Linda Kolb, BSc

Using Artificial Intelligence to Classify Activities Captured in Smart Homes

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, April 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Dipl.-Ing. Dr. techn. Roman Kern for the continuous support of my Masters thesis. I am very thankful for his guidance during the time of research and writing this thesis.

I would like to acknowledge the support of Dr. Bin Wang, Dr. Joseph Rafferty and Professor Christopher Nugent from Ulster University campus Jordanstown when advising me on what research topic to choose and their help during the initial phase of this academic work.

Last but not the least, I would like to thank my mother and my friends for supporting me spiritually throughout working on this project.

Abstract

Due to a rapid increase in the development of information technology, adding computing power to everyday objects has become a major discipline of computer science, known as "The Internet of Things". Smart environments such as smart homes are a network of connected devices with sensors attached to detect what is going on inside the house and what actions can be taken automatically to assist the resident of the house.

In this thesis, artificial intelligence algorithms to classify human activities of daily living (having breakfast, playing video games etc.) are investigated. The problem is a time series classification for sensor-based human activity recognition.

In total, nine different standard machine learning algorithms (support vector machine, logistic regression, decision trees etc.) and three deep learning models (multilayer perceptron, long short-term neural network, convolutional neural network) were compared. The algorithms were trained and tested on the UCAMI Cup 2018 data set from sensor inputs captured in a smart lab over ten days. The data set contains sensor data from four different sources: intelligent floor, proximity, binary sensors and acceleration data from a smart watch.

The mutlilayer perceptron reported a testing accuracy of 50.31%. The long short-term neural network showed an accuracy of 57.41% (+/-13.4), the convolutional neural network in 70.06% (+/-2.3) on average - resulting in only slightly higher scores than the best standard algorithm logistic regression with 65.63%. To sum up the observations of this thesis, deep learning is indeed suitable for human activity recognition. However, the convolutional neural network did not significantly outperform the best standard machine learning algorithm when using this particular data set.

Unexpectedly, the long short-term neural network and the basic multilayer perceptron performed poorly.

The key drawback of finding a fitting machine learning algorithm to solve a problem such as the one presented in this thesis is that there is no trivial solution. Experiments have to be conducted to empirically evaluate which technique and which hyperparameters yield the best results. Thus the results found in this thesis are valuable for other researchers to build on and develop further approaches based on the new insights.

Kurzfassung

Aufgrund der jüngsten Entwicklungen in der Informationstechnologie ist die Ausstattung alltägliche Objekte mit einem eingebetteten Computer zu einer bedeutenden Disziplin der Informatik geworden, bekannt unter dem Namen “Internet der Dinge”. Intelligente Umgebungen wie zum Beispiel ein “Smart Home” besteht aus einem Netzwerk verbundener Geräte. Diese Geräte können mithilfe von Sensoren feststellen, was im Inneren des Hauses passiert und welche Aktivitäten automatisiert werden können um die Bewohnern des Hauses zu unterstützen.

In der vorliegenden Masterarbeit werden Algorithmen der künstlichen Intelligenz untersucht um menschliche Aktivitäten des alltäglichen Lebens (frühstücken, Video Spiele spielen etc.) zu klassifizieren. Das Problem ist eine Klassifizierung von Zeitreihendaten für sensorbasierte Aktivitätenerkennung.

Insgesamt wurden neun verschiedene Standard Algorithmen des maschinellen Lernens (Support Vector Machine, logistische Regression, Entscheidungsbäume etc.) und drei Deep Learning Algorithmen (mehrlagiges Perzeptron, Long Short-term Neural Network, Convolutonal Neural Network) verglichen. Die Algorithmen wurden mit dem Datensatz des UCAMI Cup 2018 mit Sensordaten eines Smart Labs, die über zehn Tage lang aufgezeichnet wurden, trainiert und getestet. Der Datensatz besteht aus Sensordaten vier verschiedener Quellen: Bodensensoren, Näherungssensoren, binäre Sensoren und Beschleunigungsdaten einer Smart Watch.

Die Experimente zeigten eine Korrektklassifikationsrate von 50,31% für das mehrlagige Perzeptron, 57,41% (+/-13,4) für das Long Short-term Neural Network und 70,06% (+/-2,3) für das Convolutonal Neural Network. Das Convolutonal Neural Network schnitt am besten ab, dicht gefolgt von dem Standard Algorithmus logistische Regression. Die Resultate dieser akademischen Arbeit zeigten, dass bei diesem speziellen Datensatz die

Deep Learning Algorithmen nicht signifikant bessere Ergebnisse erbrachten als die Standard Algorithmen.

Beim Finden eines optimalen Algorithmus des maschinellen Lernens, um ein komplexes Problem wie das in dieser Arbeit beschriebene zu lösen, ist einer der größten Nachteile, dass es keine triviale, eindeutige Lösung gibt. Verschiedene empirische Experimente müssen ausgeführt und evaluiert werden, um eine passende Technik und die dazugehörigen Hyperparamenter zu finden, welche das beste Resultat liefern. Aufgrund dessen sind die Ergebnisse dieser Masterarbeit für andere Wissenschaftler äußerst wertvoll und geben neue Einblicke, auf die neue Forschung gebaut werden kann.

Contents

List of Figures

1. Introduction

Due to recent development of wide area networks and an increasing digitalisation of everyday appliances, new ways of using computers to improve the living quality of humans have evolved. Sensors are becoming inexpensive and less power consuming which facilitates technological extensions of ordinary objects. Those objects can be pieced together to an ubiquitous net of interconnected devices known as the Internet of Things (IoT). The devices communicate with each other based on standard communication protocols .

One application of IoT that is discussed in this thesis is human activity recognition in environments that contain devices with additional technological enhancements connecting them to the internet which make them "smart". An example for such an environment is a smart home where sensors detect when a person lies down in the bedroom to sleep. Once the smart home system picks up on the activity using sensors, it categorises the event as "sleeping" using a classification algorithm. The system assists the person and switches the heating system of the room off without any human interference. Thus, a monitoring smart home collects sensor data, detects events, reacts and assists the resident when needed.

The topic of human activity recognition to enable assisted living is relevant to the world's society because of various possible applications. For example, Ann and Lau, 2014 lists areas such as surveillance systems, healthcare, eldercare and human computer interaction. Eldercare is attracting considerable interest since the population shifting to a distribution containing more elderly people than young ones. New ways of dealing with eldercare have to be found which makes it a research field with great potential. The goal is to enable elderly people to stay in their own home for as long as possible instead of sending them to nursing homes which might not be a long-term

option due to financial reasons. Other promising use cases of human activity recognition are assistance with coordination and scheduling. It becomes easier for the home owner to manage tasks as for example sustainable and economic heating or waste management.

1.1. Problem Statement

This thesis tries to answer the research question: To which extend can artificial intelligence help to classify activities captured in smart environments? Artificial intelligence describes systems that act "intelligently" which means that the system simulates intelligent decisions and imitates a human as mentioned by Turing, 1950 as the "imitation game". For a human, it would be fairly easy to put a label on an observed human activity. However, a computer must learn how to distinguish between different activities and what action has to be taken accordingly. Using labelled sensor data to train a machine learning model to recognise certain activities by their features is categorised as a supervised learning task Chollet, 2018, p. 135-136 and generally known as time series classification in the domain of human activity recognition.

Massive amounts of data are required to ensure that a generalised model can actually differentiate between the vast amount of human activities. The data set used to train the model of this project is the data set from the Ubiquitous Computing and Ambient Intelligence (UCAMI) Cup 2018 which is explained in detail in Espinilla, Medina, and Nugent, 2018. Labelled data sets for activity recognition are seldom available due to the time-consuming effort and necessary infrastructure of creating them. Therefore this extensive data set is particularly valuable to researchers in the field of human activity recognition. The data was captured inside a controlled environment that tried to simulate a real life apartment called a smart lab. A 24 year old male student has pretended to live in the smart lab for ten days. The data set includes data from four different sources with labels for 24 different categories of activities of daily living. The captured sensor recordings are split into a labelled training set from seven days and three days worth of

test data. The sensor data consists of chronological signals from the sensor devices which is commonly known as a time series.

The list of activities and their definition according to Espinilla, Medina, and Nugent, 2018 are presented in Table 1.1.

Activity Name	**Description of Activity**
Take medication	This activity involved the inhabitant going to the kitchen, taking some water, removing medication from a box and swallowing the pills.
Prepare breakfast	This activity involved the inhabitant going to the kitchen, taking some products for breakfast. This activity can involve (i) making a cup of tea with kettle or (ii) making a hot chocolate drink with milk in the microwave. This activity involves placing things to eat in the dining room, but not sitting down to eat.
Prepare lunch	This activity involved the inhabitant going to the kitchen, and taking some products from the refrigerator and pantry. This activity can involve (i) preparing a plate of hot food on the fire, for example pasta or (ii) heating a precooked dish in the microwave. This activity also involves placing things to eat in the dining room, but not sitting down to eat.
Prepare dinner	This activity involved the inhabitant going to the kitchen, and taking some products from the refrigerator and pantry. This activity can involve (i) preparing a plate of hot food on the fire, for example pasta or (ii) heating a precooked dish in the microwave. This activity also involves placing things to eat in the dining room, but not sitting down to eat.
Breakfast	This activity involved the inhabitant going to the dining room in the kitchen in the morning and sitting down to eat. When the inhabitant finishes eating, they place the utensils in the sink or in the dishwasher.

Lunch	This activity involved the inhabitant going to the dining room in the kitchen in the afternoon and sitting down to eat. When the inhabitant finishes eating, he places the utensils in the sink or in the dishwasher.
Dinner	This activity involved the inhabitant going to the dining room in the kitchen in the evening and sitting down to eat. When the inhabitant finishes eating, they place the utensils in the sink or in the dishwasher.
Eat a snack	This activity involved the inhabitant going to the kitchen to take fruit or a snack, and to eat it in the kitchen or in the living room. This activity can imply that the utensils are placed in the sink or in the dishwasher.
Watch TV	This activity involved the inhabitant going to the living room, taking the remote control, sitting down on the sofa and when he was finished, the remote control was left close to the TV.
Enter the SmartLab	This activity involved the inhabitant entering the SmartLab through the entrance at the main door and putting the keys into a small basket.
Play a video game	This activity involved the inhabitant going to the living room, taking the remote controls of the TV and XBOX, and sitting on the sofa. When the inhabitant finishes playing, he gets up from the sofa and places the controls near the TV.
Relax on the sofa	This activity involved the inhabitant going to the living room, sitting on the sofa and after several minutes, getting up off the sofa.
Leave the SmartLab	This activity involved the inhabitant going to the entrance, opening the main door and leaving the SmartLab, then closing the main door.
Visit in the SmartLab	This activity involved the inhabitant going to the entrance, opening the main door, chatting with someone at the main door, and then closing the door.

Put waste in the bin	This activity involved the inhabitant going to the kitchen, picking up the waste, then taking the keys from a small basket in the entrance and exiting the SmartLab. Usually, the inhabitant comes back after around 2 min, leaving the keys back in the small basket.
Wash hands	This activity involved the inhabitant going to the bathroom, opening/closing the tap, lathering his hands, and then rinsing and drying them.
Brush teeth	This activity involved the inhabitant going to the bathroom and brushing his teeth and opening/closing the tap.
Use the toilet	This activity involved the inhabitant going to the bathroom and using the toilet, opening/closing the toilet lid and pulling the cistern.
Wash dishes	This activity involved the inhabitant going to the kitchen and placing the dirty dishes in the dishwasher, and then placing the dishes back in the right place.
Put washing into the washing machine	This activity involved the inhabitant going to the bedroom, picking up the laundry basket, going to the kitchen, putting clothes in the washing machine, waiting around 20 min and then taking the clothes out of the washing machine and placing them in the bedroom closet.
Work at the table	This activity involved the inhabitant going to the workplace, sitting down, doing work, and finally, getting up.
Dressing	This activity involved the inhabitant going to the bedroom, putting dirty clothes in the laundry basket, opening the closet, putting on clean clothes and then closing the closet.
Go to the bed	This activity involved the inhabitant going to the bedroom, lying in bed and sleeping. This activity is terminated once the inhabitant stays 1 min in bed.

Wake up	This activity involved the inhabitant getting up and out of the bed.

Table 1.1.: The activities of daily living that are represented in the data set along with their description.

The idea is to use the data set to train a model which is able to distinguish between the above listed activities. The model can be incorporated into the IoT system of a smart home. Depending on what events are being detected in the house, the system can take action and use other installed devices such as the thermostat or a light bulb to be switched on or off.

The machine learning process for this problem statement is shown in Figure 1.1. To start the machine learning process, the raw data which potentially contains noise or errors needs to be preprocessed first. Then the machine learning algorithm can be applied to the data set. After testing different hyperparameters, a fitting model is chosen to be integrated into the smart home system. The system shall predict new samples on the fly - a live classification of unseen data.

1.2. Contribution to Research

It is a non-trivial task to deal with a large amount of data which comes with using sensors. Moreover, selecting an algorithm to find a pattern in a particular data set can be a challenge. Thus, the core problems in artificial intelligence are preprocessing the raw data to receive meaningful features and finding effective classification algorithms. The contribution of this thesis to research in this field is an analysis of what kind of time series classification algorithms for activity recognition already exist and how they compare.

The theoretical way of searching for a solution to the research question is to study different methods other researchers have used to incorporate artificial intelligence into performing human activity recognition in the context of IoT.

