input, the embedding layer returns the corresponding embedding vector that will go through the model. The CBOW model receives as input a number of events according to the chosen window size. In order to be compatible with the architecture, all the embedding vectors are averaged to obtain only one vector. The embedding bag layer does this procedure automatically.


Input (indexes of each context event in the dictionary)
Embedding Bag Layer
Linear Layer
Log Softmax
Output (one-hot vector of the target event)
Input (index of each event in the dictionary)
Embedding Layer
Linear Layer
Log Softmax
Output (one-hot vector of the target event)

(A) CBOW model architecture (B) Skip-gram model architecture

FIGURE 5.6: Word2vec model architectures

Skip-gram

In Skip-gram the word embeddings are trained with the aim of learning the context in which the words appear. By giving an event to the model as input it would return the words that usually occur near this one.

After having the dataset divided into sequences, the next step is to create the inputs and labels to train the model.

Skip-gram architectures receive one input and can return many others, the contexts. This way, by defining a context window, let's say of two events, the event in the middle will be our input and two events on each side of the middle event will be the context of this word, creating pairs of data (**input**=middle event, **target**=context events). This is exemplified in figure 5.15.

The red box in figure 5.15 represents the context window of size two for the centre word "CoffeeOFF". In this example the data generated to train would be the pairs (CoffeeOFF, CoffeeON), (CoffeeOFF, ToasterON), (CoffeeOFF, FridgePING) and (CoffeeOFF, CutleryPING).

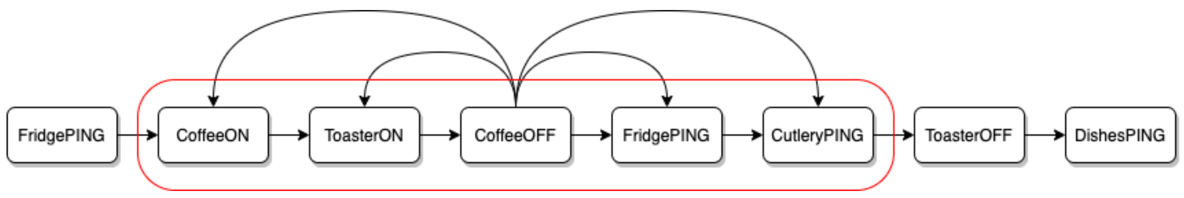


FIGURE 5.7: How to create the dataset to train Skip-gram architecture

The algorithm to take care of processing the data is the following:

Algorithm 2 Skip-gram Data setup

```
1:  $dataset \leftarrow$ stores the event sequences
2:  $CONTEXT\_SIZE \leftarrow$ size of the window of events
3:  $x \leftarrow$ stores model input
4:  $y \leftarrow$ stores model label
5:  for $i < length(dataset)$ do
6:      for $j < length(dataset[i]) - CONTEXT\_SIZE$ do
7:          for $a \leftarrow -CONTEXT\_SIZE <= CONTEXT\_SIZE$ do
8:              if $a! = 0$ then
9:                  $x \leftarrow$ appends dataset[i][j+a]
10:                 $y \leftarrow$ appends dataset[i][j]
11:             end if
12:         end for
13:     end for
14: end for
```

CBOW

The data processing for the CBOW model is the other way round of the Skip-gram in which the input of the network is the sequence of events that give context to the middle word, and the middle work is the target. One advantage of this approach over the previous one is that the training is much faster as the inputs are in a smaller number. This way, going back to figure for the CBOW approach the data generated to train the model would be a single structure, for the red box, following the same shape, (input, target). It would be like ([CoffeeON, ToasterON, FridgePING, CutleryPING], CoffeeOFF).

GloVe

The input data in GloVe architecture follows a different pattern. As specified in chapter 1 this word embeddings architecture isn't a window-based approach as it learns the embeddings from the entire corpus which consists of all the sequences displayed like a text. The problem with this architecture is that it requires a great amount of data that a single user is incapable of producing. After trying to train the glove embeddings, no results were achieved.

Algorithm 3 CBOW Data setup

```
 1: $dataset \leftarrow$ stores the event sequences
 2: $CONTEXT\_SIZE \leftarrow$ size of the window of events
 3: $x \leftarrow$ stores model input
 4: $y \leftarrow$ stores model label
 5: for $i < length(dataset)$ do
 6:     for $j < length(dataset[i]) - CONTEXT\_SIZE$ do
 7:         for $a \leftarrow -CONTEXT\_SIZE <= CONTEXT\_SIZE$ do
 8:             if $a! = 0$ then
 9:                 $aux\_x \leftarrow$ appends dataset[i][j+a]
10:             else
11:                 $y \leftarrow$ appends dataset[i][j]
12:             end if
13:         end for
14:         $x \leftarrow$ append context events
15:     end for
16: end for
```

5.8.3 Event Prediction

LSTM is the current state-of-the-art in sequence modelling. Trying to predict events in a scenario where there is not wrong order of events, as there is with common language sentences, makes this task difficult to implement. Nevertheless, taking advantage of the correlations of word found training word embeddings, a transfer learning technique will be applied to try predicting the next events with the trained embeddings in a separate model.

The architecture represented in figure 5.16 receives in the embedding layer the indexes of each word event and outputs the corresponding embedding vectors. They will be fed to three sequence LSTM layers and at the end a linear, sigmoid and softmax layers which will output a vector with the size of the vocabulary with the prediction being the index of the highest value.

5.8.4 Hyperparameter fine-tuning

There are parameters that can widely affect the performance of the final model. Parameters like batch size, number of epochs, context size for the training of word embeddings, size of the embeddings, size of the hidden layers, and so on. The number of combinations of all these parameters is considerable and testing them all by hand would be unprofitable and time-consuming. "Weights and Biases" [24] was therefore responsible for running several combinations of parameters and logging them to a website where the information could be examined afterwards.

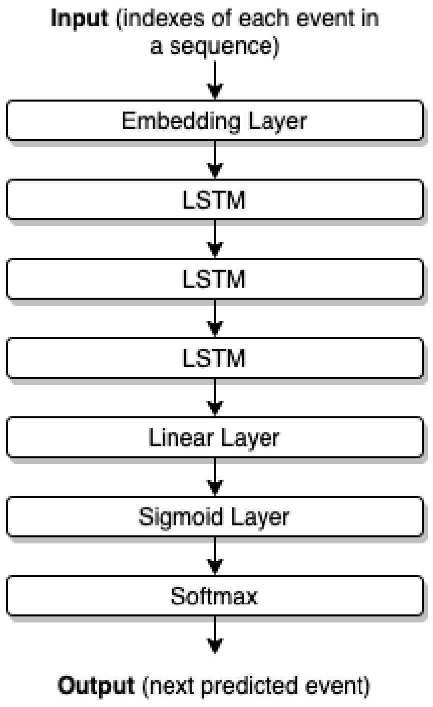

Input (indexes of each event in a sequence)
Embedding Layer
LSTM
LSTM
LSTM
Linear Layer
Sigmoid Layer
Softmax
Output (next predicted event)

FIGURE 5.8: CBOW and Skip-gram event prediction model architecture

5.8.5 Ensemble Learning

Due to the reduced number of sensors and all the problems faced during data collection, the dataset is may be too small for an accurate training of the word embeddings. This way, an approach will be taking in order to try and improve prediction.

Ensemble learning is a method that consists of combining several models to achieve better results. There are three types of ensemble learning: bagging, boosting and stacking.

- **Boosting** - combination of multiple models with the same architecture trained sequentially (depend on each other) to overcome biased dataset
- **Bagging** - combination of multiple models with the same architecture trained in parallel with different sets of data to overcome variance in the dataset
- **Stacking** - combination of heterogeneous models trained in parallel

One of the challenges this thesis faces is the reduced amount of available data that can be insufficient to train a model that produces good results. The ensemble boosting tries to overcome the problem by training on data that previous models failed to learn from, while ensemble stacking trains the same model architectures with different data. These methods try to overcome dataset problems that either are highly biased or have a high variance. Ensemble stacking trains several heterogeneous models to achieve better results, however the selected machine learning architectures are very similar, not making this method the best approach to take.

Bagging

Ensemble Bagging consists in training several models in parallel and averaging them to obtain a final prediction [25]. From the main dataset, several bootstrap samples (smaller datasets) are created to allow parallel training with different data. By training the same model architecture with different samples of the dataset, the result will be a set of different models that can hopefully produce a final model with less variance [25].

Algorithm 4 Ensemble Bagging

```
1: **Dataset**: $(x_i, y_i)$ pairs
2: **k**: number of models to train
3: for $i = 1...k$ do
4:     Take a random bootstrap sample $D_k$ from the Dataset
5:     Train model k with $D_k$
6: end for
7: Combine all the k models trained
8: Obtain votes from all the k models
9: Obtain final vote from the majority
```

Boosting

Ensemble boosting trains three models with the same architecture sequentially. After each model is trained, an evaluation is performed to check which samples of the training set fail to perform. The failed samples will then be used to train the second model and the final model is trained with the samples that both the first and seconds models failed to predict. When making the final prediction, each one of the train modules makes its inference and the one with a higher number of votes is the final decision. This approach allows models to complement themselves by training them on the tasks they are not so keen on.

Algorithm 5 Ensemble Boosting

```
1: **Dataset**: $(x_i, y_i)$ pairs
2: **N**: size of the dataset
3: Select $S_1 < N$ samples from the dataset
4: Train the first model on the $S_1$
5: Select $S_2 < N$ samples from the dataset, half of them misclassified by the first model
6: Train the seconds model on the $S_2$
7: Select $S_3 < N$ samples from the dataset that both first and second models misclassified
8: Train the third model on the $S_3$
9: Get final result by majority vote
```

5.8.6 CNN + LSTM

The purpose of the work developed by Alhussein *et al.* [13] was to forecast electric load. This approach will be based on theirs, adapting the architecture and inputs to predict events. The input will be a concatenated vector of one-hot representations of the period of the day the event took place, the day of the week and also the one-hot representation of the event.