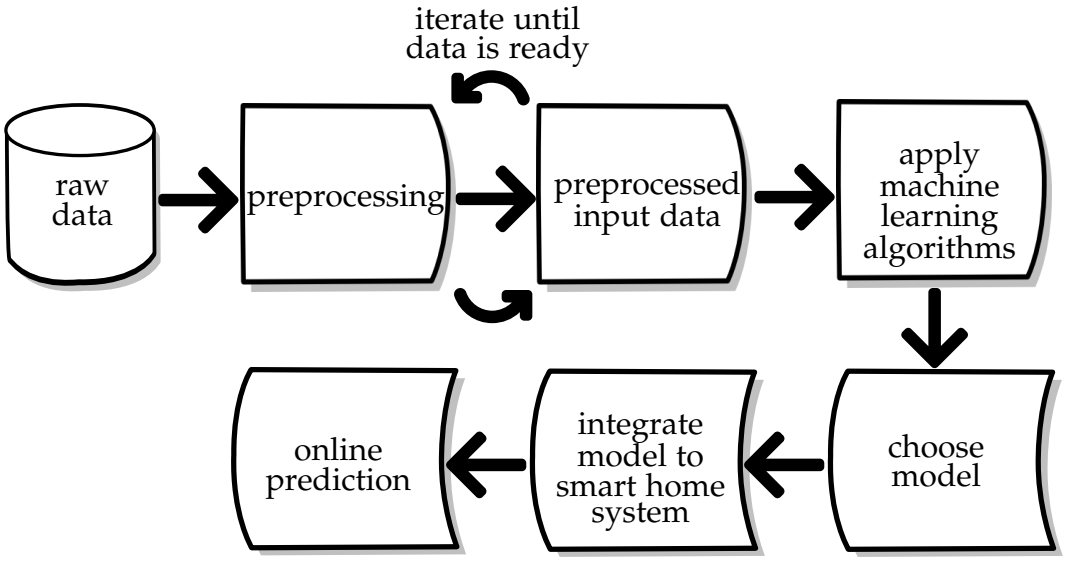


Figure 1.1.: This figure shows the machine learning process of the human activity recognition problem of this thesis. The raw data needs to be preprocessed first. Then the machine learning algorithm can be applied to the data set. After testing different hyperparameters, a fitting model is chosen to be integrated into the smart home system. The system can predict new samples online.

The main objective is to compare existing algorithms and settle on fitting methods for the classification problem of this thesis.

Regarding the practical part of this work, experiments with different machine learning algorithms are conducted to test the conclusions from the literature review. The mentioned data set from UCAMI Cup 2018 is processed and several standard machine learning and deep learning models are evaluated to classify the samples form the data set. Since sensor-based human activity recognition in a smart environment has only developed recently, few researchers have addressed the problem of finding an appropriate solution. Previous work has mainly focused on standard machine learning algorithms such as support vector machines or decision trees. Particularly deep learning methods such as neural networks have barely been used in the context of human activity recognition with time series data.

Applications for deep learning algorithms are more known in the domains of image classification and natural language processing. To the community, it is interesting to see if deep learning performs better than standard machine learning techniques. Setting up a neural network and training it is more time consuming than other machine learning algorithms. With this in mind, the reader of this thesis gains valuable insights whether spending more time on the implementation and execution of the algorithm provides better results when using this particular data set.

After the models investigated in this thesis have been trained, validated and tested, confusion matrices as well as other metrics are obtained to compare the performance of the classification. The data set has already been explored by other researchers who performed alternative classification approaches. Their existing results are compared to the metrics of the algorithms used in this thesis.

A commonly known problem in artificial intelligence is the fact that collecting good data sets to try out algorithms needs a high amount of resources such as time, budget and adequate infrastructure. Supporting the community with sharing data sets and the experiments of different techniques applied to the same data set can help data scientists learn from each other and develop new algorithms. The academic work presented in this thesis aims to add more understanding to what algorithms can be used in the context. Especially an exploration of how deep learning can be applied as a new

innovative technique proves valuable for other researchers in this field. Furthermore, the search for the most effective hyperparameters and methods to choose and improve a neural network architecture can be of use for other developers. Additionally, readers can benefit from learning about how Node-RED can be integrated to connect devices to build an IoT platform.

1.3. Thesis Outline

Chapter 2 discusses what an IoT platform can look like and elaborates on the challenge of using sensors inside of smart homes. Furthermore, the formal definitions of artificial intelligence, machine learning and deep learning are given and the theoretical background of the used deep learning algorithms are explained. An overview of the state of the art in the field of human activity recognition is also part of this chapter, including an analysis and comparison of the solutions of other programmers that participated in the UCAMI Cup 2018.

The proposed classification of this thesis is intended to be used in a smart home. Chapter 3 lists three use cases that could be solved by a smart system using the classification algorithm in the problem domain of assisted living. Additionally, functional and non-functional requirements for sensor-based IoT platforms are suggested.

Chapter 4 explores the data preparation pipeline of the data set in detail to explain to the user how the data is prepared before it is fed to the classification algorithms. The chapter starts by looking at how the data was collected, followed by a deeper look into the data set and its particularities. The feature selection process is justified.

Chapter 5 includes the used classification algorithms and lists the parameters. It is split into standard machine learning algorithms and deep learning algorithms. The network architectures of the chosen deep learning algorithms are described. The actual code snippets of the algorithm configurations is shown in A as part of the appendix. B lists the versions of the used tools and libraries.

Finally, the results of the algorithms are depicted and compared in Chapter 6. The chapter uses metrics such as accuracy, recall, precision, F1 score, MCC and categorical cross-entropy to grade the performance of the algorithms. Moreover, confusion matrices visualise which activities are wrongly classified.

Chapter 7 summarises the work of this thesis and ends with some concluding remarks. Further ideas about how to build on the knowledge gained from this academic work are proposed.

2. Related Work

This chapter shall provide the reader with basic knowledge about the major topics of this thesis. Section 2.1 discusses background information about the Internet of Things, artificial intelligence and its subordinate method neural networks. Section 2.2 presents existing solutions related to the problem domain of sensor-based human activity recognition.

2.1. Background

To comprehend the topics presented in this thesis, section 2.1.1 starts with an introduction to the information technology discipline called Internet of Things. In particular, sensor-based smart environments and its challenges are discussed. Human activity recognition is explained in detail as it is the main area of application for the machine learning algorithms presented in this thesis. Time series data sets are explored since they are closely related to activity recognition. Section 2.1.2 introduces artificial intelligence with regards to supervised learning, especially classification. The concept of neural networks is the focus of the classification solution in Chapter 5, therefore it is described in detail in section 2.1.3. Two deep learning algorithms (CNN, LSTM) are investigated in detail.

2.1.1. Internet of Things

Internet of Things (IoT) is a vastly growing topic in information technology and finds its application in various fields. According to Sharma, Shamkuwar, and I. Singh, 2019, the definition of IoT corresponds to a high number of objects being connected to the internet, either by wire or wireless, to

enable a better data transfer, make use of data analytics to infer meaningful information from the data and base decisions on this new knowledge. The idea is for those objects to improve aspects of human life such as waste management or health. Small everyday life decisions are being taken over by computing devices, for example refrigerators or thermostats. A possible architecture according to Sharma, Shamkuwar, and I. Singh, 2019 is the device-to-gateway communication model (see Figure 2.1 for a visual understanding). For example, a local area network (LAN) connects the sensor devices to a gateway. The sensors send their data to the gateway which forwards the data to a cloud service that stores the data. This service could also act as an application service or the application service could be a separate entity.

Smart Environments

R. Singh et al., 2019 claims that the application of automation in homes, cities, offices and industry has become increasingly popular. Sensors are placed in such environments to enable everyday objects to take over tasks related to their surroundings. To indicate that an environment or a single object possesses additional computational power to broaden and automate their function adds the prefix *smart* to them. Autonomous IoT systems of interconnected devices in an enclosed space such as a city or a building help with coordination and makes processes more efficient.

One example of a smart environment connected to this thesis are smart homes. The generally known term for gathering sensor data, extracting knowledge and learn which actions to take to improve the living quality of a resident of a smart home is called Ambient Assisted Living Paola et al., 2017. Possible examples for sensor devices in the network of a smart home are sensors to measure temperature, lighting or humidity. The use cases of a smart home are diverse and depend on what sensors and devices are installed inside the smart home. Potential use cases related to smart homes are further investigated in Chapter 3.

Using sensors comes with a range of challenges. Power consumption, real life processing, managing memory space and enabling the devices to communicate with each other are just some of the hardware requirements

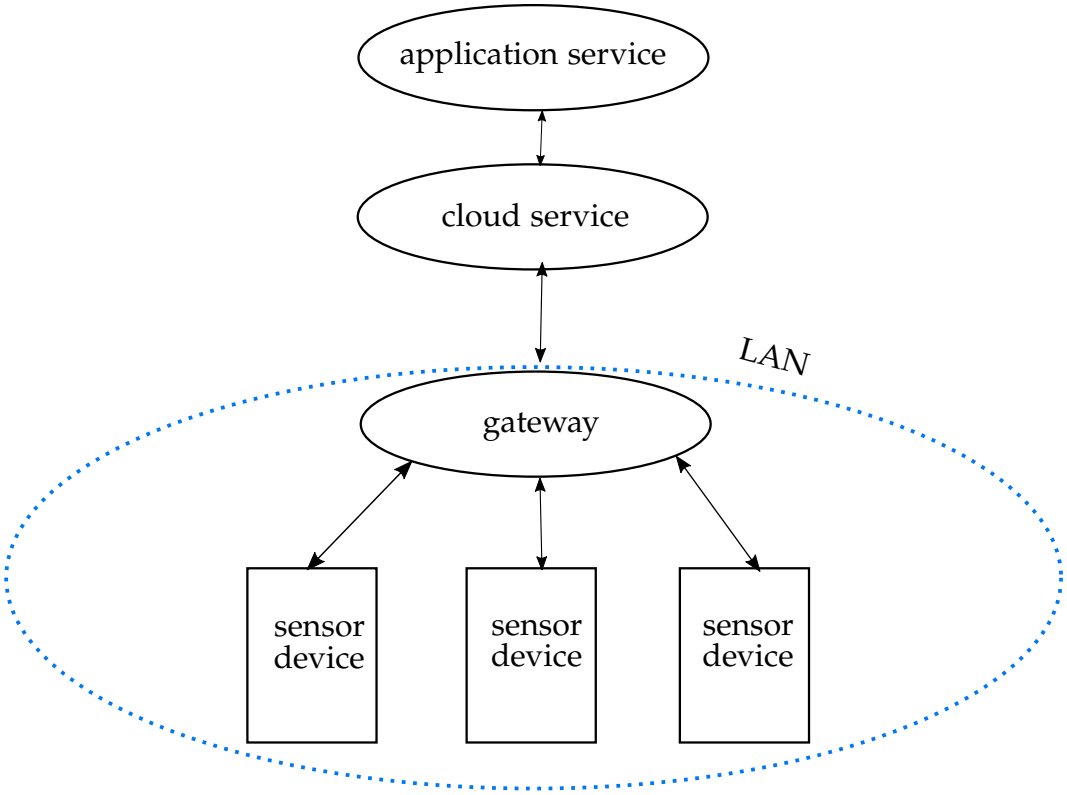


Figure 2.1.: Device-to-gateway communication model for as IoT architecture according to Sharma, Shamkuwar, and I. Singh, 2019: A local area network (LAN) connects the sensor devices to a gateway. The sensors send their data to the gateway which forwards the data to a cloud service that stores the data. This service could also act as an application service or the application service could be a separate entity.

that arise in IoT applications with interconnected sensors. The requirements for a smart home using the algorithms presented in this thesis are discussed in the Chapter 3 in detail. Concerning the machine learning aspects of senors in smart environments, special attention has to be drawn to the data cleaning - dealing with missing values and detecting outliers. Data cleaning is part of the preprocessing which is explained below.

Human Activity Recognition

Human activity recognition is the study to classify human activities occurring in daily life. Depending on the activities that are intended to be predicted, different sensors are used to obtain the needed information. In this thesis, activities of daily living are investigated (having breakfast, going to bed, playing video games etc.). Since those activities can not simply be judged from sensors related to the motion of the person, further sensors need to be installed in the smart environment the person is situated in.

2.1.2. Artificial Intelligence

The usage of mathematics to compute new knowledge from bare figures and thus enable computers to decide like humans has become immensely popular. The basic principle is that a computer is presented with an input and an associated task. The computer uses a set of rules to process the input and gives an answer on how to solve the task. Many new terms have arisen alongside this development that are topic of this thesis: *artificial intelligence* (AI), *machine learning* and *deep learning*. Chollet, 2018, p. 33-59 gives a differentiation of the meanings of those words which is summarised in the next paragraph.

AI is defined by Chollet, 2018, p. 35 as "the effort to automate intellectual tasks normally performed by humans". This definition opens up a wide variety of usage areas for AI, spanning from playing chess to predicting the stock market. In the case of playing chess, the computer simply has to follow a set of pre-defined rules. On the other hand, the problem of predicting how the stock market will evolve in the future requires the

computer to look at previous data and learn from it. This procedure is known as machine learning - a subgroup of AI where the programmers are not explicitly instructing the computer on how to process the data, but instead the computer learns and crafts the rules on its own by looking at the fed information. The learned rules are exploited to get new original ways of how to automate processing incoming data and subsequently solve a task.

To define the term *learning* precisely, the machine benefits from looking at past experiences when solving similar problems just as the new one presented to the machine. The algorithm tries to find a fitting generalisation method to learn from them and be able to handle new problems.

Machine learning is based on statistics and tries to build intelligent systems that are able to learn from data and solve problems which do not have an algorithmic solution (e.g. hand-coded rules that are unable to adapt on their own to fluctuations in the incoming data). One example to illustrate machine learning is supervised learning. It requires input data (for example the subject line of an email), an expected output (if the email is spam or not) and a measurement of success as feedback if the algorithm is performing correctly or not. While the machine learning algorithm is viewing the input data, it is trained to learn from what the expected output should be, comes up with rules and is being punished in case the prediction is wrong. These rules can be a single mathematical function, representing the mapping of the input to the output. Deep learning is a sub field of machine learning (see Figure 2.2 for how AI, machine learning and deep learning are related to each other) and focuses on building successive (*deep*) layers of increasingly meaningful representations instead of using just one (*shallow*) representation. The generated model is compound and can solve highly complex problems.

The outcome of the learning process is a *model*, a representation of the data. The model can also be called a hypothesis - an *educated guess* that needs to be evaluated. According to Géron, 2017, p. 39-40, there are three different types of learning explained in the following list:

- *Supervised learning:* The input data comes with labels that define the class identity of a data point. The idea is for the machine learning algorithm to train how to predict this label correctly from viewing

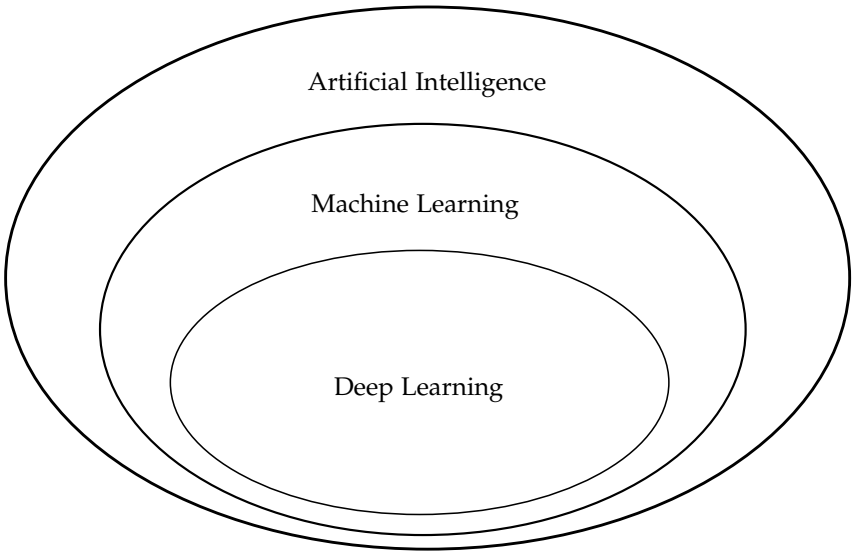


Figure 2.2.: Graph showing the relation between artificial intelligence, machine learning and deep learning. Artificial intelligence encapsulates various different methods, one of them being machine learning. Deep learning is a sub field of machine learning.

labeled data points and getting feedback from how accurate one prediction was. Logistic regression is an algorithm based on this type of learning. A use case is spam email detection.

- *Unsupervised learning:* The input data does not come with labels. The data is explored by the algorithm and - according to the observation of the features in the data - the data points are. for example, aggregated to groups, detected as outliers or patterns are discovered. One example is k-means clustering and a possible use case is clustering customers into segments.
- *Reinforcement learning:* This is a behaviour-based type of learning and is built on a reward system. The model receives rewards depending on how it behaves and reacts to inputs in certain situations. The goal for the algorithm is to maximise the sum of its rewards by only taking the decisions that are rewarded.

Supervised Learning

The learning type represented in this thesis is supervised learning. A typical training data set contains N samples $\{(x_1, y_1), ..., (x_N, y_N)\}$, $x \in X$ denoting the inputs and $y \in Y$ the output data. In the case of classification, the output data is a scalar representing the class the input data point belongs to. The goal of supervised learning is to find a decision function $f : X \mapsto Y$, associating each input x_n with an output y_n. The inputs are often called *feature vectors* $x_n \in \mathbb{R}^d$. Each feature vector holds d features which is mapped to an output. The supervised learning algorithm for classification looks at every input feature vector and tries to predict the class using guidance from an *error function* (also called *loss function*). The goal is to predict new unseen data points based on the classification knowledge the machine learning algorithm has gained.

The input data set in this thesis is represented as a data table. The relevant terms when working with classification are sample, feature and target value. A *sample* is a row in the data table and represents one instance that is processed by the machine learning algorithm. A *feature* is a column in the data table. Every sample has a variable that is associated with this feature. The aim is for the features to be independent of each other to ensure that the features do not represent the same knowledge. Redundancy creates overhead in the computation. The *target value* is the ground truth - the class label that the machine learning algorithm should learn to predict. Each input sample is associated with an output - a predicted one and the target value. A *class label* is a symbol representing this class such as *spam* and *not spam* in the context of spam email detection. In the input data set, the class name text is mapped to an integer number. The class labels are usually mutually exclusive - only one class can be associated with one sample. The machine learning algorithm will create a model based on what it learned from trying to predict the target value. It is also possible that the algorithm lists for each data sample how probable it is that the data point belongs to each available class using a percentage as measurement. The class probability that has the highest percentage is the one chosen as the dominant one.

To get feedback on how well the model generalises to predict new unseen data points, the model should be tested to monitor its performance. To

realise this, input data can be held back for evaluation. Géron, 2017, p. 77-78 mentions the following sets :

- *Training:* This set is used to train the model. The model tries to learn how to predict the target label of a data point. The parameters (for example weights) are fit to this data set.
- *Validation (optional):* This set is used to tune the parameters of a classifier during the training phase and give some guidance by measuring the performance of the model during training. For example, this data set can give an independent indication when to stop training. Just checking with the training set might lead to a bias.
- *Testing:* This set is used to test the model's performance (generalisation and predictive power) when classifying unseen data.

Géron, 2017, p. 77 suggests that a good set up is to use 80 % of the data for training and 20 % for testing and to use cross-validation (the training set is split up into complementary subsets which are used as validation and training sets in different combinations). Often several different models are created. The model with the best performance on the validation set should be chosen.

Time Series Classification

The data set from this project consists of multiple time series aggregated into a single one. A time series observes one or multiple random variables over time. For certain problems, it is relevant to not just look at a stationary, steady state view of the sample, but look at the problem in a more dynamic way, taking into account how variables change over time by adding recurrent dynamics Rolls and Treves, 1998, p. 351. Time is discrete since it consists of a series of equal time steps. A time series data set contains - in the context of sensor-based machine learning - a single value x_t of the same sensor feature measured at time step t (t = ...,-1, 0, 1, 2,...) Rolls and Treves, 1998, p. 351. Time series data is distinguished into univariate and multivariate. Univariate time series analysis only observes one variable over time, multivariate time series analysis considers multiple variables. A multivariate time series T is

a series of ordered observations collected sequentially over time and can be denoted as follows:

$$x_i(t); [i = 1, ..., n; t = 1, ..., m]$$

where: i = index of the measurements collected at each time point t
n = number of variables observed over time, $n > 1$ for multivariate
m = number of observations

The aim of time series analysis is to detect trends or seasonalities in the data. Time series classification intents to classify a novel data point by looking back at the other samples in time. Furthermore, it is possible to use machine learning to give a forecast of what value will occur in the future. The problem presented in this thesis is a multivariate time series classification using sensor data from a span of ten days. The input data for a time series classification problem is a 3D data tensor of shape (samples, time steps, features) Chollet, 2018, p. 103.

A couple of challenges come with learning from time series data: The model shall be able to handle sparse data sets and the right preprocessing measurements have to be investigated. Additionally, labelling is often a hard task for this type of data set, thus unlabelled data might be present in the data set. Labels are important since the model needs to have a feedback system to reliably learn from results and optimise its parameters accordingly.

Feature selection (extracting only features that are relevant for the problem at hand) for time series classification does not just rely on the captured data. Often additional statistical features are extracted. Time series data shows high fluctuation in the value of a feature. To smooth down the spikes in the data to get a feeling for the trend of the data, data points can be aggregated to a mean over a range of time steps. Often a single data point does not give an interpretable representation of the data - the mean might be more interesting to the data analyst. For example, when tracking temperature of a lake over a year, it might be enough to just include one data sample per day representing the mean temperature of the lake.

Typical machine learning algorithms that are already massively investigated in the domain of sensor-based time series classification are either nonlinear or ensemble algorithms[1]. *Ensemble* methods are models which are built on top of many other models to combine their classifiers and achieve a better performance Géron, 2017, p. 167. The nonlinear supervised learning algorithms are logistic regression, naive bayes, decision tree, support vector machine and k-nearest neighbours. The used ensemble algorithms are random forest, bagging, extra tree and gradient boosting.

Logistic regression is a classification algorithm that is set out to find a connection between the input features and the probability of a particular class as outcome using the sigmoid function. *Naive bayes* is a probabilistic classifier based on Bayes' Theorem. A *decision tree* is an easily interpretable predictive model based on a set of rules organised as a tree structure. *Support vector machines* find a hyperplane to use as a classification criteria. *K-nearest neighbours* uses the feature similarity to classify a data point by a majority vote of its k neighbours. *Random forest* builds several decision trees and chooses the class that was predicted most often. *Bagging* runs multiple weak learners independently from each other and operates a deterministic averaging process to decide on the class. The *extra tree* classifier works similar to the random forest, but uses randomised decision trees called extra trees. *Gradient boosting* lets weak learners run sequentially and combines them additively.

Data Processing Pipeline

For the training of machine learning algorithms, often massive amounts of data are necessary to achieve a reliable model. Before the data can be used, it has to be preprocessed. Figure 2.3 shows a typical preprocessing pipeline.

First, data has to be collected. The data collection process can be costly and time-consuming. When working with sensors, extensive infrastructure is necessary. Additionally, when dealing with a supervised learning problem,

[1] *Machine Learning Mastery - Machine Learning Algorithms for Human Activity Recognition* 2019.

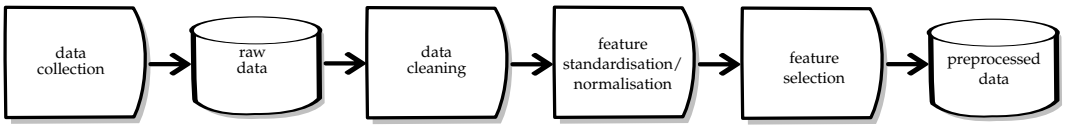


Figure 2.3.: The data preprocessing steps are: data collection → raw input data → data cleaning → feature standardisation/normalisation → feature selection → preprocessed data.

the input data needs to be labelled. The labelling task can hardly be automated and requires human resources.

Raw input data can be noisy (containing errors or outliers) and inconsistent (missing values). *Outliers* are anomalous data points, representing an *extreme* sample compared to the majority of other data samples. Left untreated, outliers can create a skew variable distribution for this feature. This misrepresentation of the data can result in a negative effect on the machine learning algorithm. There are various statistical and machine learning approaches to deal with outlier removal. Samples with errors or missing values can either be omitted or replaced by calculating a mean, median or mode of the feature and use this value instead of the missing or wrong entry.

With complex problems, the features are often not just a small set, but a large amount of properties which each only bring little info on its own. Therefore the features must be combined in the most meaningful way. Thus, a crucial part of the preprocessing pipeline is *feature selection*. During this step, only features relevant to the problem are extracted. Features that represent similar information should be combined to a single feature. Redundant information slows down the computation. In some cases it is not obvious which features contain relevant information for the machine learning problem. To check if features correlate with each other, the *Pearson correlation* is used in this thesis. The Pearson correlation quantifies the relationship between two features with a value between -1 and 1. The closer the number is to -1 or 1, the more correlated the two features are. If the value is 0, the features are linearly independent. A good method to analyse visually how features are correlated is creating a heat map from the result of the Pearson correlation. Furthermore, the variance of a feature can be examined. The smaller the variance, the less likely is it that this feature provides a high amount of

information. A variance threshold is defined to select only high variance features according to that threshold. This method needs to be reviewed closely as it can eliminate features that are relevant nonetheless.

Multivariate time series data is often high dimensional. To further reduce the high amount of features, *dimensionality reduction* is a suggested technique to decrease the dimensions and provide a compressed input data more suitable the machine learning algorithm. Dimensionality reduction may improve the performance of the machine learning algorithm. Additionally, the dimension transformation speeds up the training time and the data table will take up less memory space Géron, 2017, p. 42. A method to decrease the dimensionality and remove outliers from the input data set is *principal component analysis*. This unsupervised statistical method checks if features of the data set are redundant and then projects the data set to a lower dimensional linear space while maximising the variance of the projected data Bishop, 2006, p. 561 . A compressed representation of the data set (without information redundancy, outliers or noise) is gained which acts as the new input to the machine learning algorithm.

After meaningful features have been extracted, the data types of the resulting data set shall be reviewed to further preprocess them.[2] defines the types of data in statistics as follows :

- *Categorical data:* nominal (discrete units) or ordinal (discrete and ordered units)
- *Numerical data:* discrete (can be counted but not measured) or continuous (can be measured but not counted)

To facilitate learning, the input data has to be prepared for the machine learning algorithm. Rashid, 2017, p. 94 suggests that inputs should be scaled to small values from 0.01 to 0.99 or -1 to +1 for neural networks, depending on the problem. Numerical features should be scaled - standardised or normalised. Standardisation transforms the input values to have a mean of 0 and a standard deviation of 1 while normalisation re-scales the values simply to a range from 0 to 1.

[2] *Towards Data Science - Data Types in Statistics* 2019.

2.1.3. Neural Networks

The main machine learning algorithms that are explored in this thesis are *neural networks*, also referred to as *artificial neural networks*. They can solve classification as well as regression problems, but the following explanation will focus on classification only. Neural networks come from a biological inspiration and were first introduced in their modern form by Rosenblatt, 1958 as a concept called the *single layer perceptron*, known as the simplest form of a neural network. The principle idea is to recreate how the human brain works. The name *neural network* comes from neurons in a brain. Biologists discovered that brain cells consist of a network of neurons connected to each other with synapses: The synapses transmit information via chemical transmission, only leading to a reaction if the combined impulse from the synapses are above a threshold Rolls and Treves, 1998, p. 3. The perceptron is modelled similarly to this discovery. Its components are shown in Figure 2.4. One neuron receives signals from multiple other neurons and - based on the weighted sum of the signals - the neuron might send a signal itself. The perceptron takes this idea to form its elementary building blocks Several inputs are weighted and then summed up. The result is compared to a threshold limit. The generated predictive output from this threshold check is binary: 1 if the result exceeds the threshold, 0 if it is below. The perceptron can solve linearly separable problems such as creating a decision boundary for a binary classification problem or modelling logical gates such as AND, OR and NOT. Nevertheless, other researchers discovered certain limitations of the perceptron . For example, it cannot solve the XOR input-output mapping because the perceptron is only capable to fit to linear decision boundaries and the XOR gate is a non-linear pattern. Scientists suggested an extension to the perceptron, defining neural networks how they are used today.