The input vectors used by them had greater dimensions that the ones used in this thesis, which has effect on the convolutional layers used, that got their number reduced to one instead of one, as well as the maxpool and relu layers. These are followed by three sequence modeling LSTM layers, a dropout layer to prevent a model from overfiting and finally a linear layer at the output.

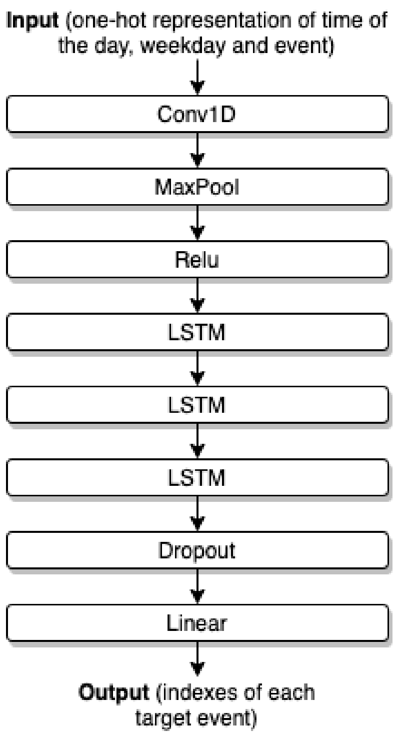

Input (one-hot representation of time of the day, weekday and event)
Conv1D
MaxPool
Relu
LSTM
LSTM
LSTM
Dropout
Linear
Output (indexes of each target event)

FIGURE 5.9: Hybrid CNN and LSTM architecture based on the proposed architecture of Alhussein *et al.*

5.9 Sequences Tree

While still trying to find the best machine learning approach possible to detect the pattern in the user activity, a more traditional non-machine learning approach was implemented to serve as baseline method.

This algorithm is truly based on the sequences generated on the data analysis procedure. To briefly explain how it all works, it consists of a prefix tree that stores all these sequences and

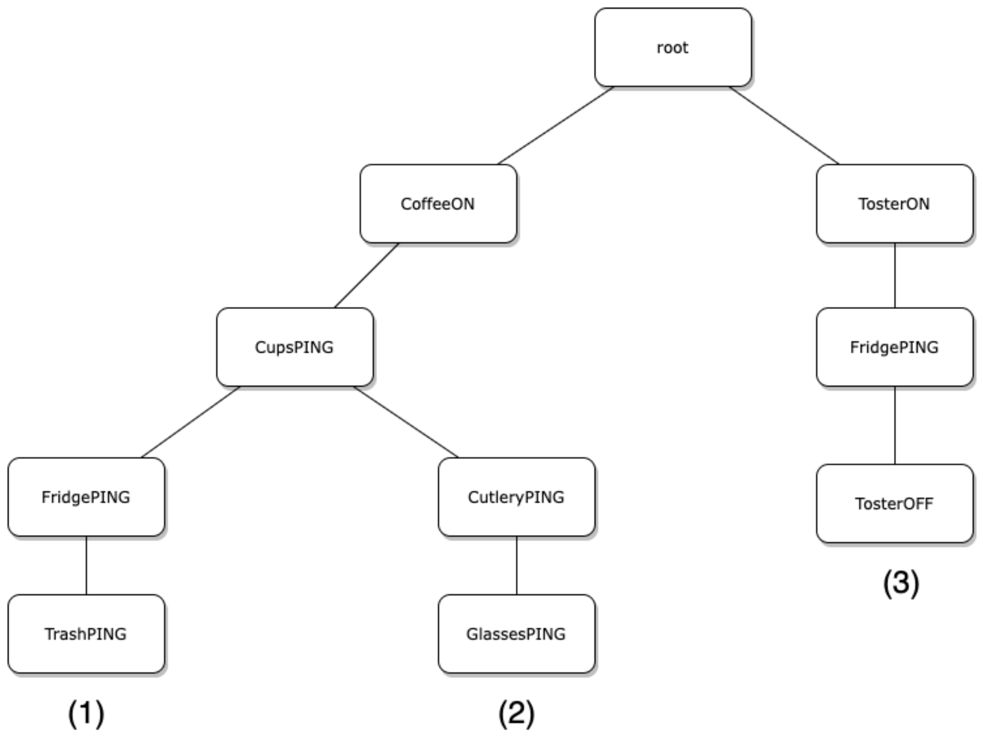


FIGURE 5.10: Prefix Tree

analyses the similarity of a user sequence by getting the most similar sequences stored in the tree, comparing it with all of them by matching the subsequences both contain.

5.9.1 Main Structures

Being a traditional non-machine learning model, there is a lot to take into account and it comes as a cost of many data structures. The following structures are the ones in charge of storing the analysed sequences on which any other structure will rely to get the necessary data.

Sequences Tree

This is the main structure that stores the sequences by which the similarity will be evaluated. After the data analysis described in chapter 5, the data is added to this structure. A file containing all the sequences is read and the prefix tree is created. This tree avoids redundancy by taking to its advantage equal parts of different sequences. For example, if two sequences have three events at the beginning that are the same, only three nodes will be created that will be common for both the sequences. On the third node, the sequences diverge resulting in two children for this node. This only works if equal events are placed at the beginning of sequences. The address of the last node of the sequence is stored in a lookup table, making it possible to access a given sequence without iterating through the whole tree.

Lookup table

The sub sequence similarity algorithm requires sequences with which the comparison can be performed and therefore it is essential a quick access to a sequence stored in the tree. The lookup table stores the address of the last node of each sequence in the prefix tree, with a different ID making it possible to obtain access to a particular series of events. Taking the example in figure 5.10, for each of the sequences (1), (2) and (3) there would be an entry that has the address for the last node of the sequence.

Occurrences Index

Each event is registered in the occurrences index. This structure stores the ID of every sequence where that particular event appears. When a user sequence arrives to be compared with the ones existing in the prefix tree, one of the steps is to get all the sequence IDs of every event and get their intersection. The remaining IDs are the ones whose sequences have all the events in common. These IDs are represented by the numbers below each sequence in figure 5.10.

5.9.2 Similarity Algorithm

Several structures are used to process the algorithm. For each room, in this case, bedroom, living room and kitchen, exists a structure that takes place in that room. This means that the similarity analysis is performed independently for each room.

The framework used to collect the data from the sensors [23] is based in MQTT. This feature allowed the reception of events in real-time, by subscribing to a topic. Whenever an event is received, a set of tasks is performed in order to evaluate the similarity of the sequence received until that moment:

1 Receive event JSON message and process the information to check if it's an interaction event and in that case if the event is expected. An event is expected if it is a PING, an OFF event after an ON event or a OFF event after an ON event, for each appliance. NOTE: after an ON/OFF event, repeated events may arrive with similar information, which is not desirable to consider as part of a user sequence.

2 Search the occurrences index for the sequences ID where the events have taken place

3 Get the intersection of all the sequence IDs to find the only sequences that match all the events

4 Get the address of the last node of each matched sequence and iterate through each one of them to get the whole set of events

5 Evaluate the user sequence for each matched tree sequence by analysing the subsequence similarity, meaning, searching for sets of following events in both sequences

6 Whenever a new event arrives, it is added to the vector the stores the user sequence currently active

7 If after the last PING or OFF event, a 10 minute interval has elapsed, a new sequence will be generated

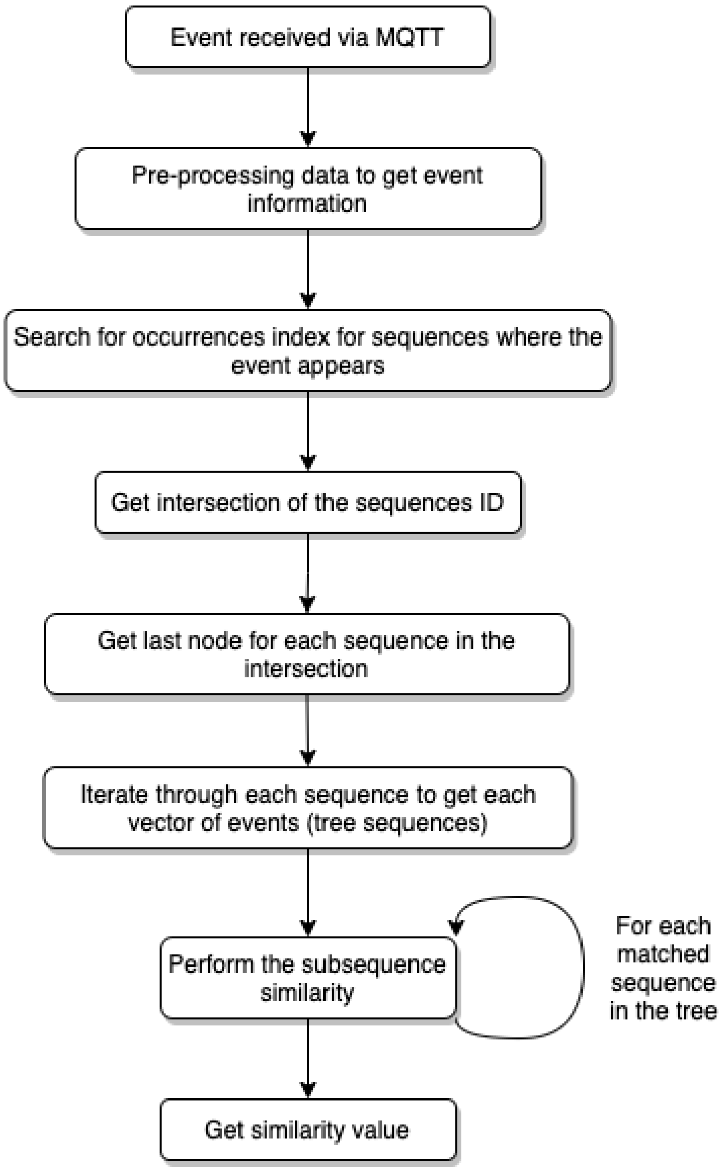

Event received via MQTT
Pre-processing data to get event information
Search for occurrences index for sequences where the event appears
Get intersection of the sequences ID
Get last node for each sequence in the intersection
Iterate through each sequence to get each vector of events (tree sequences)
Perform the subsequence similarity
For each matched sequence in the tree
Get similarity value