In broad terms, what was added to improve the perceptron was a more powerful activation function to smooth down the output function in contrast to the binary decision used in the classic single layer perceptron. An *activation function* transforms the result into a classification decision, mapping the input values to required output values. Various activation functions exist. The perceptron relies on a unit step activation function while the sigmoid function is a popular activation function for non-linearly separable

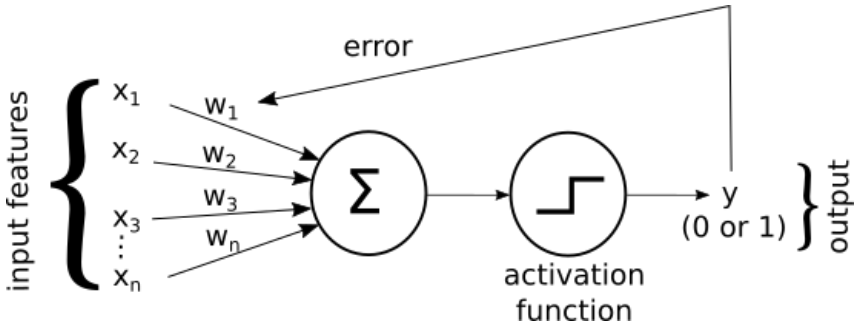
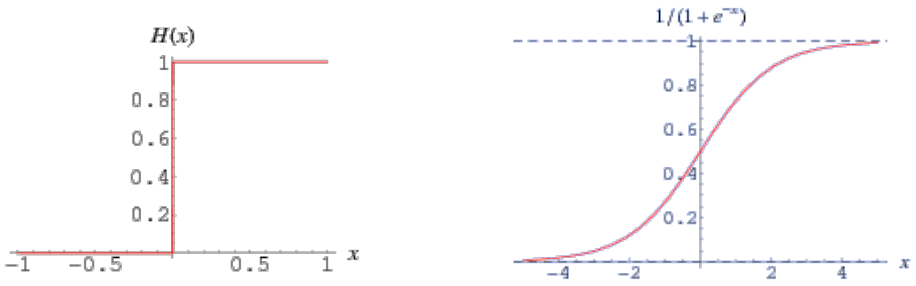


Figure 2.4.: The principal components of the single layer perceptron, the simplest version of a feedforward neural network, are shown in this image. Inputs are weighted and then summed. The unit step function is used as an activation function, generating binary outputs in a range of [0,1]. The error function calculates the error which is then used for backpropagation to update the weights and thus receive a more accurate prediction in the next iteration.

problems. Both function shapes are compared in Figure 2.5. Additionally, multiple layers were introduced to the neural networks to enable them to solve complex problems. The single layer perceptron and the multi-layered perceptron are classified as *feedforward neural networks*. *Feedforward* refers to the fact that this type of neural network does not form a cycle or receive input from an external source. Information is only fed forward to the subsequent layers in the network. In the network architecture shown in Figure 2.5, one neuron is connected with each neuron of the next layer. Therefore this architecture of a neural network is called a *fully-connected* network.

The core components of a modern feedforward neural network are one input layer, one or more hidden layers and one output layer (see Figure 2.6). The interconnected neural units learn and infer rules from observational data. The layers are connected with weights that moderate input impulses. The input layer does not perform any operations on the data, it just serves the data to the first hidden layer. A hidden layer takes the inputs, multiplies weights with the inputs, creates a sum and uses an activation function to generate the output of the neuron. The output layer transforms the data from the last hidden layer to a predictive output for classification. The goal is to find the adequate weights w for the function f that models the relationship between the output labels y_n that correspond to the observation



(a) Unit Step Function (b) Sigmoid Function

Figure 2.5.: Different activation functions are compared in this figure. The *unit step function* or *heaviside step function* shown on the left is an activation function resulting in binary values. The image is taken from Weisstein, n.d.(a). The *sigmoid function* on the right is a popular activation function for neural networks. This function is also used for logistic regression and sometimes referred to as the *logistic function*. It is differentiable and continuous. The image is taken from Weisstein, n.d.(b).

and the input features x_n as in Equation2.1.

$$y_j = f(x_n; w) \tag{2.1}$$

The algorithm of a neural network starts with initialising the weights to 0 or randomly chosen small values. Then, the features of a data point are taken as an input to calculate the weighted sum in each neuron, following the equation in Equation 2.2. A bias is added as an additional constant before the activation function is applied.

$$p_j = -\sum_i w_{ij} * x_i + b_j \tag{2.2}$$

where: x_i = input from neuron i
w_{ij} = weight assigned to the connection between neuron i and j
b_j = bias term

Then the activation function determines the predicted class for this particular sample. The current state-of-the-art activation function is the *Rectified Linear*

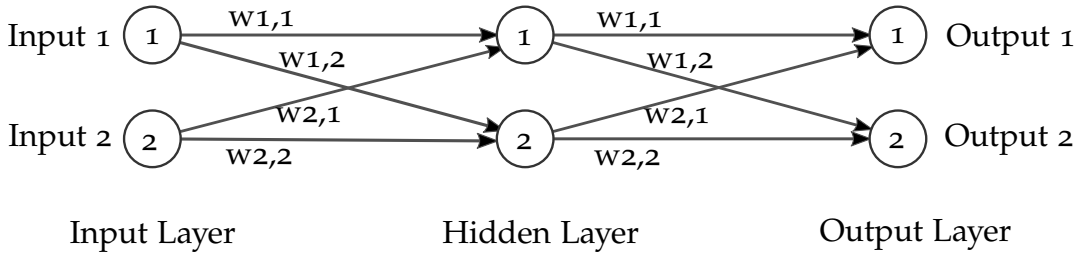


Figure 2.6.: This graph shows the elementary building blocks of a neural network. It consists of an input layer, at least one hidden layer and an output layer. The shown neural network contains only one hidden layer. The nodes are connected by weights.

Unit (ReLU) since they have shown to train better than sigmoid activation functions Patterson and Gibson, 2017, p. 131. The ReLU function is denoted in Equation 2.3. It returns 0 for inputs ¡ 0, but when the input rises above 0, the function models a linear relationship.

$$o_j = max(0, p_j) \tag{2.3}$$

The prediction is checked against the ground truth using a loss function to calculate the error. The goal is to find the weights w that minimise the chosen error function which is done by maximising the likelihood L to find the most likely explanation of the data and find a model that fits best, as shown in Equation2.4.

$$w = arg \max_{w} L \tag{2.4}$$

The error function which was used for the neural network models in this thesis is the *categorical cross-entropy loss* (also called *log loss* or *softmax loss*) since it is the default loss function for multi-class classification[3]. The target labels are one-hot encoded, meaning that the labels are transformed to a matrix. The matrix has the labels as columns and the amount of data samples as rows. For each data sample, there is a 1 marking the correct class

[3] *Machine Learning Mastery - How to choose loss functions when training deep learning neural networks* 2019.

label and a 0 for all other columns. The predicted outputs are transformed in the output layer by the *softmax activation function* (see Equation 2.5). The transformation creates a column vector containing probabilities in the range [0,1]. Each probability states how likely it is that the data sample belongs to a class. Naturally, the sum of all probabilities equals to 1. The class label with the highest probability is the one that is most likely to be true to the class from the observational data. The softmax function basically calculates the ratio of the exponential of a single parameter y_i to the sum of exponential parameters of all values coming from the last neuron before the output layer.

$$S(y_i) = \frac{exp_i^y}{\sum_j exp_j^y} \tag{2.5}$$

where: y_i = single input parameter in vector
$\sum_j exp_j^y$ = sum of exponential parameters of all values in the inputs

The loss function then compares the results from the softmax function to the one-hot encoded target labels. The loss is calculated for each label per sample and then sums the losses up as shown in Equation 2.6. Using the natural log penalises the loss - the more wrong the prediction is, the larger is the loss.

$$J(\theta) = -\sum_{c=1}^{M}(y_{o,c}\, log(p_{o,c})) \tag{2.6}$$

where: M = the number of classes
log = the natural log
y = binary value indicating if class label c is correct observation o
p = predicted probability for observation o is of class c

There is a high amount of possible weight permutations, but no analytical solution. Thus the weights need to be found in an iterative process. In

case the sample is misclassified, the weights are adjusted with the help of a learning rate. The *learning rate* is a scalar representing the rate how quickly or slowly the weights are updated during each step of the iteration. A risk is that the target is overshot when setting the learning rate too high. If the output was predicted correctly, the weights are left as they are. The sample is processed again to train the predictive ability of the model. This iterative process of calculating the loss and then updating the weights is called *backpropagation*. This algorithm processes the weights in a backwards motion and converges on a minimum of the error function. Backpropagation for the models of this thesis used the *stochastic gradient descent algorithm* (see Equation2.7 for the formula) to find the minimum of the chosen differentiable error function Rashid, 2017, p. 73. The basic idea is to check the slope of the function and progress in the direction where the minimum of the function lies. The gradient descent guides the way to the minimum until convergence. It is similar to a hiker trying to climb down a hill in the dark and checking where the slope goes down with a flashlight. Taking big steps down the hill might end up in overshooting and missing the minimum. This can be solved by an *adaptive learning rate*: The learning rate is the size of the steps the gradient descent algorithm takes to reach the minimum of the function Rashid, 2017, p. 70.

$$\text{repeat} \quad \text{until} \quad \text{convergence:}$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.7}$$

where: α = learning rate
$J(\theta)$ = loss function

An *optimiser* specifies exactly how the gradient of the loss function will be used when updating the weights during backpropagation. The optimiser used in the training of the deep learning algorithms of this thesis was Nadam (Dozat, 2016), which combines the Nesterov momentum with the popular Adam algorithm (Kingma and Ba, 2014).

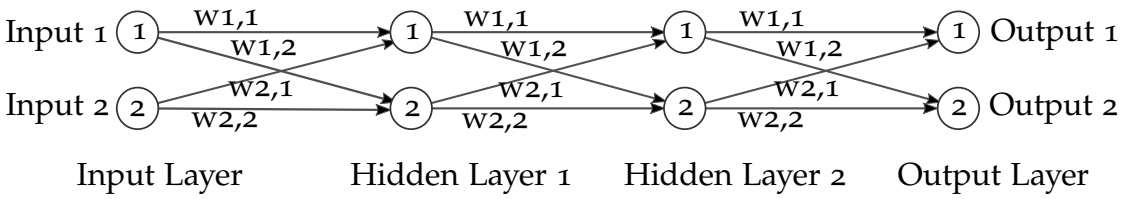


Figure 2.7.: A neural network consists of an input layer, at least one hidden layer and an output layer. The shown neural network has an input layer, two hidden layers and an output layer. The nodes are connected by weights. A neural network with more than one hidden layer is defined as a *multilayer perceptron* and is a *deep learning method*.

During the learning process, a neural network runs several independent training loops with the same input data to increase the accuracy of the model. The gradient descent algorithm starts each time with different initial weights to ensure that the global minimum is found while *climbing down the hill* Rashid, 2017, p. 77. Finally, all the valleys of the gradient descent are compared and the lowest one is taken as the global minimum.

As already explained, neural networks map inputs to target labels using hidden layers to learn the output. A special sub field of machine learning is deep learning. *Deep learning* in the case of neural networks builds a pipeline of various simple transformations, using more than one hidden layer between input and output for learning. A deeper level of feature abstraction is established in the training phase. Deep learning neural networks are defined as *multilayer perceptrons*. A neural network with only one hidden layer is a *non-deep* or *shallow neural network*.

A growing body of literature has investigated deep learning as a new way of solving machine learning problems. The learning ability of deep learning has only come to light recently due to the increasing flow of data and the need to process the data for problem solving. Since problems with large amounts of data are appearing more often now in computer science, deep learning has increased in popularity since it can deal with a large amount of input data. In the following sections, two particular deep learning algorithms for sequence modeling are explored: convolutional neural networks and long short-term memory recurrent neural networks. The choice to use these two types of neural networks next to a plain feedforward multilayered neural

network resulted from the insights gained by the literature review.

Long short-term Memory Recurrent Neural Network

As already mentioned, deep learning is constructed with a deep structure showing a formation of several hidden layers. Other neural network architectures evolved out of the standard feedforward neural network as it is described above. One of them is called *recurrent neural network* (RNN). An advantage mentioned in the literature is that RNNs work especially well for modelling sequence data and can produce a sequence as an output where the last element is the predicted next value for the input sequence. The algorithm is typically applied in speech recognition, natural language processing or stock prediction. An RNN benefits from a sequential memory that can learn from the time steps that have happened in the past. This recurrent factor is built into the RNN cell architecture by adding a loop that passes previous information forward (see Figure 2.8). The thus established looping mechanism stores the information in a hidden state. The hidden state is a representation of the previous inputs and allows information to be propagated from one step to the next, making RNNs are a perfect fit for sequences and lists.

Bengio, Simard, and Frasconi, 1994 found out that RNNs suffer from the *vanishing gradient effect* during the execution of the backpropagation algorithm. This problem occurs when the gradient is becoming smaller and smaller towards the layers in the beginning of the network, causing the magnitude of the the recursive update to the weights to be insignificant. The more steps are processed, the less information from the initial steps is present. The gradient ceases to attribute much to the learning of the early layers of the network due to the gradient update (see Equation 2.8 for the gradient update rule).

$$new\,weight = weight - learningrate * gradient \tag{2.8}$$

This vanishing gradient effect can be compared to the RNN having a short-term memory, preventing the network from learning from long-term dependencies . A *long short-term memory* (LSTM) network is an evolved

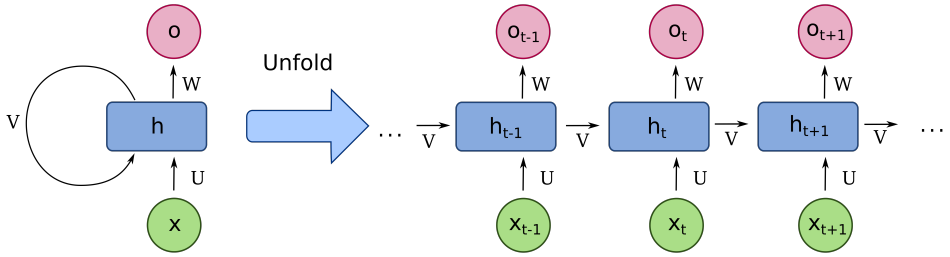


Figure 2.8.: At each learning step iteration, the network is not only fed the next data point x_t but also information from the hidden state h_t calculated by the previous data point. Thus local dependencies can be learned since the output of the previous sample comes back as an input to the next. The components of the RNN are shown from bottom to top: input, hidden state, output. The variables U, V and W are the weights of the RNN. Image taken from Deloche, n.d.(b) (CC BY-SA 4.0).

version of the basic RNN and was first introduced by Hochreiter and Schmidhuber, 1997. Standard RNNs can not learn from time steps that are way back in the sequence Bengio, Simard, and Frasconi, 1994, but LSTMs are able to connect the information and learn from the long-term dependence. Therefore LSTMs work well for long sequences.

The neuron cell of an RNN is replaced by an LSTM cell which only keeps relevant information and additionally can connect information from the fast to the present task, revealing context that has appeared further back in the sequence. An LSTM cell basically contains a more extensive series of matrix computations. At each time step t in the input sequence, the computation of the hidden state is a function taking the input vector x_t and the hidden state coming from the previous time step h_{t-1} (see Equation 2.9). The output of the cell y_t is also gained by a function including the previous state and inputs coming from the current time step t.

$$h_t = f(h_{t-1}, x_t) \tag{2.9}$$

Data in an LSTM is processed sequentially, just like a basic RNN. What is different are the operations inside of a cell which allow for the LSTM

to forget or keep information. The LSTM cell is displayed in Figure 2.9. The core components unique to an LSTM neural network are the cell state (additionally to the hidden state) and gates (forget gate, input gate, output gate). Information is added or removed to the cell state of the neural network, moderated by the gates which are neural networks itself that learn what information is relevant to keep and what can be omitted. The cell state c_t is the long-term state and the hidden state is for short-term memory. In a forward pass, the long-term state travels through the forget gate where some memories are dropped. Then, new memories selected by the input gate are added to c_t. The result from the addition is the output for the long-term state and is passed straight on. Further on, c_t is copied, send through a *tanh activation function* and element-wise multiplied by the output gate, producing the hidden state h_t. Tanh regulates the outputs so that they stay between 1 and -1. The new cell state and new hidden state are transmitted to the next time step. The three gates are results from matrix computations involving the previous hidden state and the feature input from the data point that is fed to the network at the current time step. The main layer similar to the layers in a basic multilayer perceptron analyses the current inputs and the previous hidden state, resulting in the output G_t. The gates moderate what memory is stored and what is forgotten, applying the sigmoid activation function with a range of [0,1]. Since the outputs of the gates are element-wise multiplied with other variables of the cell, the gates regulate exactly what is kept and what disappears since any number multiplied by 0 is 0 and any number multiplied by 1 is the number. Each gate can be seen as a single layer neural network with weight matrix that needs to be learned during training using gradient descent.

Each of the gates possesses a different purpose:

- *Forget gate:* Regulates what memories of the long-term state from the previous cell should disappear by multiplying F_t with c_{t-1}.
- *Input gate:* Decides by using I_t what info from G_t needs to be added to the long-term state.
- *Output gate:* Determines what the next hidden state should be by taking into account what parts of the long-term state are relevant (O_t).

Naturally, it is possible to stack multiple layers of LSTM cells to create a deeper network.

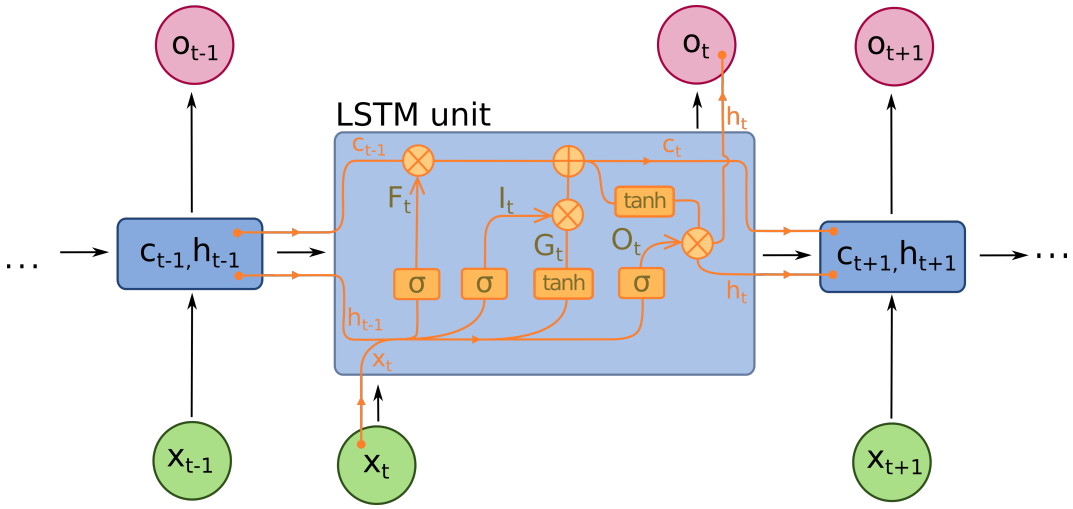


Figure 2.9.: The components of the LSTM are shown from bottom to top: input x_t, hidden state h_t and cell state c_t, output o_t. Image taken from Deloche, n.d.(a) (CC BY-SA 4.0).

Convolutional Neural Networks

Unlike the LSTM network, the *convolutional neural network* (CNN) does not include a recurrent factor. CNNs are especially popular in the problem domain of image classification, but have shown their applicability in the area of human activity recognition as well. As the application area of image classification suggests, CNNs work well with spatial relationships in data. The algorithm can generate a sequence as an output or a class.

A CNN learns *feature filters* which are convoluted with the input to detect edges and thus shapes in the input data. A convolution is a non-linear mathematical operation used instead of general matrix multiplication Goodfellow, Bengio, and Courville, 2016, p. 327. The convolution operation which is essentially an element-wise multiplication and addition can be denoted as in Equation 2.10.

$$s(t) = (x*w)(t) \tag{2.10}$$

where: w = probability density function, in the context called kernel
x = input
$s(t)$ = feature map
$*$ = the convolution operation
t = time index t

The kernel is a two dimensional array of parameters (weights) that are adapted by the training of the CNN by using backpropagation, just as the other neural networks presented in this thesis. The kernel contain zeros and values where information is stored, referred to as sparse weights. When using several kernels, they are stacked on each other and are called a *filter* which is three dimensional. The kernel is usually smaller in size than the input to reuse it to detect meaningful features by re-applying the kernel with a sliding window approach. The result of the kernel applied to the input is called a *feature map*. For time series data, a one dimensional convolution is performed which means that the kernel slides over a single dimension. Time series data just consists of one value in contrast to color images which have three separate channels of red, green and blue. Time series data could be loosely compared to a grey scale image in this context. Several convolutions can be performed subsequently to learn more complex features as the network goes deeper.

To reduce the likelihood of overfitting, a *max pooling* layer (first defined by Zhou and Chellappa, 1988) is often chosen to prevent the feature maps from becoming too receptible to small variations in the data. The algorithm works simply by taking a specified amount of neighbouring values in a matrix, creating a pool of those values and taking the maximum to transfer to the output of this layer. One dimensional max pooling is used in this thesis and works, as the name says, only on values of one single row in the matrix.

Methods to improve Classification Results

When working with neural networks for classification, it is common to conduct multiple experiments by running the training algorithm with different hyperparameters to reach the best model performance. Usually with machine learning there is no definite perfect solution - different models

have to be defined, trained and tested empirically as it is done in the presented work. In this section, the methods that were used to evaluate and improve the performance of the classifiers are discussed. The goal is to provide the reader with a general understanding of the methodologies used to extract the results documented in the evaluation.

There are various ways of grading the predictive ability of a model. The chosen metrics to measure the model performance in this thesis are accuracy, recall, precision, F1 score, Matthews correlation coefficient (MCC) and the categorical cross-entropy loss. The exact formulas for accuracy, precision, recall, F1 score and MCC according to Patterson and Gibson, 2017, p. 38-40 are defined as follows in Equations 2.11, 2.12, 2.13, 2.14 and 2.15.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.11}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.12}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.13}$$

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.14}$$

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.15}$$

where:
TP = true positives (positive class is correctly predicted)
FP = false positives (positive class is incorrectly predicted)
TN = true negatives (negative class correctly predicted)
FN = false negatives (negative class is incorrectly predicted)
$\hat{y}_1, \hat{y}_2, ..., \hat{y}_n$ = predicted values
$y_1, y_2, ..., y_n$ = observed values (ground truth)
n = number of observations

The *accuracy* is the mere proportion of correct guesses and can be misleading in case the classes are unbalanced. Detailed analysis can be achieved by tracking the aforementioned metrics precision, recall, F1 score and MCC. The *F1 score* is the harmonic mean of precision and recall reduced to a single measure. Additionally, a *confusion matrix* is often used to evaluate the prediction results of a classifier. A confusion matrix is basically a table stating the predicted labels and the actual labels for a classifier Patterson and Gibson, 2017, p. 36. This way it is possible to see exactly which classes are classified correctly and which are misclassified. The *MCC* gives a value between -1 and +1, trying to represent the confusion matrix with just a single value. +1 means a perfect prediction of the model, 0 is no better than a random prediction and -1 represents total disagreement between the prediction and the ground truth.

In machine learning, the researcher always searches for the best result - the most fitting model. Examining the metrics can give clues of what goes wrong in the prediction. To improve the model, either the preprocessing can be improved or the machine learning algorithm creating the model can be engineered differently. Improving the preprocessing means going back to exploring the raw input data and adding additional data cleaning such as outlier detection or selecting different features than used in the last run of the machine learning algorithm. Furthermore, *hyperparameters* are parameters for tuning the network to learn from the input data better and faster Patterson and Gibson, 2017, p. 28. Hyperparameters can be optimised by empirically training networks with different settings and choosing the hyperparameters that end up with the best result. Hyperparameter optimisation can be time-consuming because many different combinations of settings exist and each time the model has to be trained from scratch. Testing a number of possible combinations of hyperparameters and evaluating their performance is called *grid search*. Here, the number of possible combinations that have to be examined grows exponentially with every hyperparameter that is added.

From examining different hyperparameters, the programmer ends up with several different models. To choose the best model, the metrics as listed above can be used to assess how well the classification problem is solved. Additionally, the model selection can be based on further information such as over- and underfitting. Both are challenges the model construction is

frequently faced with. *Overfitting* occurs when the model is too complex and thus unable to generalise. The model has basically only learned how to align its prediction to samples from the training data set and is unable to deal with new unseen data samples. Possible solutions are simplifying the model (for example reducing the features), collect more training data (if the data set is sparse) or clean the training data better (remove the data errors and outliers). *Underfitting* occurs when the model is too general and fails to be accurate enough for the particular data set. In this case, a more complex model should be chosen, better features should be fed to the learning algorithm or the set constraints on the model should be reduced. After improving the model, it should be just right - not too general and not too complex. Achieving this state is a struggle that can mostly only be solved by going through various iterations of making changes to the training data, the hyperparameters or the setup of the chosen model. Géron, 2017, p. 71-74

When encountering insufficient quantity of training data and thus overfitting, adding data points based on existing data but with minor alterations can be used as additional training. This process is known as *Data augmentation*. In the field of image detection, data augmentation is performed by transforming the image, for example rotating the image. In the context of time series data, the data points can be downsampled. Where it makes sense, *downsampling* is a method to split the samples into smaller slots of time. For example, a time step of 30 minutes could be downsampled into six slots of five minutes leading to more training data. Minor relevance in the scope of this work is given to data augmentation since samples cannot be artificially created as with images.

The *learning rate* can be optimised to battle overfitting. Many experiments have to be observed to compensate random effects when climbing down with gradient descent. If the learning rate is high, the optimum can be reached faster, but it is also possible that the step is too large and the best answer to the problem is missed Rashid, 2017, p. 154 . A small learning rate will lengthen the training process.

A technique called *regularisation* is a measure against overfitting by constraining the model. One method tested in the practical part of this thesis is dropout. *Dropout* randomly drops a neuron preventing it from taking part in the

forward pass or in the backpropagation to force the neural network to generalise Patterson and Gibson, 2017, p. 63. *L1* and *L2* regularisation add an additional term to the cost function that is minimised. The idea is to penalise larger weights to make them smaller and therefore reduce the complexity of the model.

To achieve faster convergence, *batch normalisation* can be used. The *batch size* refers to the amount of training samples that are used in one iteration (forward pass, then backpropagation) of the training process. Batch normalisation normalises the activations of the previous layer at each batch and reduces the sensitivity of the weight initialisation during training Patterson and Gibson, 2017, p. 264.

To achieve the best result, neural networks are trained several times with the same hyperparameters and the same training data. These runs are called *epochs* and naturally make training take longer Rashid, 2017, p. 155. The idea is that the gradient descent algorithm to minimise the loss function starts randomly at different initial points and might end up in a local minima instead of the global one. Trying out several runs, the global minimum can be found.

Besides tuning the hyperparameters, a direct way of interfering with the training and validation procedure to get a network that generalises well is by introducing *early stopping*. During the iterative process of learning, the validation error should drop with each epoch until it reaches a minimum. The model should not be trained any further because this will lead to the model fitting to the data set even more and thus loosing its ability to generalise. The model will overfit and the validation error starts to increase from there. The training needs to be stopped once the validation error is at its lowest point. This point can solely be found by conducting experiments and monitoring the validation error to find its global minimum. A simple solution is the usage of early stopping to end training once the validation error has not decreased for a pre-defined number of epochs.

Furthermore, the network architecture can be changed up. It is good practise to first try out a shallow network with just a few layers, then a deep network and compare their results. From looking at the performance, the programmer can lean into a direction and try out further network architectures with different hidden nodes according from what was observed in the

previous training runs. There is no definitive answer to how many hidden layers should be used - it has to be explored with experiments of different model variations.

Until here, several approaches to improve the model's performance were discussed. The need for looking into those possibilities was the reason that the deep learning models presented in this work suffered from overfitting. Nonetheless, there is a vast amount of further methods to tackle problems within the model which were not included in this section.