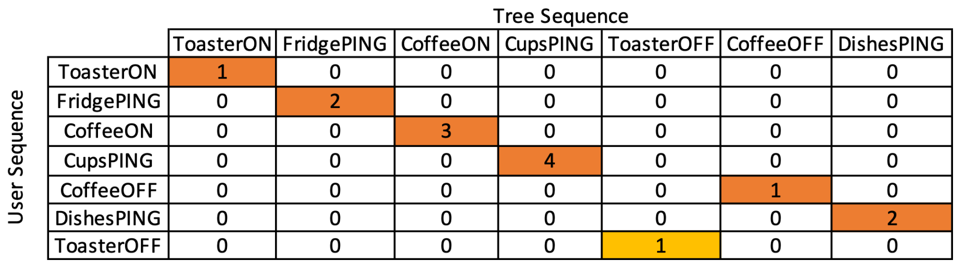
FIGURE 5.11: Pipeline of the traditional approach algorithm

Subsequence similarity

Let's consider a user sequence (US) of size m and a tree sequence (TS) of size n. They will form a matrix of size $m \times n$ which will be the voting system of this similarity comparison.

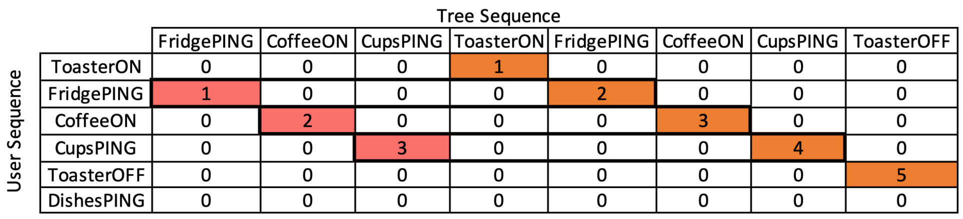
All the values of the matrix are initialised to zero. By iterating through each of the sequence vectors, a comparison between each of the selected events will be made. If both the events at position i and j are the same, the value in the matrix at location $[i][j]$ will be equal to the sum of one to the value of the its upper diagonal value, meaning, $matrix[i][j] = matrix[i-1][j-1]+1$. If this condition doesn't verify, the value remains the same, which means, zero.

If there is a match of a subsequence between the user and tree sequences, a diagonal of incremental values will exist (Table 5.4).



		Tree Sequence						
		ToasterON	FridgePING	CoffeeON	CupsPING	ToasterOFF	CoffeeOFF	DishesPING
User Sequence	ToasterON	1	0	0	0	0	0	0
	FridgePING	0	2	0	0	0	0	0
	CoffeeON	0	0	3	0	0	0	0
	CupsPING	0	0	0	4	0	0	0
	CoffeeOFF	0	0	0	0	0	1	0
	DishesPING	0	0	0	0	0	0	2
	ToasterOFF	0	0	0	0	1	0	0

Table 5.4: Subsequence similarity: The orange diagonal represents an existing subsequence between the user sequence and one of matched sequences store in the prefix tree. The numbers in yellow represent single event matches



		Tree Sequence							
		FridgePING	CoffeeON	CupsPING	ToasterON	FridgePING	CoffeeON	CupsPING	ToasterOFF
User Sequence	ToasterON	0	0	0	1	0	0	0	0
	FridgePING	1	0	0	0	2	0	0	0
	CoffeeON	0	2	0	0	0	3	0	0
	CupsPING	0	0	3	0	0	0	4	0
	ToasterOFF	0	0	0	0	0	0	0	5
	DishesPING	0	0	0	0	0	0	0	0

Table 5.5: Subsequence similarity: The darker line outlines the repeated subsequence events. To avoid double counting, only the orange subsequence, that start first, is considered

The higher the number of subsequences found in the matrix, the higher is the similarity. Although, if a subsequence repeats itself more than once along the tree sequence, there will be more than one diagonal in the same lines (Table 5.5). As the algorithm is supposed to compare the similarity of the user sequence with the already registered ones in the tree, this isn't a desirable behaviour. This would cause the algorithm to double count the same subsequence in the user sequence. If two or more diagonals overlap, the algorithm will only consider the one that starts first in the similarity equation. Let K be the sum of the sizes of all subsequences found, the similarity equation is as follows:

$$Similarity = \frac{2 \times K}{\text{size (user sequence)} \ + \ \text{size (tree sequence)}} \tag{5.2}$$

This evaluates the user sequence in two different ways:

1 Considers all the common events, generating a general similarity ratio

2 Only considers sub-sequences (with size greater than 1) and, in this case, the order by which the events occur matters

The values returned by the similarity function can have different meanings:

- If **Similatiry = 0** the user sequence has no subsequence matches with the tree sequence
- If **0 < Similarity < 1** the user sequence has that amount of similarity with the tree sequence
- If **Similarity = 1** the user sequence and the tree sequence are a perfect match
- If **Similarity > 1** the user sequence and the tree sequence have subsequence match but the user sequence is already bigger than the tree sequence and there are matches of the tree sequence that appear more than once in the user sequence

5.9.3 Challenges

The human being is capable of performing the same activity in an infinite number of ways. In this particular case, it means that the same activity, let's say lunch, can have a wide range of possible events, order, length, duration, and so on. The unpredictability of the human being makes this method, not the best approach as it can only work based on the stored information, not having the ability to generalise for unseen scenarios. This is a role where machine learning models can have a better performance. Those methods will be explored in the next chapter.

5.10 Machine Learning

There are various ways and architectures to deal with the task at hand. All of them were evaluated and some of them were tested in order to evaluate which one could better identify the events and the context in which they happen with the best accuracy possible. The following sections will describe all the steps taken until the final result and why some types of architectures perform better than others. But before that, there is something that all of the models have in common, which is the dataset.

5.10.1 Dataset

There are various challenges to overcome when training a neural network. In this case, as already mentioned, the dataset can introduce a high level of bias into the model. As mentioned in the Introduction, there are three main challenges that can influence the results: (1) the amount of data, (2) an unbalanced dataset and (3) data heterogeneity.

The last of the mentioned challenges was mitigated when the events were coded as words. From the data analysis described in chapter 5, the result is a file with sequences of events generated by the user when performing his daily activities. These sequences are like sentences in a text in which the text is the whole set of sequences available. Most of the NLP models that exist are trained with large datasets from websites like "Wikipedia" that are capable of providing text corpus with millions of words, sentences in which there is a logic to the order in which they appear, there is a context which makes it possible to find patterns and define rules to model sequence data.

However, all the data collected through the sensors comes not only from a restricted dataset but the order in which these events appear in a sequence doesn't follow a specific set of rules. To better explain, lets imagine a user is taking his breakfast and generating the following sequence:

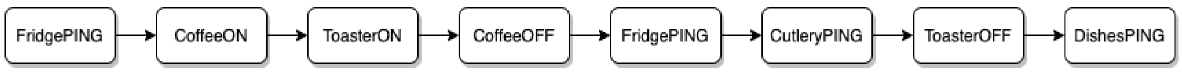


FIGURE 5.12: Example of a sequence of events representing a user having breakfast

This is possible to understand that very probably breakfast is being prepared. Now lets change the order of these events:

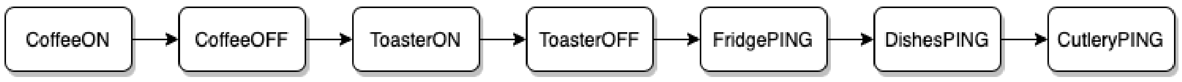


FIGURE 5.13: Example of another sequence of events of a user taking his breakfast that contain the same events but in a different order

This sequence of events represents exactly the same activity, although the order of the events has changed. It's possible to noticed that for every ON event there is always an OFF event, but apart from that, the order could be any.

In a normal sentence, from a book for example, there are nouns, verbs, adjectives and so on. The order in which those words appear has to follow a certain set of rules, otherwise the sentence would make no sense. With this type of events there is always space for randomness, which is not common in a corpus of text. The small size of the dataset, with the already presented twenty seven words, may make it difficult to find relations between them as the context in which some of the appliances are used can change drastically and the sequence still makes sense.

To try to find whether this may influence accuracy, several model architectures were tested. But one of the things all models have in common is the word representation.

5.10.2 Word Embeddings

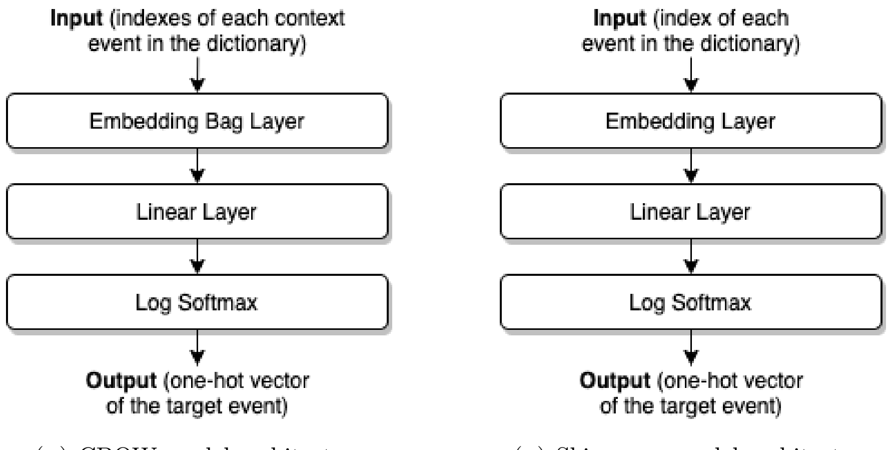
As introduced, there are two state-of-the-art word embeddings architectures, Word2Vec (CBOW and Skip-gram) and Glove. Each of them requires different input shapes thus they will be addressed separately. The initial sequence recognition is equal in every approach. In a first step, the CSV file generated by the data analysis algorithm is read. This file contains all the events divided into sequences identified by a sequence ID. To train the different architectures it's probably a better approach do deal with each sequence individually and therefore these events need to be aggregated into sequences, which means turning a dataset of events into a dataset of sequences.