2.2. State of the Art

In this section, cutting edge papers related to the problem domain of multivariate time series classification are discussed. The literature review mainly focused on supervised machine learning algorithms. The solutions to solve the classification task that have already been implemented by other researchers using the same data set from the UCAMI Cup 2018 are explored.

Relevant literature is researched using keywords from the following list:

- UCAMI 2018
- Activity recognition in smart environments
- Activity recognition classification
- Sensor-based activity recognition
- Multivariate time series classification
- CNN activity recognition
- CNN with LSTM

Appropriate search engines for literature research in this domain are Semantic Scholar[4], the sensors section of the MDPI Open Access Journal[5] and Google Scholar[6]. Additionally, the website of the Ulster Institutional Repository[7] at Ulster University has related papers to the problem statement of this

[4] *Semantic Scholar* 2019.
[5] *Sensors - MDPI Open Access Journal* 2019.
[6] *Google Scholar* 2019.
[7] *Ulster Institutional Repository, Ulster University* 2019.

thesis. The papers explaining and analysing the chosen data set where taken from the proceedings of the UCAMI 2018 conference. The relevance of found papers is determined by the year it was published in and by the similarity to the problem domain.

2.2.1. Solutions by other Scientists using the same Data Set

The chosen data set from the UCAMI Cup 2018 has already been analysed by different researchers to settle on a good classification solution. An overview of the algorithms, the features and the testing accuracy is displayed in 2.1. Some of them presented strong results based on the testing accuracy of how many classes were classified correctly.

Karvonen and Kleyko, 2018 approached the problem with a domain knowledge-based solution. They chose to use only input data from the binary sensors out of all four data sources included in the data set. The paper states that conventional machine learning algorithms such as support vector machines do not consider the temporal relationships between activities. The presented solution is an expert system similar to a finite state machine with an accuracy of 81.3%. A finite state machine models a sequence of states and the transitions between these states. The states represent the activities. This type of system has the advantage that the inferred rules of the expert system are easy for humans to understand. Especially in the ehealth sector, the users of this system should be able to comprehend and reproduce how the classification was done. An issue with this type of solution is that once the resident of the smart home changes the daily habits, the system struggles to classify the activities correctly. Another problem with sequences and this data set is that the data was captured in a controlled environment. The test person in the lab might have behaved differently because the person was aware that the activities are recorded. Furthermore, humans might switch their habits or might be interrupted. This means that the sequences modelled in the finite state machine would be out of order and could not be classified anymore. Another factor mentioned by the authors is that the data set is fairly small to provide a reliable input to train the model.

The details of the solution proposed by Lago and Inoue, 2018 describe

Author	Year	Algorithm	Features	Testing Accuracy
Karvonen and Kleyko, 2018	2018	expert system similar to a finite state machine	binary sensors	81.3%
Lago and Inoue, 2018	2018	hybrid model (hidden markov chain and logic model)	binary and proximity sensors	45.0%
Salomón and Tîrnăucă, 2018	2018	weighted finite automata	binary, proximity and floor sensors	90.65%
Jiménez and Seco, 2018	2018	multi-event naive Bayes classifier	binary, proximity, acceleration and floor sensors	60.5%
Razzaq et al., 2018	2018	Filtered-Classifier (Weka tool)	binary, proximity, acceleration and floor sensors	47%
Ceron, López, and Eskofier, 2018	2018	J48, Ib1, support vector machines, random forest, AdaBoostM1 (best result) and bagging	binary, proximity, acceleration and floor sensors	62.77%

Table 2.1.: Overview table of publications using the same data set as in this thesis, ordered by the appearance in the text. The table displays which algorithm was used, which features where selected and how well it performed according to the testing accuracy. The available features originated from four different data sources of the UCAMI Cup 2018 data set: intelligent floor, proximity, binary sensors and acceleration data from a smart watch.

a combination of a probabilistic and a descriptive model. They applied the binary sensor and proximity sensor data as inputs for the activity recognition process. The probabilistic model is based on a hidden markov chain and represents each activity as a state. The transitions of the hidden markov chain model the sequences of activities. Since the test person in the smart home lab switches from activity to activity, the result is a sequence of observations which fits for the application of hidden markov chains. The emission probabilities for the markov chain were calculated using a neural network and considered the mean duration for every type of the 24 activities. Thus it was possible to calculate the probability of staying in the same activity or changing into a different activity. The description-based part of the model relied on the verbal description of every activity given by the data set description. The logic was implemented using Java classes, but due to the difficulty of transforming the activity descriptions into logic, the activities "wash hands" and "wash dishes" were skipped. Based on the description of each activity, it was possible to determine events and use them to mark the end and start of each activity. The overall accuracy was 45.0%. The hybrid model performed poorly on activities with a short duration such as "put washing into washing machine" or "visit in the SmartLab". Both could not be recognised at all by the classifier. The advantage of a description model is that it can classify activities that the model has never seen before since the model relies on the proper description of the class. Nevertheless, if the sensor that marks the end of an activity is not triggered, the end of the activity is not recognised and the state change is completely missed. Then it is impossible for the hidden markov chain model to update the state and change to the most probable subsequent activity.

Salomón and Tîrnăucă, 2018 solved the time series classification problem with the given data set using a semi-supervised machine learning algorithm with a weighted finite automata (similar to the finite state machine by Karvonen and Kleyko, 2018) and regular expressions. Additionally to using the binary senors, their contribution also incorporated the floor sensor data to figure out the test person's position in the room through frequency distribution. One automaton was trained for each segment of the day to determine the flow of activities for the segments. The weights between the states of the state machine were calculated by how often this particular path was taken. With a semi-supervised process, the description of each

activity was gained by first training an automaton per activity, transform it into a regular expression and then hand-tweaking the result. The regular expression denoted which sensors were typically active and in which order, for each individual activity. By combining the transition probabilities of the automata for each segment and the regular expression that represented each activity, the activities could be classified. The proposed system resulted in an accuracy of 90.65%. They argue that the errors in their prediction model were mainly caused by wrong predictions of starting and ending times of the activities. They believe this was due to the human transcribers making errors and additional noise in the data set.

Jiménez and Seco, 2018 presents a multi-event naive Bayes classifier to estimate which activities were performed. A naive Bayes classifier is a probabilistic method based on Bayes' theorem. This approach took the input information from all four data sources. Performing the classification with the test data set reached an accuracy of 60.5%.

In Razzaq et al., 2018 the researchers achieved a 94.0% accuracy for training data and a 47.0% accuracy for the test data set using a FilteredClassifier from the Weka Tool *Weka 3: Data Mining Software in Java* 2019. The large difference between the training and testing accuracy is a sign for overfitting of the classifier. This system used all available sensor data to train the model. The FilteredClassifier was implemented with the default StringToWordVector filter and with random forest as the base classifier. The Weka filter took over the preprocessing - cleaning up the data set and altering it for the classifier.

Ceron, López, and Eskofier, 2018 submitted a solution using the CRoss Industry Standard Process for Data Mining (CRISP-DM) methodology for data mining projects. The six phases of this methodology according to Wirth and Hipp, 2000 are:

1. Business understanding
2. Data understanding
3. Data preparation
4. Modeling
5. Evaluation
6. Deployment

The system by Ceron, López, and Eskofier, 2018 took input data from the event streams of the binary senors, the proximity data, the acceleration data from the smart watch and the location data from the intelligent floor. The used models were J48, Ib1, support vector machines, random forest, AdaBoostM1 and bagging. AdaBoostM1 generated the best results out of the list. A problem that was mentioned by the authors is that the data set has a class imbalance: There are not enough samples of the activities "wash dishes" and "Playing video game" to predict those activities reliably. The activities related to eating meals of the day were aggregated into one single event. For example, "dinner", "lunch" and "breakfast" were summarised as the activity "eating". This step was also performed to the activities related to preparing those meals. The bodily movements of those activities are indistinguishable, only the time of the day is different. Merging those activities increased the classification accuracy by 13%. Even though the 10-fold-cross-validation on the given training set resulted in a classification accuracy of 92.1%, the accuracy for classifying activities of the test data set was only 60.1%, 62.77% without the zero class (class used for samples that are not associated to any out of the 24 defined classes).

2.2.2. Other Approaches for Solving Sensor-Based Activity Recognition

According to Razzaq et al., n.d., the main issue with standard pattern recognition methods (support vector machines, hidden markov models, naive bayes) used for sensor-based activity recognition is that the features are always extracted with hand-crafted approaches, based on experience and domain knowledge of the data scientist programming the model. The extracted features end up being rather shallow e.g. statistical features like mean or variance. This limitation might not be enough to learn complex activities such as the data from the UCAMI Cup 2018. Another problem with activity recognition is that usually the labeled data set is small due to the fact that it had to be labeled by someone and a large amount is needed to learn activities that might not always yield exactly the same sensor data in the same way. Additionally, most standard pattern recognition algorithms learn from static data, but sensor data for activity recognition comes in sequences

of sensor data observations. Each each sample has different sensor events that only make sense when being considered sequentially. Taking into account the nature of the data set of this thesis, deep learning models can perform abstract feature extraction and model building simultaneously. One option of a deep learning model relevant to the problem domain of activity recognition are CNNs. When using time series data, CNNs can leverage the abilities of detecting local patterns Razzaq et al., n.d.

Hammerla and Plötz, 2016 explored several deep learning algorithms for human activity recognition using wearables in their paper. Different benchmark data sets were used to compare deep feed-forward neural networks, CNNs and recurrent neural networks based on LSTM. Their results show that bi-directional LSTMs performed significantly better than other state-of-the art classification algorithms.

3. Use Cases & Requirements

The classification algorithms for activity recognition presented in this thesis are intended to be used in a smart home environment. In this chapter, the use cases and requirements for a hypothetical smart home based on the proposed algorithm proposed in this thesis are analysed.

3.1. Use Cases

This section includes three different use case diagrams to demonstrate in detail how a user might interact with a system based on the proposed classification of this thesis. The presented use cases are related to currently relevant topics like the growing percentage of elderly people in the world's society (ehealth, eldercare), saving energy and efficient disposal of waste (waste management).

The general idea is to classify activities of daily living with data from a smart home, analyse the data, track anomalies and take action accordingly. The tables 3.1, 3.2 and 3.2 describe the potential use cases in detail. The description explains the use cases using a Sensing-Logic-Action approach. Sensing describes the data sources for receiving the sensor information, logic characterises the analysing and decision making part of the system and action defines what action the system will take in this specific use case.

3.2. Requirements

This section lists the requirements the smart home that uses the proposed algorithm is confronted with.

Title	Turning off the heating after the inhabitant goes to sleep
Problem description	The inhabitant of the smart home goes to sleep and wants the heating to be off to save energy.
Sensing	Location data from intelligent floor, binary and proximity sensor data (for example proximity sensors of the bed), acceleration data from wrist watch.
Logic	The person lies down in the bed and stays there for more than one minute. The logic part of the system recognises that the activity "go to the bed" has started.
Action	The smart home system turns off the heating in case it is on.

Table 3.1.: A use case for turning off the heating once the inhabitant goes to sleep.

Title	Tracking meal preparation for the elderly
Problem description	An elderly person might not be able to prepare a meal by themselves or forget to have a meal regularly.
Sensing	Location data from intelligent floor, binary and proximity sensor data (for example magnetic contact and proximity sensors for the fridge), acceleration data from wrist watch.
Logic	A segment of the day (morning, afternoon, evening) ends without the activity of eating which means that this activity has been skipped. The involved activities are Act05 "breakfast", Act06 "lunch" and Act07 "dinner" .
Action	A delivery service is notified to provide a meal for the inhabitant.

Table 3.2.: A use case for tracking meals using the proposed system.

Title	Efficient waste management in the home
Problem description	The activity "put waste in the bin" is monitored to ensure that waste is picked up once the waste bin outside of the smart home is full.
Sensing	Location data from intelligent floor, binary and proximity sensor data (for example magnetic contact and proximity sensors for the trash), acceleration data from wrist watch.
Logic	The logic part of the system tracks how often this activity happens and when the waste collection was last called. The related activity is Act15 "put waste in the bin".
Action	The related action is to call waste collection after a pre-defined number of occurrences of this activity.

Table 3.3.: A use case for tracking waste disposal using the proposed system.

3.2.1. Functional Requirements

The functions that the system must provide can be multifaceted and individually adapted to the inhabitant of the smart home to improve the quality of living by automation and assistance. Additionally, the functionalities depend on the sensors that are installed in the home and the according smart devices acting on the activities. Proposals for potential usage scenarios that the smart home could be faced with are listed above in the Section 3.1.

3.2.2. Non-Functional Requirements

An IoT platform must fulfill a high number of non-functional requirements. First and foremost, the classification algorithm used for the activity recognition must be reliable and robust. Since the era of artificial intelligence just started, the society might have doubts about the trustworthiness of artificial intelligence. Often artificial intelligence is mentioned as a black box for its users. Thus peeking into the reliability of the decision making process of a system is especially important when the computer must decide in situations which have a significant impact on a human's life such as in the health sector. Therefore the decisions of the system must be accurate, transparent and comprehensible which is hard to achieve in the case of high level artificial intelligence algorithms.

Moreover, the system needs to be able to processing real time data and ensure the connectivity and mutual compatibility of different protocols and standards. IoT applications are set up as an inter-connected network to be able to realise communication between the devices, the data storage service and the actual application service. In the case of this thesis, sensors detect activities and send the data to a destination in the embedded system. Due to the many possibilities of sensors that could be used in the context of activity recognition, the system must be able to manage communication with a range of different devices. When dealing with a vast number of distributed devices, fault detection should be implemented to prepare for when sensors are sending wrong data or have low battery or suffer from some other technical fault. The network's fault tolerance also ties in with the requirement of reliability.

According to Priyadarshini, Bagjadab, and Mishra, 2019, the main security requirements for an Internet of Things system are:

- Availability of service: The smart home has to constantly be available and therefore be protected against denial of service attacks.
- Authentication, authorisation, accounting: Only people who own the privilege to enter the system should be able to.
- Data privacy, data confidentiality, data integrity: Since smart homes are possibly dealing with sensible data, it is essential that the captured data is being protected by proper encryption mechanisms.
- Energy efficiency: When dealing with devices intended for long-term use, it is important to use power efficient products with a longevity.

Another important quality attribute that needs to be discussed in relation with the topic of smart environments is scalability. In IoT, scalability is a considerable challenge due to the increasing complexity of using a high number of different devices in a network and ensuring the flow of data. An IoT system generates a massive amount of information, therefore an adequate architecture must be set up to store the data and process it.

Due to the high number of computations that are associated with deep learning, the proper hardware is necessary to speed up the training of the algorithm. For example, instead of central processing units (CPUs), graphical processing units (GPUs) can be used to make the training fast. Evidently, the usability of control panels or other types of inputs to control the smart home system has to be checked, for example by conducting usability tests with real users.

4. Data Preparation

This chapter focuses on the preparation of the data that serves as an input for the machine learning algorithms presented in this thesis. The chapter describes how the raw data was initially captured. Then, the performed preprocessing steps are explained. The choices for the preprocessing steps are based on the theoretical concepts already explained in this thesis. The same input data set is used for the machine learning algorithms. An exploratory analysis of the data set gives the reader a chance to get to know the data. The chosen programming language for preprocessing the data is Python[1], including the powerful tools Pandas[2] for manipulating the data structures and matplotlib[3] for the plots.

4.1. Data Collection

The sensor data used for this project is gained from the data recorded inside an experimental smart lab setup at the University of Jaén. The data set was published for the UCAMI Cup which was a competition for human activity recognition. A single resident, a 24 year old student, carried out 24 different activities of daily living in the smart lab.

The smart lab tries to reproduce the environment of a real life apartment and to capture what the resident is doing inside of the lab. The smart lab measures 25 square meter and has five different areas inside of it: entrance hall, kitchen, living room with a work desk and bedroom with an integrated bathroom (see Figure 4.3 for the interior of the lab). Several sensors have

[1] *Python Programming Language* 2019.
[2] *Pandas - Python Data Analysis Library* 2019.
[3] *Matplotlib - Plotting in Python* 2019.

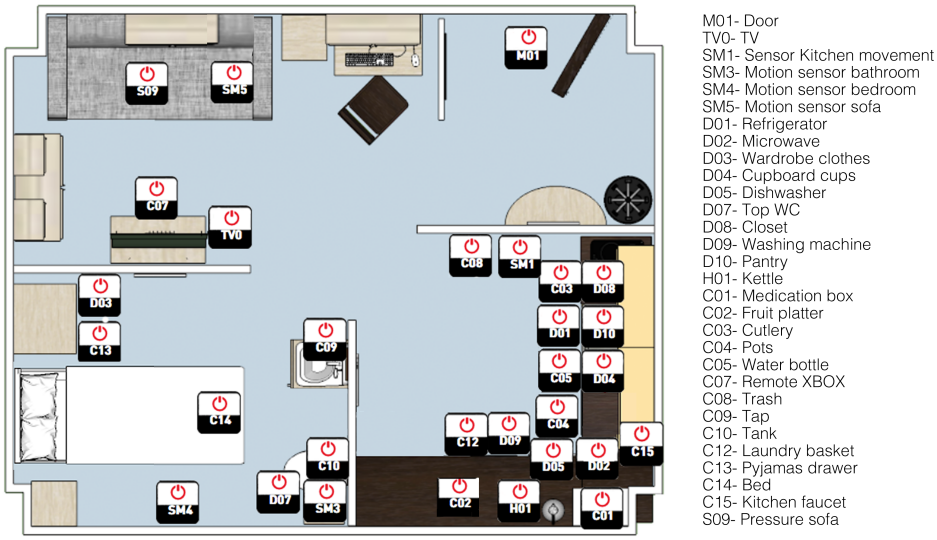
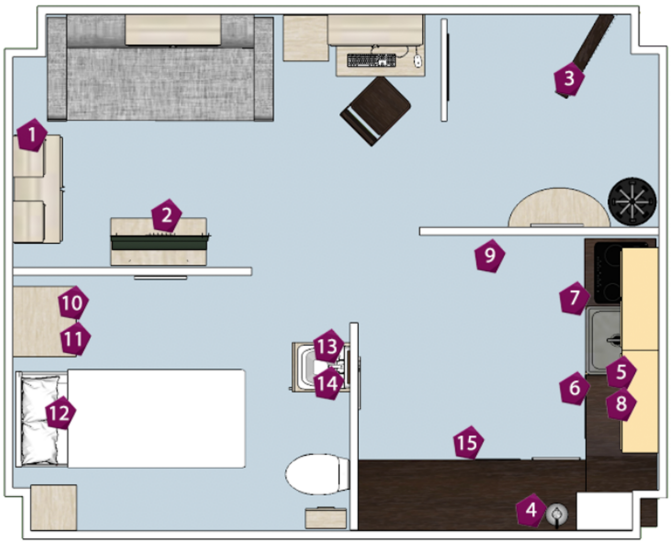


Figure 4.1.: Layout of binary sensors in smart home lab. Reprinted from Espinilla, Medina, and Nugent, 2018 (CC BY-NC 4.0).

been placed inside of the lab in order to record the resident's actions. The sensors collecting the data are deployed in close proximity or even directly attached to objects of interest. The four types of data sources included in the UCAMI data set are:

- 30 binary sensors transmitting a binary value for either magnetic contact, motion or pressure (see Figure 4.1 for their layout)
- Proximity data between a smart watch worn by an resident and a set of 15 Bluetooth Low Energy (BLE) beacons positioned as can be seen in Figure 4.2
- Acceleration data from 3 axes generated by a smart watch
- Location information provided by an intelligent floor with 40 tiles (see Figure 4.3)

As the layouts of the deployed sensors show, the devices capturing the data are stationary except for the acceleration data coming from the wrist watch. The objects are placed close to areas inside the smart lab that are related to the activities defined in Table 1.1.



1-Book
2-TV Controller
3-Entrance door
4-Medicine box
5-Food cupboard
6-Fridge
7-Pot drawer
8-Water bottle
9-Garbage can
10-Wardrobe door
11-Pyjamas drawer
12-Bed
13-Bathroom tap
14-Toothbrush
15-Laundry basket

Figure 4.2.: Layout of proximity sensors in smart home lab. Reprinted from Espinilla, Medina, and Nugent, 2018 (CC BY-NC 4.0).

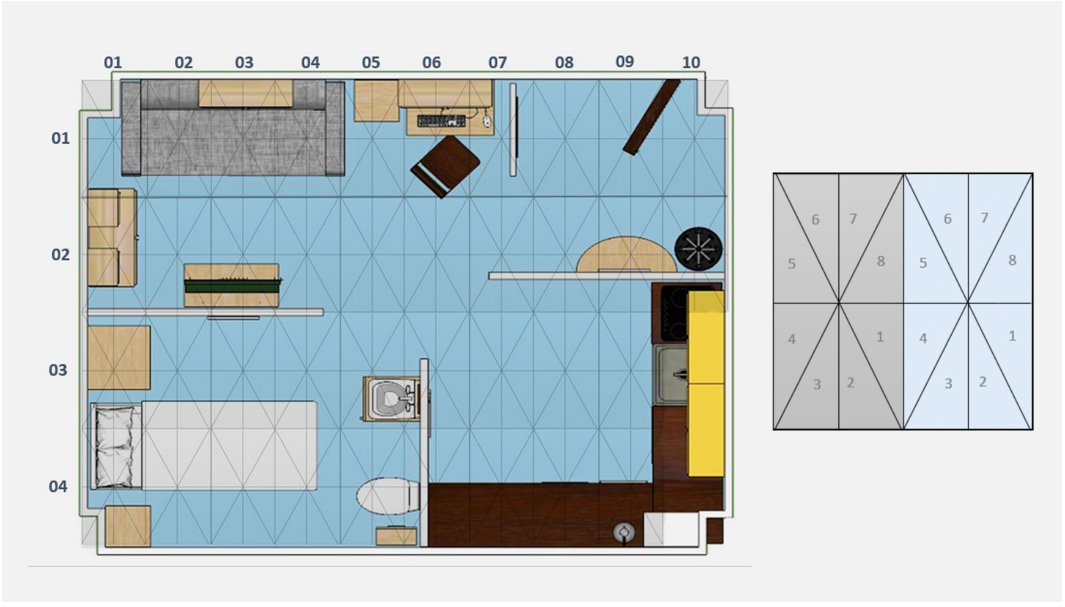


Figure 4.3.: Layout of intelligent floor modules in smart home lab. Reprinted from Espinilla, Medina, and Nugent, 2018 (CC BY-NC 4.0).

The basic idea of creating the proposed system is to develop techniques and tools to provide solutions for improving assistance in smart homes. Using a model for activity recognition can predict what kind of activity the resident of the smart home is performing. Depending on the activity, actions can be taken during the activity or after it is over. For example, the smart home can assess how often the resident has taken the trash out. After a specific amount of time, the smart home system can alert the waste collection to collect all the trash bags. The description of the activity "put waste in the bin" is: "This activity involved the resident going to the kitchen, picking up the waste, then taking the keys from a small basket in the entrance and exiting the Smart Lab. Usually, the resident comes back after around 2 min, leaving the keys back in the small basket." The related sensors to this activity are the binary sensor for the magnetic contact on the trash bin and on the door, the intelligent floor tiles in the kitchen and in the entrance and the proximity sensors on the door and garbage can. There are eleven samples of this activity in the training set and four in the test set.

4.2. Exploratory Data Analysis

When comparing frequencies of each activity from the training set and the testing set in Figure 4.4, it is clear that the classes are unbalanced since there are high differences in the frequencies. 169 activities were recorded in total in the training set. The activity that is seen most often in the training set is "Brush teeth" with 21 counted occurrences. The activities "Relax on the sofa", "Visit in the SmartLab" and "Play a video game" only occurred once in the training set. In the testing data set of only three days worth of data, there are 77 activities. Two activities do not occur in the testing data set: "eat a snack" and "put washing into washing machine". Furthermore, there is a significant difference in the frequencies between the training and the testing set. The activity "relax on the sofa" which is only present once in the training set, was tracked eight times in the testing set.

Between the activities, there are times where sensor data is present, but no activity is assigned. When preprocessing, those samples are labelled with "no activity" in the data set. The "no activity" samples were omitted of the

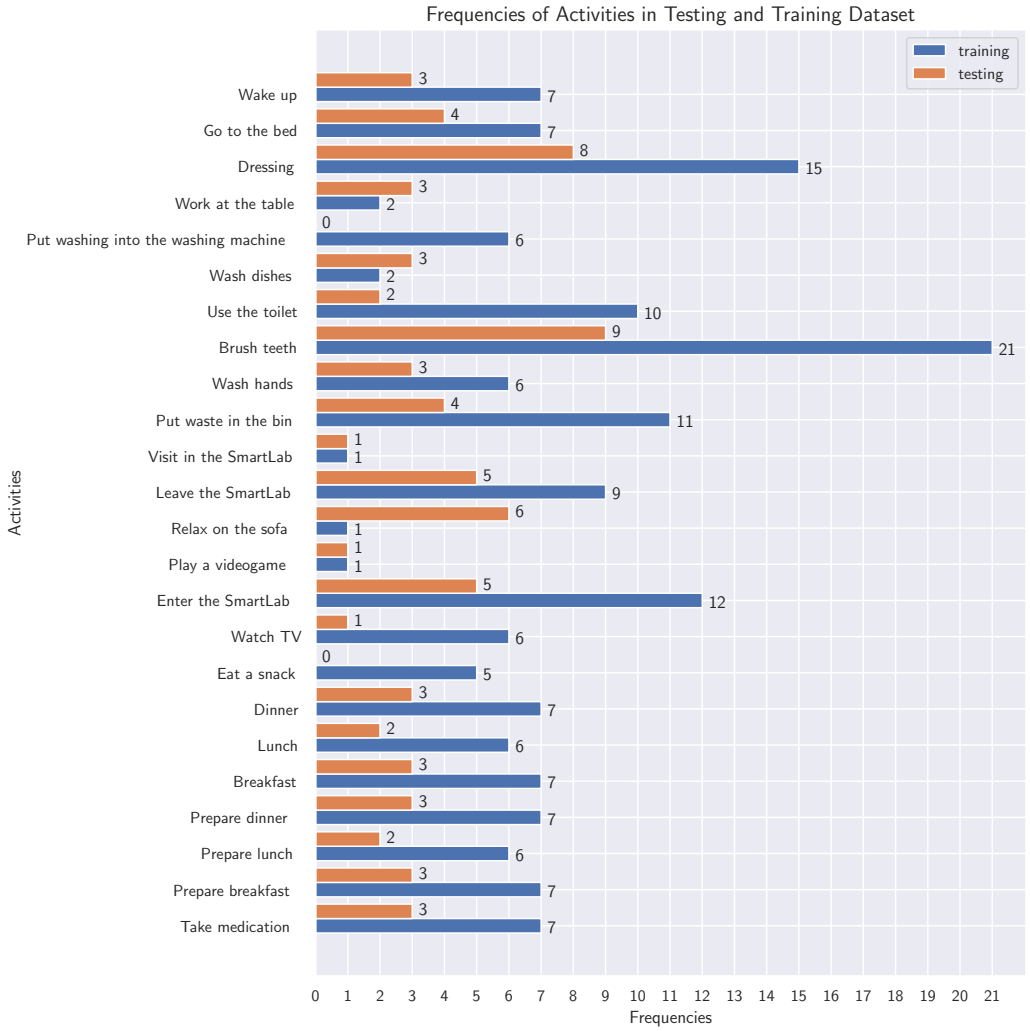


Figure 4.4.: Bar Chart visualising the frequencies of each activity in the training and testing data set. It is visible that there is an imbalance of the frequencies of the activities.