Algorithm 6 Data setup

```
 1: $data \leftarrow$ read data from file
 2: $dataset \leftarrow$ create empty list to store sequence dataset
 3: $aux\_sequence \leftarrow$ auxiliar structure to temporarily store a sequence of events
 4: $sequenceID = 0$
 5: for $i < length(data)$ do
 6:     if $sequenceID == data[i].sequenceID$ then
 7:         $aux\_sequence \leftarrow$ append current event data[i]
 8:     else
 9:         $dataset \leftarrow$ append aux_sequence
10:         sequenceID = data[i].sequenceID
11:         $aux\_sequence \leftarrow clear$
12:         $aux\_sequence \leftarrow$ append current event data[i]
13:     end if
14: end for
```

Word2Vec

Both the architectures used to train the Skip-gram and CBOW are very simple. They consist only of a hidden layer, that contains the embeddings which will then be fed to a linear layer, in order to convert the embeddings to an output with the size of the event vocabulary and finally a log softmax layer. For each event received as input, the embedding layer returns the corresponding embedding vector that will go through the model. The CBOW model receives as input a number of events according to the chosen window size. In order to be compatible with the architecture, all the embedding vectors are averaged to obtain only one vector. The embedding bag layer does this procedure automatically.


Input (indexes of each context event in the dictionary)
Embedding Bag Layer
Linear Layer
Log Softmax
Output (one-hot vector of the target event)
Input (index of each event in the dictionary)
Embedding Layer
Linear Layer
Log Softmax
Output (one-hot vector of the target event)

(A) CBOW model architecture (B) Skip-gram model architecture

FIGURE 5.14: Word2vec model architectures

Skip-gram

In Skip-gram the word embeddings are trained with the aim of learning the context in which the words appear. By giving an event to the model as input it would return the words that usually occur near this one.

After having the dataset divided into sequences, the next step is to create the inputs and labels to train the model.

Skip-gram architectures receive one input and can return many others, the contexts. This way, by defining a context window, let's say of two events, the event in the middle will be our input and two events on each side of the middle event will be the context of this word, creating pairs of data (**input**=middle event, **target**=context events). This is exemplified in figure 5.15.

The red box in figure 5.15 represents the context window of size two for the centre word "CoffeeOFF". In this example the data generated to train would be the pairs (CoffeeOFF, CoffeeON), (CoffeeOFF, ToasterON), (CoffeeOFF, FridgePING) and (CoffeeOFF, CutleryPING).

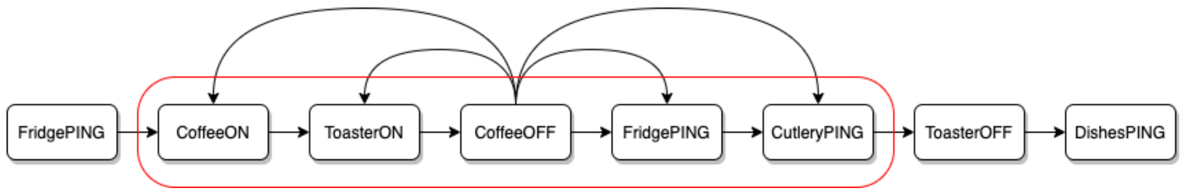


FIGURE 5.15: How to create the dataset to train Skip-gram architecture

The algorithm to take care of processing the data is the following:

Algorithm 7 Skip-gram Data setup

```
1:  $dataset \leftarrow$ stores the event sequences
2:  $CONTEXT\_SIZE \leftarrow$ size of the window of events
3:  $x \leftarrow$ stores model input
4:  $y \leftarrow$ stores model label
5:  for $i < length(dataset)$ do
6:      for $j < length(dataset[i]) - CONTEXT\_SIZE$ do
7:          for $a \leftarrow -CONTEXT\_SIZE <= CONTEXT\_SIZE$ do
8:              if $a! = 0$ then
9:                  $x \leftarrow$ appends dataset[i][j+a]
10:                 $y \leftarrow$ appends dataset[i][j]
11:             end if
12:         end for
13:     end for
14: end for
```

CBOW

The data processing for the CBOW model is the other way round of the Skip-gram in which the input of the network is the sequence of events that give context to the middle word, and the middle work is the target. One advantage of this approach over the previous one is that the training is much faster as the inputs are in a smaller number. This way, going back to figure for the CBOW approach the data generated to train the model would be a single structure, for the red box, following the same shape, (input, target). It would be like ([CoffeeON, ToasterON, FridgePING, CutleryPING], CoffeeOFF).

GloVe

The input data in GloVe architecture follows a different pattern. As specified in chapter 1 this word embeddings architecture isn't a window-based approach as it learns the embeddings from the entire corpus which consists of all the sequences displayed like a text. The problem with this architecture is that it requires a great amount of data that a single user is incapable of producing. After trying to train the glove embeddings, no results were achieved.

Algorithm 8 CBOW Data setup

```
1: $dataset \leftarrow$ stores the event sequences
2: $CONTEXT\_SIZE \leftarrow$ size of the window of events
3: $x \leftarrow$ stores model input
4: $y \leftarrow$ stores model label
5: for $i < length(dataset)$ do
6:     for $j < length(dataset[i]) - CONTEXT\_SIZE$ do
7:         for $a \leftarrow -CONTEXT\_SIZE <= CONTEXT\_SIZE$ do
8:             if $a! = 0$ then
9:                 $aux\_x \leftarrow$ appends dataset[i][j+a]
10:            else
11:                $y \leftarrow$ appends dataset[i][j]
12:            end if
13:        end for
14:        $x \leftarrow$ append context events
15:    end for
16: end for
```

5.10.3 Event Prediction

LSTM is the current state-of-the-art in sequence modelling. Trying to predict events in a scenario where there is not wrong order of events, as there is with common language sentences, makes this task difficult to implement. Nevertheless, taking advantage of the correlations of word found training word embeddings, a transfer learning technique will be applied to try predicting the next events with the trained embeddings in a separate model.

The architecture represented in figure 5.16 receives in the embedding layer the indexes of each word event and outputs the corresponding embedding vectors. They will be fed to three sequence LSTM layers and at the end a linear, sigmoid and softmax layers which will output a vector with the size of the vocabulary with the prediction being the index of the highest value.

5.10.4 Hyperparameter fine-tuning

There are parameters that can widely affect the performance of the final model. Parameters like batch size, number of epochs, context size for the training of word embeddings, size of the embeddings, size of the hidden layers, and so on. The number of combinations of all these parameters is considerable and testing them all by hand would be unprofitable and time-consuming. "Weights and Biases" [24] was therefore responsible for running several combinations of parameters and logging them to a website where the information could be examined afterwards.

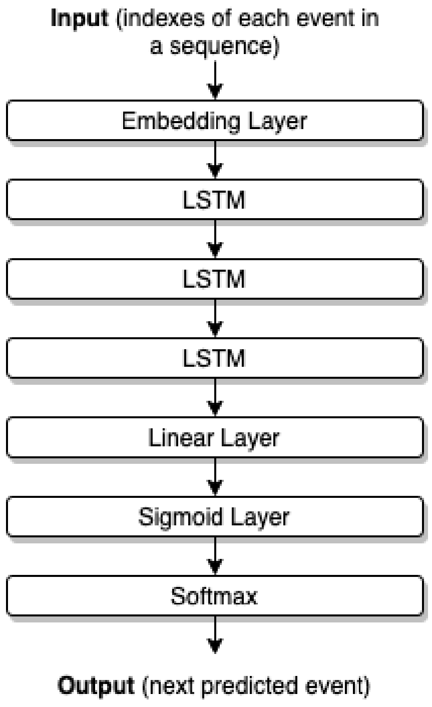

Input (indexes of each event in a sequence)
Embedding Layer
LSTM
LSTM
LSTM
Linear Layer
Sigmoid Layer
Softmax
Output (next predicted event)

FIGURE 5.16: CBOW and Skip-gram event prediction model architecture

5.10.5 Ensemble Learning

Due to the reduced number of sensors and all the problems faced during data collection, the dataset is may be too small for an accurate training of the word embeddings. This way, an approach will be taking in order to try and improve prediction.

Ensemble learning is a method that consists of combining several models to achieve better results. There are three types of ensemble learning: bagging, boosting and stacking.

- **Boosting** - combination of multiple models with the same architecture trained sequentially (depend on each other) to overcome biased dataset
- **Bagging** - combination of multiple models with the same architecture trained in parallel with different sets of data to overcome variance in the dataset
- **Stacking** - combination of heterogeneous models trained in parallel

One of the challenges this thesis faces is the reduced amount of available data that can be insufficient to train a model that produces good results. The ensemble boosting tries to overcome the problem by training on data that previous models failed to learn from, while ensemble stacking trains the same model architectures with different data. These methods try to overcome dataset problems that either are highly biased or have a high variance. Ensemble stacking trains several heterogeneous models to achieve better results, however the selected machine learning architectures are very similar, not making this method the best approach to take.

Bagging

Ensemble Bagging consists in training several models in parallel and averaging them to obtain a final prediction [25]. From the main dataset, several bootstrap samples (smaller datasets) are created to allow parallel training with different data. By training the same model architecture with different samples of the dataset, the result will be a set of different models that can hopefully produce a final model with less variance [25].

Algorithm 9 Ensemble Bagging

```
1: **Dataset**: $(x_i, y_i)$ pairs
2: **k**: number of models to train
3: for $i = 1...k$ do
4:     Take a random bootstrap sample $D_k$ from the Dataset
5:     Train model k with $D_k$
6: end for
7: Combine all the k models trained
8: Obtain votes from all the k models
9: Obtain final vote from the majority
```

Boosting

Ensemble boosting trains three models with the same architecture sequentially. After each model is trained, an evaluation is performed to check which samples of the training set fail to perform. The failed samples will then be used to train the second model and the final model is trained with the samples that both the first and seconds models failed to predict. When making the final prediction, each one of the train modules makes its inference and the one with a higher number of votes is the final decision. This approach allows models to complement themselves by training them on the tasks they are not so keen on.

Algorithm 10 Ensemble Boosting

```
1: **Dataset**: $(x_i, y_i)$ pairs
2: **N**: size of the dataset
3: Select $S_1 < N$ samples from the dataset
4: Train the first model on the $S_1$
5: Select $S_2 < N$ samples from the dataset, half of them misclassified by the first model
6: Train the seconds model on the $S_2$
7: Select $S_3 < N$ samples from the dataset that both first and second models misclassified
8: Train the third model on the $S_3$
9: Get final result by majority vote
```

5.10.6 CNN + LSTM

The purpose of the work developed by Alhussein *et al.* [13] was to forecast electric load. This approach will be based on theirs, adapting the architecture and inputs to predict events. The input will be a concatenated vector of one-hot representations of the period of the day the event took place, the day of the week and also the one-hot representation of the event.

The input vectors used by them had greater dimensions that the ones used in this thesis, which has effect on the convolutional layers used, that got their number reduced to one instead of one, as well as the maxpool and relu layers. These are followed by three sequence modeling LSTM layers, a dropout layer to prevent a model from overfiting and finally a linear layer at the output.

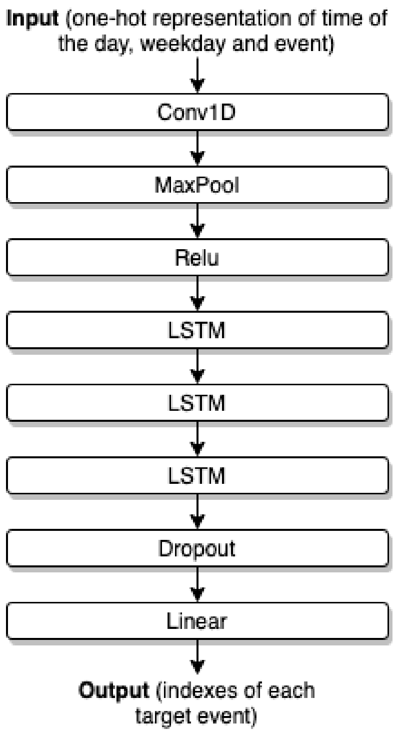

Input (one-hot representation of time of the day, weekday and event)
Conv1D
MaxPool
Relu
LSTM
LSTM
LSTM
Dropout
Linear
Output (indexes of each target event)

FIGURE 5.17: Hybrid CNN and LSTM architecture based on the proposed architecture of Alhussein *et al.*

6 Experimental Results

To evaluate the proposed methods, two datasets were generated from all the data gathered from the sensors (Table 6.1). The datasets, one for training and another for testing, were used in both the implemented methods so that the results can be compared. To train the machine learning architectures, the training dataset was splitted in two set of data, one exclusively for training, which represents 80% of the training dataset and the remaining 20% were destined to validate the accuracy of the trained model and hyper-parameter tuning.

	Training Dataset		**Test Dataset**	
	No. Events	**%**	**No. Events**	**%**
Coffee Machine	1342	8.18%	738	8.77%
Cutlery	1537	9.37%	819	9.73%
Toaster	328	2.00%	202	2.40%
Sandwich Maker	78	0.48%	66	0.78%
Cooking Robot	304	1.85%	134	1.59%
Microwave	410	2.50%	314	3.73%
Fridge	3575	21.80%	1633	19.41%
Oven	683	4.17%	260	3.09%
Herbs and Spices	1021	6.23%	429	5.10%
Glasses	906	5.53%	450	5.35%
Tools	272	1.66%	0	0.00%
Dishes	991	6.04%	463	5.50%
Pans	1545	9.42%	813	9.66%
Coffee Cups	580	3.54%	460	5.47%
Trash	1121	6.84%	688	8.18%
Platters	1703	10.39%	946	11.24%
Total:	16396		8415	
No. Sequences generated:	1958		998	
No. Days:	59		21	

Table 6.1: Distribution of the gathered data from the sensors into a training dataset and a test dataset

Both datasets must contain approximately the same data distribution for the same events, with the exception of the Tool event, which didn't get any events in the testing dataset in twenty-one days of data, due to an unknown problem. Since keeping this event could influence the results, it was deleted from both datasets.

6.1 Sequence Tree

The train dataset was fed to the algorithm that read the event sequences and stored them in a tree. For each event in the test dataset, similarity analysis was performed and presented at the end of each sequence. The results in the histograms presented in figures 6.2 and 6.1 represent the similarity values for each sequence detected by the proposed approach.

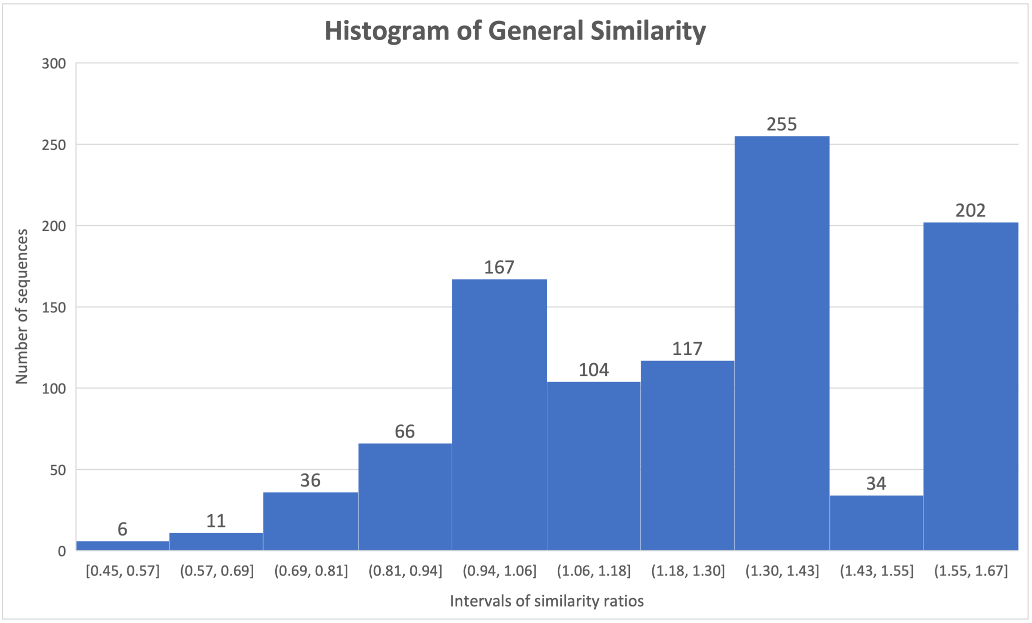


FIGURE 6.1: Histogram representing the general sequence similarity

In the first histogram (6.1) it is visible the distribution of sequences by ratio interval of general similarity. It only takes into account the matched events that exist in both the user event sequence and the matched tree event sequences. This way, the higher the ratio, the higher is the number of matched events. By analysing the histogram it is noticeable that most of the event sequences are placed above 0.94 (94%) which represents a very good similarity with previous behaviour of the user.

The second histogram (6.2) represents the distribution of the sequences by subsequence similarity ratio intervals. This metric values the order by which the events appear which is similar to say that this metric tries to find patterns in the event sequences that exist in the tree. The results show that almost half of the sequences got similarity values between 0 and 14%. Analysing the several sequences that got a ratio between 0.98 and 1.12, it is noticeable

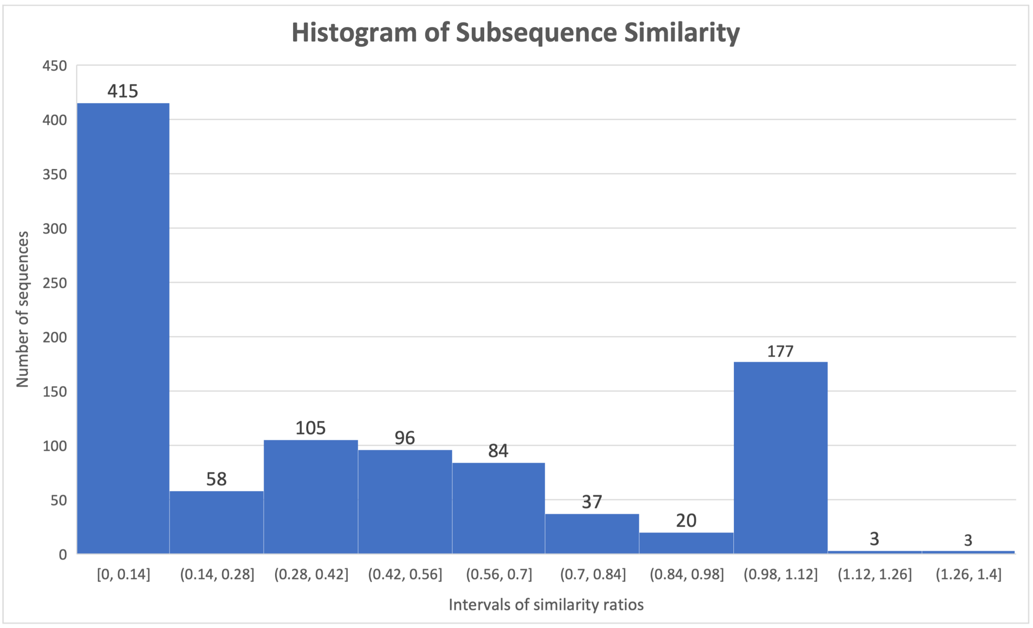


FIGURE 6.2: Histogram representing the subsequence similarity

that most of the user event sequences have a size in range two to four and found a perfect match with tree event sequences and that most of the sequences with a ratio bellow 0.5 have a size of at least 10 events. This suggests that the higher the range of sequences, the more difficult it is to find similarities between them.

6.2 Machine Learning

6.2.1 Word Embeddings

There are some key parameters that have effect on the training of the word embeddings, which are the context size, embeddings dimension, batch size and number of epochs. The context size is the size of the window of events to consider for training. The embeddings dimension is the size of each word vector. The batch size and number of epochs are more general machine learning parameters that are the number of data samples to give as input to the model before updating the weights, and the number of times the whole dataset will train the network, respectively.

To test the best combination of these values that could provide the best accuracy possible, the tool "Weights and Biases" [24] was used to test different combinations of parameters more efficiently, using a validation dataset. This tool was used in some of the machine learning approaches taken. The hyper-parameter tuning for both the CBOW and Skip-gram is displayed in Figures 6.3 and 6.4, respectively.

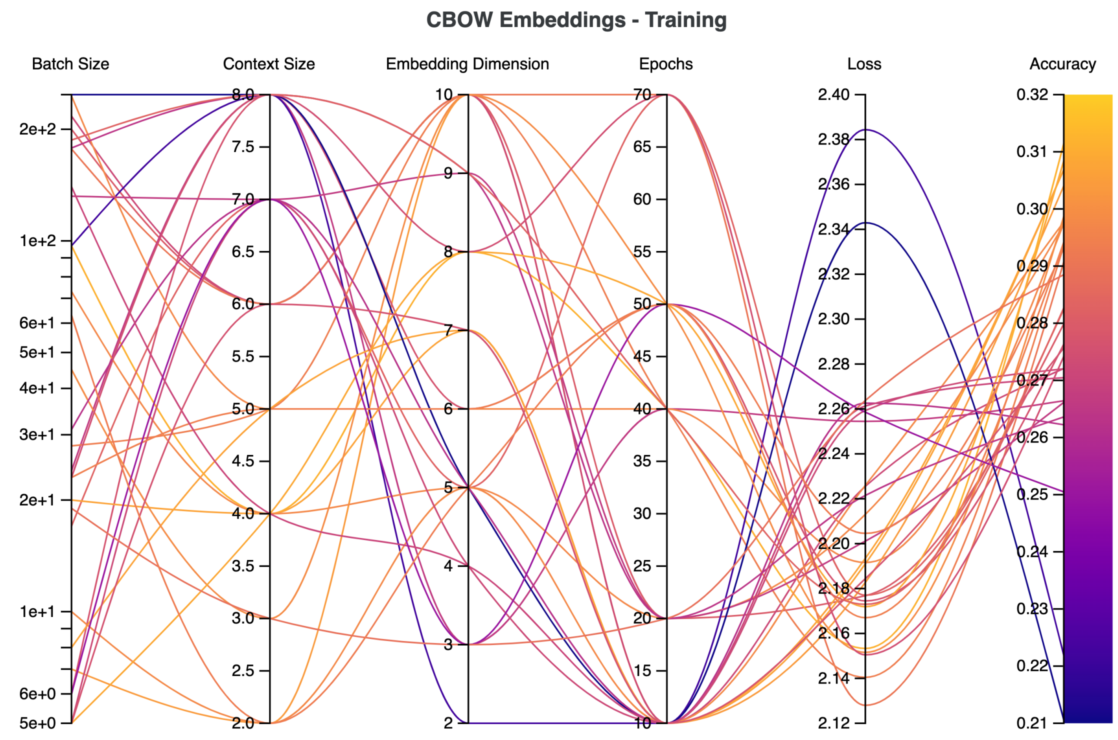


FIGURE 6.3: Hyper-parameter fine-tuning of the CBOW Embeddings model architecture

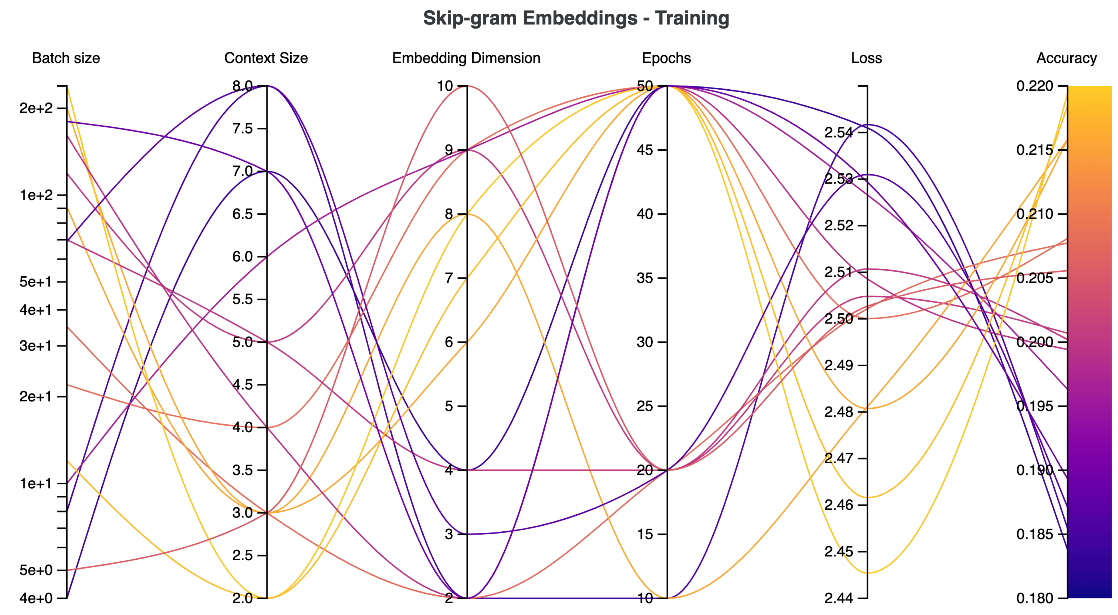


FIGURE 6.4: Hyper-parameter fine-tuning of the CBOW Embeddings model architecture

As previously mentioned, the CBOW embeddings are trained on the context to predict the middle word, while the Skip-gram embeddings are trained on the middle word to predict the context words. The second approach is computationally more intensive that the first. It is possible to notice that the training of the CBOW embeddings got a higher validation accuracy in general. Nevertheless, both embeddings were taken into consideration in further machine learning approaches. The best hyper-parameter configuration for each of the models is described in Table 6.2

The final accuracy of each model trained with the hyper-parameters above mentioned are the following:

	Batch size	**Context size**	**Embedding Dimension**	**Epochs**	**Accuracy**
CBOW	97	4	8	40	31.14%
Skip-gram	239	2	8	50	21.94%

TABLE 6.2: Evaluation of CBOW and Skip-gram models

The following machine learning architectures that require word embeddings will have them by transfer learning.

6.2.2 LSTM and Word Embeddings

The hyper-parameter fine-tuning for the architecture in figure 5.16 are represented in figure 6.5.

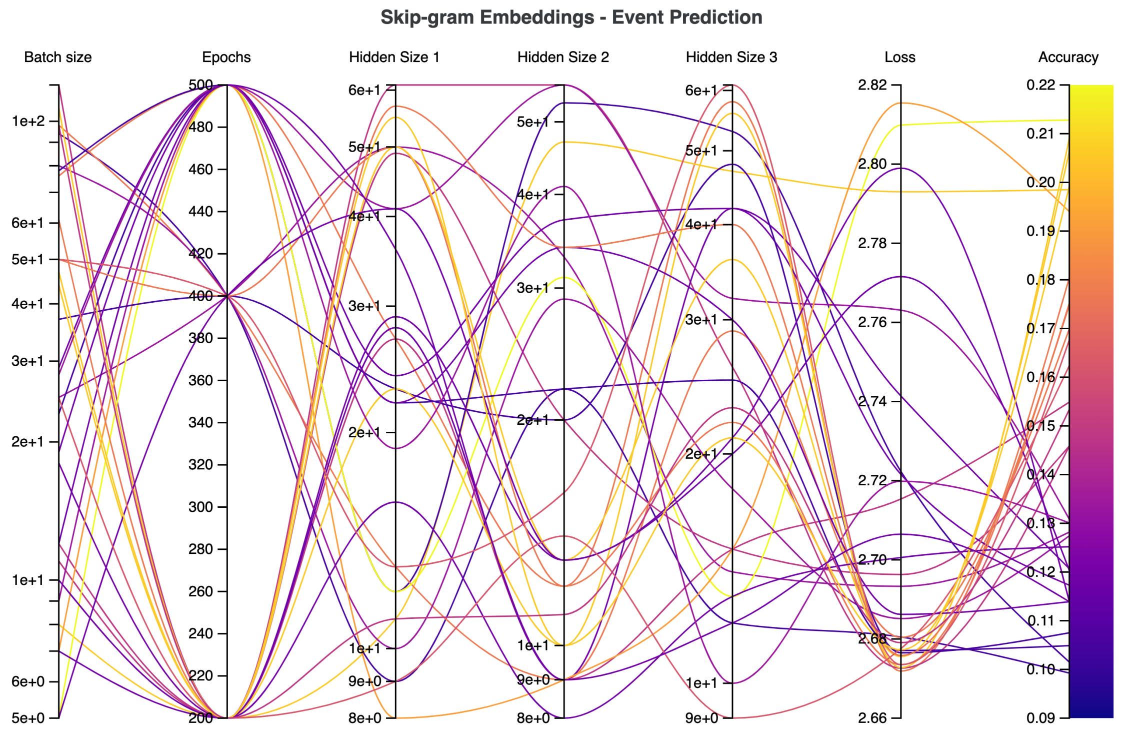


FIGURE 6.5: Hyper-parameter fine-tuning of the event prediction architecture with Skip-gram word embeddings

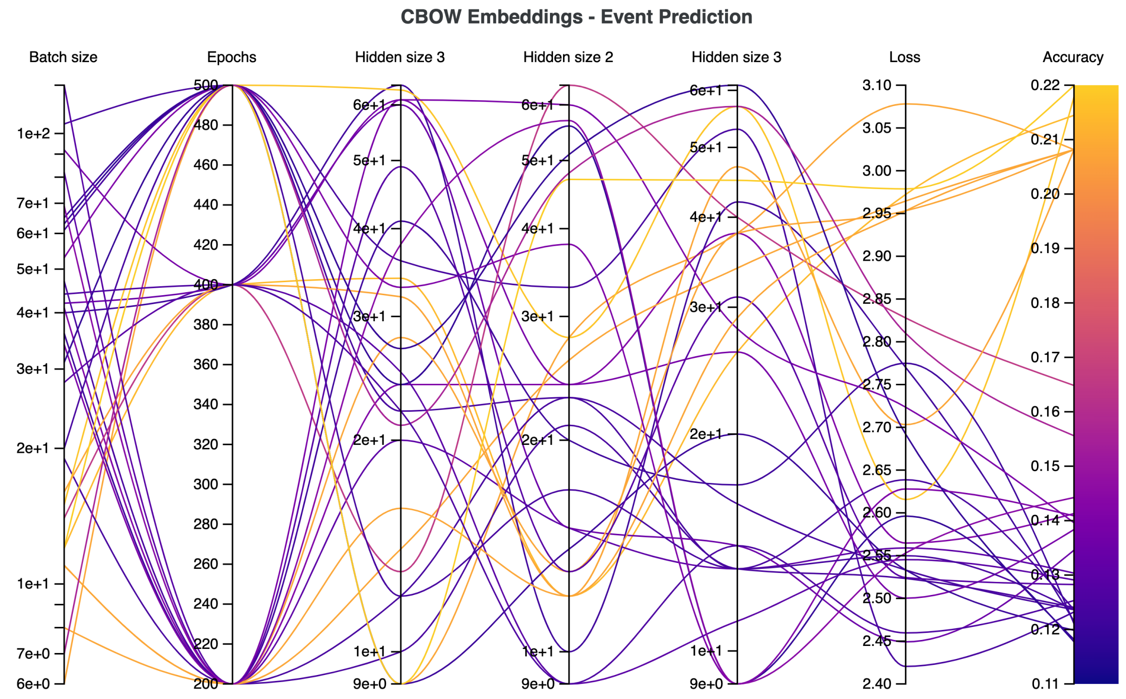


FIGURE 6.6: Hyper-parameter fine-tuning of the event prediction architecture with CBOW word embeddings

From the above plots it is possible to select the hyper-parameters that got the best accuracy for each of the event prediction architectures. These values are presented in table 6.3.

	Batch	**Hidden size 1**	**Hidden size 2**	**Hidden size 3**	**Epochs**	**Accuracy**
Skip-gram	5	12	31	13	500	21.28%
CBOW	12	9	47	45	500	21.99%

TABLE 6.3: Hyper-parameters and accuracy for both CBOW and Skip-gram event prediction models

6.2.3 Ensemble Boosting

Ensemble boosting requires more computational power due to the three models that are required to train in order to obtain the final decision. Due to the limited resources available, this approach was only trained with CBOW embeddings. The results are presented in table 6.4.

	Hidden Size			Accuracy			
Run No.	**LSTM 1**	**LSTM 2**	**LSTM 3**	**Model 1**	**Model 2**	**Model 3**	**Final**
1	69	35	128	8.64%	11.38%	14.54%	3.34%
2	58	28	65	6.89%	7.21%	7.14%	3.22%
3	65	32	32	18.17%	19.36%	6.49%	2.99%
4	**44**	**43**	**50**	**21.38 %**	**36.63 %**	**14.65 %**	**8.05 %**
5	34	125	19	12.40%	12.15%	13.45%	5.36%
6	60	23	66	6.64%	7.21%	6.01%	0.60%
7	17	29	59	26.62%	30.76%	11.89%	6.05%
8	100	95	53	8.34%	12.64%	10.91%	5.18%

Table 6.4: Ensemble Boosting Hyper-parameter tuning

Ensemble bagging differentiates from ensemble boosting by the dataset that is used to train the different models. To access the accuracy of ensemble bagging, the training dataset described in table 6.1, was divided in four parts, with 25% of the data each. Three of these sets were used to train each of the three models and the remaining part was used to validate the model. Keeping the hyper-parameters that best performed in ensemble boosting, the results of the ensemble bagging are described in table 6.5.

Model 1	**Model 2**	**Model 3**	**Final Accuracy**
32.64%	22.98%	25.85%	17.85%

Table 6.5: Ensemble bagging results

6.2.4 CNN + LSTM

The inputs fed to this model kept the same shape of the one proposed in [13]. Instead of using word embeddings, the accuracy was tested by feeding one-hot representation of the events, time of the day and day of the week. Unlike the other approaches, this one explores the prediction of multiple events which was one of the hyper-parameters to be fine-tuned. The results are represented in Figure 6.7.

Batch size	Context size	Events to predict	Epochs	Accuracy
228	5	1	400	9.39%

TABLE 6.6: Hyper-parameters of the highest accuracy CNN-LSTM model trained

	Batch size	Context size	Events to predict	Epochs	Accuracy
CBOW	228	4	1	400	9.69%
Skip-gram	228	2	1	400	14.34%

TABLE 6.7: CNN-LSTM model trained with word embeddings as event representation

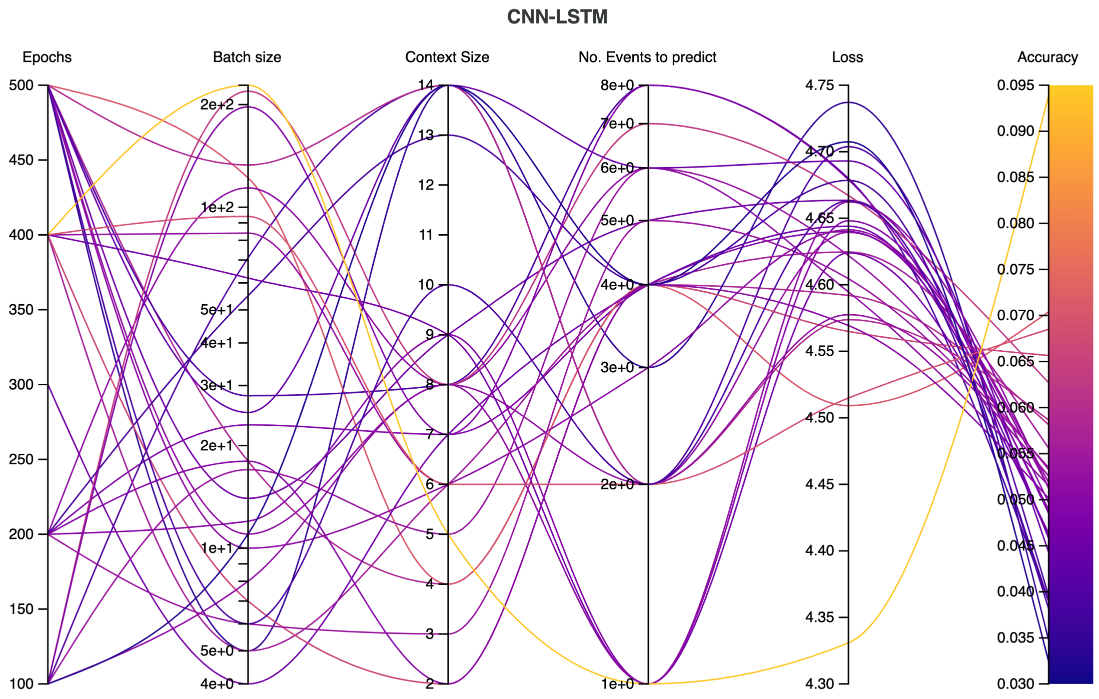


FIGURE 6.7: Hyper-parameter tuning of the Hybrid CNN-LSTM architecture

The highest accuracy model training is described in table 6.6.

Before word embeddings, word representation was performed using one-hot encoding. To access the performance of word embeddings, the model was trained by replacing the one-hot representation of the event by its word embedding representation. The hyper-parameters are the same as in table 6.6 except the context size, which will follow the corresponding presented in table 6.2. The results are represented in table 6.7

6.3 Overall results

Two approaches were proposed, one lossless without the use of machine learning techniques with the aim of analysing behaviour similarity and several machine learning approaches to predict future events. The word embedding model tries to learn the correlation between words that happen near one another. The type of vocabulary usually applied to train these models has a considerable size and the order by which the words appear follows certain rules that don't happen in this case scenario. The sensors used generated a total of nineteen events that could happen in any order. This not only makes it difficult to find word correlation but also to predict the next events.

In the sequences tree approach, the subsequence similarity evaluates user sequence events that appear in the same order as previously seen behaviour. A higher value suggests the existence of a pattern between two sequences. Due to the fact that almost half of the sequences evaluated got a subsequence ratio below 28% and that the length of these sequences was of at least 10 events each, is it possible to conclude that finding patterns in this scenario is a difficult task, which is confirmed by the low accuracy of the machine learning models developed.

The training of the words embeddings got an accuracy of 31.14% in getting the center word from a set of nine words (CBOW) and an accuracy of 21.94% in getting each word, given the center one, in a radius of 2 words (Skip-gram).

Nevertheless, from a total of 20 events, the probability of choosing the correct future event would be equal to 5%. The LSTM with Word Embeddings models got accuracies of 21.28% and 21.99% which is much better than a random guess.

The ensemble boosting, usually applied when the data available is poor, trains several models with data that the previous models failed to learn. The selected model architecture was the LSTM with word embeddings (immediately mentioned above). The best final accuracy obtained was 8.05%. On the other hand, it is noticeable that the second model, which was trained with data that the previous model failed to predict, got a better prediction accuracy. The ensemble bagging got better results than ensemble boosting, having a final accuracy of 17.85% yet being a lower accuracy than each of the models trained separately. Ensemble learning tries to achieve better performing model, but that not always verifies. Ensemble boosting and bagging aim to tackle different dataset problems and struggle to overcome the problems of one another. The unbalanced dataset, the amount of data available and the non-equally distribution in time, sets a high level of complexity that both ensemble methods are unable to accuratly learn from and therefore have a lower performance than the models trained separately.

Finally, the hybrid CNN-LSTM model architecture used was adapted from a proposed architecture that introduced CNN to better learn the features and the LSTM to model their sequence. The final accuracy of the one-hot representation of the events was 9.39%. By replacing the event representation by the corresponding word embeddings, it is notorious that word embeddings outperformed the one-hot representation, with Skip-gram having an advantage over

CBOW.

Although the machine learning results are not great, they demonstrate better prediction capacity than a random guess. The unbalanced dataset, certainly the reduced amount of data available and the various combinations of event sequences, and the ambition of producing an unsupervised approach highly increased the complexity of the work developed. These challenges have a great impact on the accuracy of the results, which was also verified in the lossless approach proposed that also struggled to find patterns in longer sequences. The results achieved show that it is possible to predict and find word correlation, even in difficult scenarios, which is a better approach than a random guess.

7 Conclusion and Future Work

The main purpose of this work was to develop a system capable of monitoring elderly activity in a non-intrusive environment. The thesis can be separated into two parts: the design of the hardware architecture and the development of data analysis algorithms.

The hardware architecture is composed of several accelerometer sensors and smart plugs. These devices are connected via ZigBee to a raspberry pi, acting as the gateway, and the messages they send are also available by subscription to an MQTT topic. This feature allows for remote monitoring and data analysis in real-time by assuring the delivery of messages. The framework used to deal with all communication was Zigbee2MQTT [23], an open-source framework that removes the need for the vendors' gateway, making it easier to integrate in a customised environment. All the data collected by the designed architecture was pre-processed in order to convert the events to word representations, by detecting the smart plug ON and OFF states based on the instant power consumption.

The work developed in the field of data analysis was different from already existing monitoring systems by being an unsupervised approach. Most of the existing related work was focused on labelling a set of actions to the activities they represent. This implies restraints in scalability on the grounds that every user can perform the same activity differently causing those approaches to be faulty. The work developed tries to analyse the data as a whole, find patterns in the user activity and use them to monitor the user. To accomplish that, several approaches were developed, one by analysing activity similarity and another by exploring prediction algorithms using machine learning.

The first approach is a lossless model that stores sequences of events into a structure (a tree of sequences) and analyses user sequences generated with the stored sequences, possible to do in real-time with MQTT. By doing so, it searches for patterns in the data by finding common subsequences and scoring them with a similarity ratio. A general similarity ratio is also returned, which only takes into consideration the existing events in both sequences.

With an accurate detection of user patterns, it is possible to build a model capable of predicting the user's next actions. To try and accomplish this, several machine learning models were implemented and tested. LSTM is the current state of the art in sequence modelling. It is widely used in language processing models due to its capacity of learning from long sequences of data.

Word embeddings were trained with the vocabulary generated in data pre-processing with the aim of trying to establish correlations between events, following both the architectures of Word2Vec. These were applied several times alongside LSTM's to try predicting sets of events from longer sequences.

The hybrid CNN-LSTM architecture and the ensemble learning approach were implemented with the hope to improve feature learning by using Convolutional layers and by training models on data that others failed to learn.

To sum up, the main contributions of this thesis were:

- The design of a system that allows the collection of data using non-intrusive sensors
- Representation of meaningful event information into semantic representation (power consumption)
- Implementation of machine learning algorithms to detect activity patterns and predict user activity
- Implementation of techniques and models to overcome data constraints

7.1 Future Work

It is essential that an elderly monitoring system operates with less number of flaws as possible and the work developed proved that there is still a long way towards the perfect approach to achieve a monitoring system that fits every user.

Although each user has distinct behaviour, there are probably key aspects in common. Federated Learning could be an approach to take in future work. It consists of gathering the models trained for each user and averaging them into a single one that can hopefully find patterns common to every user. This can bring benefits to a new user that already has a pre-trained model as starting point and may require less data to detect activity patterns.

Apart from the work developed, there are various other approaches important in elderly monitoring and many more that could be implemented:

- Fall detection system by using smartwatches or smartbands
- Vital signs monitoring
- Medication monitoring

These suggestions are already being investigated and some systems have already been proposed to deal with such situations. Although the number of systems that include all these features is very low or even none, which sets this as a prominent system.

7 Bibliography

[1] N. Roy, R. Dubé, C. Després, A. Freitas, and F. Légaré, “Choosing between staying at home or moving: A systematic review of factors influencing housing decisions among frail older adults,” *PLoS ONE*, vol. 13, 2018.

[2] N. Peel, D. Kassulke, and R. McClure, “Population based study of hospitalised fall related injuries in older people,” *Injury prevention : journal of the International Society for Child and Adolescent Injury Prevention*, vol. 8, no. 4, pp. 280–283, Dec. 2002, ISSN: 1353-8047. DOI: `10.1136/ip.8.4.280`. [Online]. Available: `https://europepmc.org/articles/PMC1756575`.

[3] P. Bellagente, C. Crema, A. Depari, P. Ferrari, A. Flammini, G. Lanfranchi, G. Lenzi, M. Maddiona, S. Rinaldi, E. Sisinni, and G. Ziliani, “Remote and non-invasive monitoring of elderly in a smart city context,” in *2018 IEEE Sensors Applications Symposium (SAS)*, 2018, pp. 1–6. DOI: `10.1109/SAS.2018.8336732`.

[4] S. Pinto, J. Cabral, and T. Gomes, “We-care: An iot-based health care system for elderly people,” in *2017 IEEE International Conference on Industrial Technology (ICIT)*, 2017, pp. 1378–1383. DOI: `10.1109/ICIT.2017.7915565`.

[5] B. Hu, H. Fahmi, L. Yuhao, C. Kiong, and A. Harun, “Internet of things (iot) monitoring system for elderly,” Aug. 2018, pp. 1–6. DOI: `10.1109/ICIAS.2018.8540567`.

[6] J. Alcalá, J. Ureña, and Á. Hernández, “Activity supervision tool using non-intrusive load monitoring systems,” in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, 2015, pp. 1–4. DOI: `10.1109/ETFA.2015.7301622`.

[7] J. Hao, A. Bouzouane, and S. Gaboury, “Recognizing multi-resident activities in non-intrusive sensor-based smart homes by formal concept analysis,” *Neurocomputing*, vol. 318, pp. 75–89, 2018, ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2018.08.033`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0925231218309834`.

[8] *Datasets*. [Online]. Available: http://casas.wsu.edu/datasets/.

[9] H. Zhu, H. Chen, and R. Brown, "A sequence-to-sequence model-based deep learning approach for recognizing activity of daily living for senior care," *Journal of Biomedical Informatics*, vol. 84, pp. 148–158, 2018, ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2018.07.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046418301321.

[10] T. Gueniche, P. Fournier Viger, and V. Tseng, "Compact prediction tree: A lossless model for accurate sequence prediction," vol. 8347, Dec. 2013, ISBN: 978-3-642-53916-9. DOI: 10.1007/978-3-642-53917-6_16.

[11] H. Fang and C. Hu, "Recognizing human activity in smart home using deep learning algorithm," in *Proceedings of the 33rd Chinese Control Conference*, 2014, pp. 4716–4720. DOI: 10.1109/ChiCC.2014.6895735.

[12] K. Li and Y. Fu, "Prediction of human activity by discovering temporal sequence patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1644–1657, 2014. DOI: 10.1109/TPAMI.2013.2297321.

[13] M. Alhussein, K. Aurangzeb, and S. I. Haider, "Hybrid cnn-lstm model for short-term individual household load forecasting," *IEEE Access*, vol. 8, pp. 180 544–180 557, 2020. DOI: 10.1109/ACCESS.2020.3028281.

[14] N. Tax, "Human activity prediction in smart home environments with lstm neural networks," Jun. 2018. DOI: 10.1109/IE.2018.00014.

[15] Y. Du, Y. Lim, and Y. Tan, "Activity prediction using lstm in smart home," in *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, 2019, pp. 918–919. DOI: 10.1109/GCCE46687.2019.9015492.

[16] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, 2014. arXiv: 1409.3215 [cs.CL].

[17] Y. Li, N. Du, and S. Bengio, *Time-dependent representation for neural event sequence prediction*, 2018. arXiv: 1708.00065 [cs.LG].

[18] A. Boyd, R. Bamler, S. Mandt, and P. Smyth, *User-dependent neural sequence models for continuous-time event data*, 2020. arXiv: 2011.03231 [stat.ML].

[19] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 2342–2350. [Online]. Available: `https://proceedings.mlr.press/v37/jozefowicz15.html`.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: `10.1162/neco.1997.9.8.1735`.

[21] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, 2013. arXiv: `1301.3781 [cs.CL]`.

[22] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014.

[23] *Zigbee2mqtt*. [Online]. Available: `https://www.zigbee2mqtt.io/` (visited on 01/24/2021).

[24] *Weights and biases*. [Online]. Available: `https://wandb.ai/site` (visited on 10/18/2021).

[25] *Ensemble methods: Bagging, boosting and stacking*. [Online]. Available: `https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205` (visited on 10/08/2021).