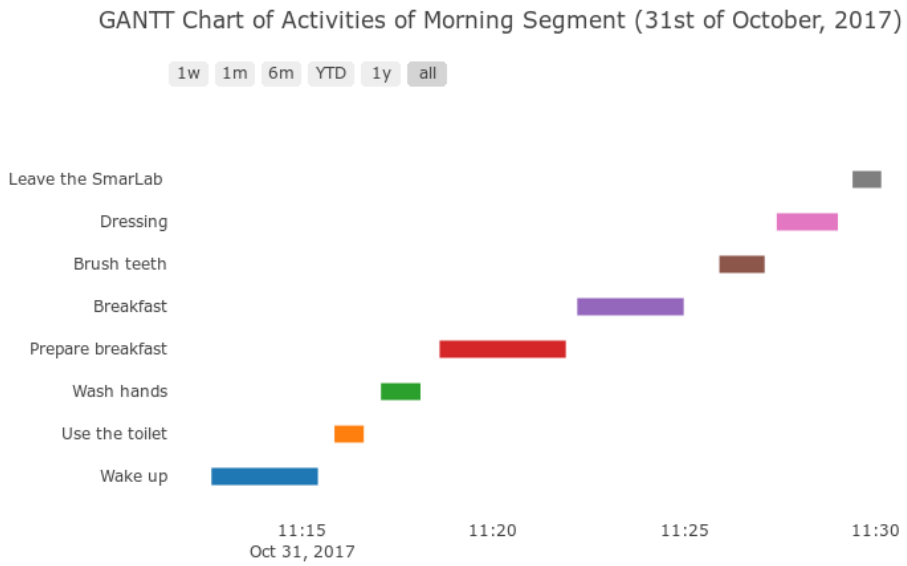


Figure 4.5.: This GANTT Chart shows which sequence of activities the resident performed in the morning segment of the 31st of October, 2017.

data set in the end because they barely occur in the testing set since the labelling process worked differently there. Figure 4.5 shows an example of the activities in the morning segment of the 31st of October, 2017. The activities are happening in sequential order, never in parallel. In between the recorded activities, there are gaps that result from sensor data with timestamps that are not assigned to any activity. The GANTT chart shows a typical morning: The resident wakes up, uses the toilet, washes his hands, prepares breakfast, eats breakfast, brushes his teeth, dresses himself and then leaves the smart lab. The activities "wake up", "prepare breakfast" and "breakfast" itself take the longest time. The shortest activities from that morning are "leave the smart lab" and "use the toilet".

In Figure 4.6 the durations of the activities taken from the training set are visualised. A few outliers are visible in the box plot. The activities that tend to result in the longest durations are "put washing into the washing machine", "work at the table" and "watch TV" Nevertheless, the

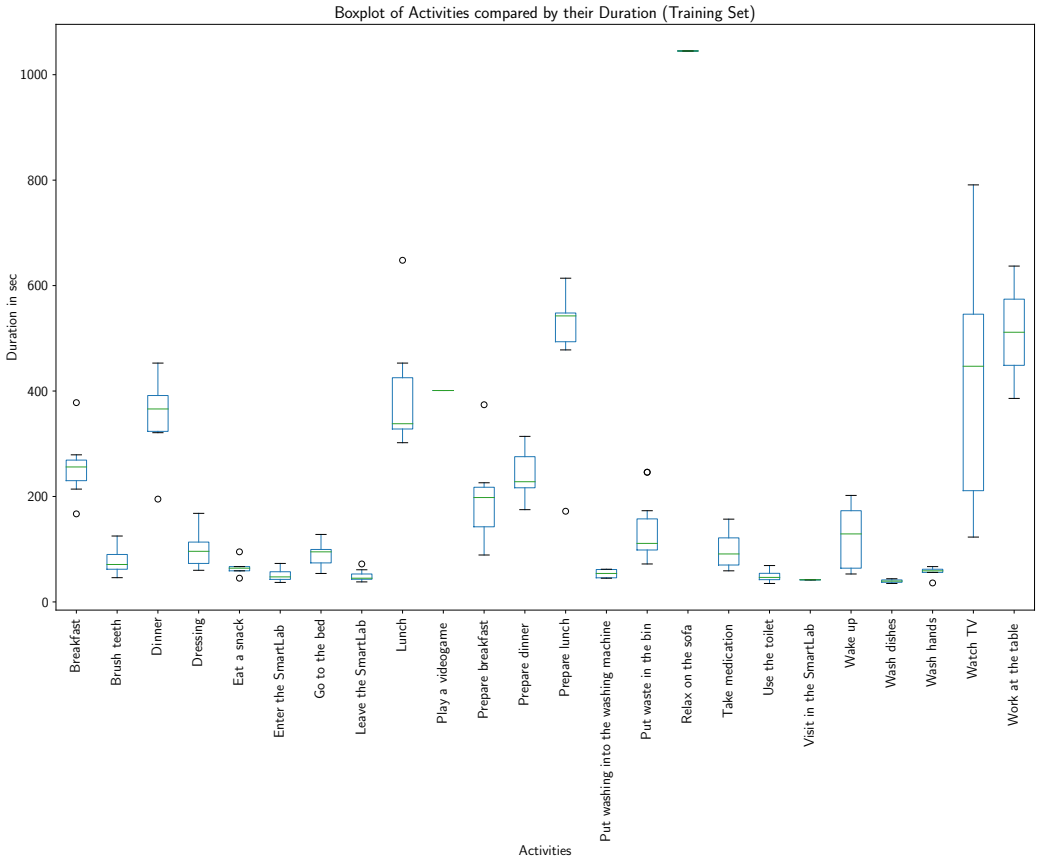


Figure 4.6.: The durations per activity in the training data set are shown as a box plot.

activity "watch TV" has a large interquartile range which indicates that there are various samples with highly fluctuating durations for this activity in particular.

4.3. Data Preprocessing

Before the data is fed to the machine learning algorithm, the raw data needs to be preprocessed. The preprocessing pipeline is displayed in Figure 4.7. First the sensor data needs to be synchronised into samples representing a slot of a predefined time. Then, the data has to be checked for categorical values, e.g. text data can be transformed into nominal numbers. The emerging features should be standardised and normalised where it makes sense. Afterwards, feature selection is performed which results into the preprocessed data ready to be used for the machine learning algorithm.

The data is split into a testing and a training set: Seven days worth of data were used for the training, three days were used for the testing. The raw data set is segmented into different times of the day - morning, afternoon and evening. The testing data is preprocessed in the same way as the training data.

4.3.1. Sensor Data Alignment into 5 Second Samples

The sensors have different frequencies at which data is being transmitted. The highest amount of sensor information comes from the acceleration data source, collected with a sample frequency of 50 Hz. Because the frequencies of the data are not the same, the sensor data needs to be aggregated to form proper input samples. First, each of the data sources is merged into samples of a duration of five seconds. This step is done because it is easier for a model to recognise an activity with a broader time slot that has more information stored to hint at what activity could be performed currently. Different durations were tested, but five seconds seemed to be a good length for each sample.

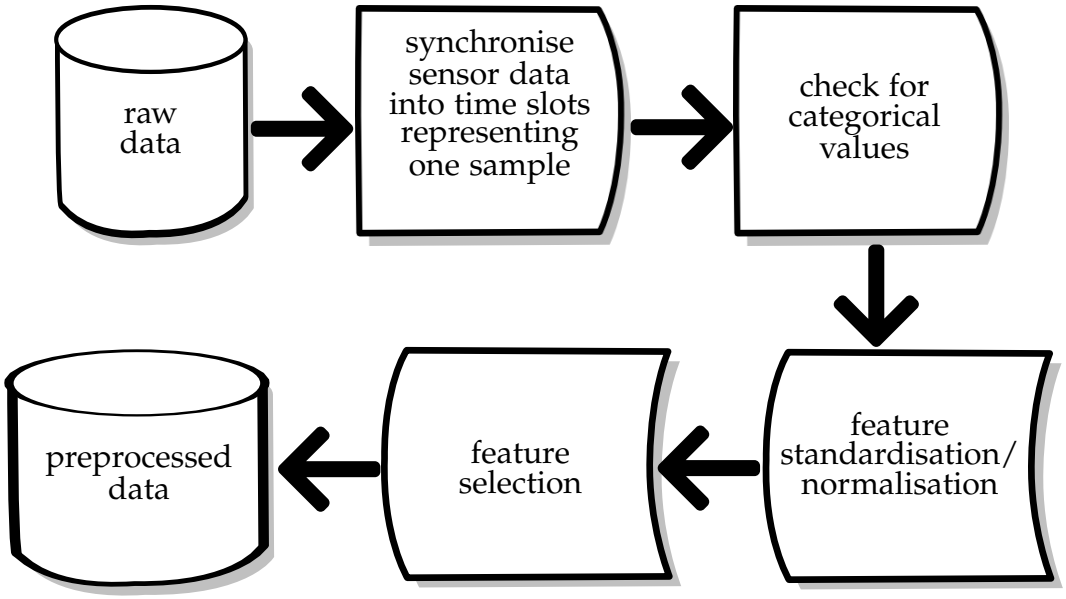


Figure 4.7.: The raw data is preprocessed before the data is fed to the machine learning algorithm. First the sensor data needs to be synchronised into samples representing a slot of a predefined time. Then, the data has to be checked for categorical values, e.g. text data can be transformed into nominal numbers. The emerging features should be standardised and normalised where it makes sense. Afterwards, feature selection is performed which results into the preprocessed data ready to be used for the machine learning algorithm.

Sometimes it happens that multiple values of the same sensor are present for the same five second slot. Then the sensor data has to be aggregated to a single value. The floor data is transformed into one feature per floor tile module and is set to 1 if there is sensor data available for that particular time slot. Otherwise the value is 0 meaning there is no sensor data coming from a module. The binary sensors and proximity sensors are added as one feature per device. The binary sensors already come in a binary format, 1 signaling that there is motion, pressure or no magnetic contact present. The value stays at 1 for all subsequent samples until a 0 is received from the sensor meaning that there is no motion, pressure or missing magnetic contact anymore. If there are multiple entries for the same object in the same time slot, the value is summarised to 1 in case there is a 1 in the merged samples coming from the sensor during that slot. Otherwise the value for that object is 0. The proximity sensor data is measured using a *Received Signal Strength Indicator* (RSSI) which translates to how close an object with a proximity sensor attached is to the wrist watch worn by the resident. Thus, the proximity data is added with 0, meaning no proximity at all. Due to the quality of the signal, the RSSI is preprocessed in the way that any measurement greater than -97 means proximity - the higher the value, the closer the distance to the object placed inside the apartment. If the RSSI measure gives evidence for proximity, the value for the respective device is 1, otherwise it is 0. When merging the proximity data into the slots, 1 is used if there is a 1 present in the data to aggregate. The acceleration data is included with the three axes X, Y and Z, each as their own feature. The mean of the sensor values is taken when merging the data into the five seconds slot.

After resampling the existing sensor data into five seconds slots, the data from all four data sources is merged using the continuous timestamp from the captured sensor data for alignment. If there is sensor data missing for any of the floor, binary sensors or proximity features, 0 is assigned to this value since no data being present is a high sign that there is nothing of interest happening around those sensors. For the acceleration data, the last valid observation is propagated forward to fill missing data.

Since the activities related to the different meals of the day depend strongly on the segment (morning, afternoon, evening) during which the data was captured, the segment is added as its own feature. By adding the segment

to the features, the model should be able to differentiate between breakfast, lunch and dinner. After adding all the features, the activity label is added to the data table according to the timestamp. When there is sensor data present, but no activity assigned to it, the sample is labelled with "no activity" which can be seen as its own, additional class added to the 24 activities of daily living presented in in Table 1.1. Since the samples of the "no activity" class mostly occur in the training set but are barely present in the testing set, this class is eliminated. The imbalance of the presence of the "no activity" happened due to different ways of labelling the data.

4.3.2. Normalisation

The features from the binary sensors, proximity and from the intelligent floor are binary making it irrelevant to perform standardisation or normalisation. The only ordinal data in the data set is the acceleration data and the segment of the day. This data sources are normalised using the MinMaxScaler from the scikit-learn preprocessing class which scales the values in a range between 0 and 1. An experiment showed that using no normalisation does not change the outcome of the algorithms significantly. The idea of normalisation is to bring the features into a similar range which can lead to faster convergence.

The preprocessing steps result in 5.344 samples for training and 2.920 samples for the testing.

4.3.3. Feature Selection

After merging the sensor data into five second slots with inputs from all four data sources, it becomes apparent that the original data set includes a great amount of features and an extremely sparse input matrix (see Figure 4.8). Feature selection is performed to evaluate which features actually bring valuable information for the model creation. A Pearson correlation is calculated to check if certain features in the training set have a linear relation. When looking at the result of the Pearson correlation, it is noticeable that the binary sensors "kitchen faucet (C015)", "cutlery (C03)"

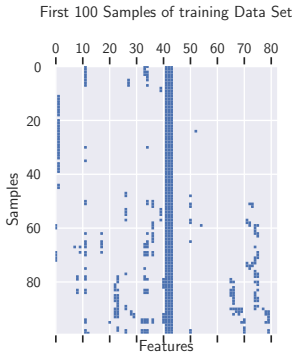


Figure 4.8.: This plot shows the first 100 samples of the training data set. The plot visualises the non-zero values of the 2D input array. The data set is sparse, only the acceleration data is present in every sample since it was collected at a high frequency.

and "laundry basket (C012)" are not present in the training set and neither in the testing set. The sensor "Remote XBOX (C07)" is only present in the testing set, with an occurrence of 1 only once. Due to the fact that the model will not be able to learn anything from those sensors, the four features are removed. The resulting heatmap of the Pearson correlation of the features in the training set is visible in the Figure 4.9. When analysing the Pearson correlation even further, it is visible that the adjacent floor tiles have a minor association with each other. The sofa pressure sensor (binary sensor) in the living room and the proximity sensor of the kettle in the kitchen have a very high association of 92.6%. This is interesting since they are in completely different spots in the apartment. Some features from sensor data collecting devices that are placed close to each other in the smart lab show an association of medium strength, for example the proximity sensors from the bed and the pyjamas drawer.

Since there are 40 modules for the intelligent floor which is quite a large number of features, it was tested if the number of spatial features could be reduced by collapsing those features into rooms (entrance, bedroom, kitchen, living room). The aggregation was applied according to the drawings of the tile arrangement. This feature reduction results in a lower accuracy and is therefore not used in the final run of the machine learning algorithm. After analysing the data from the floor tiles further, it becomes visible that there

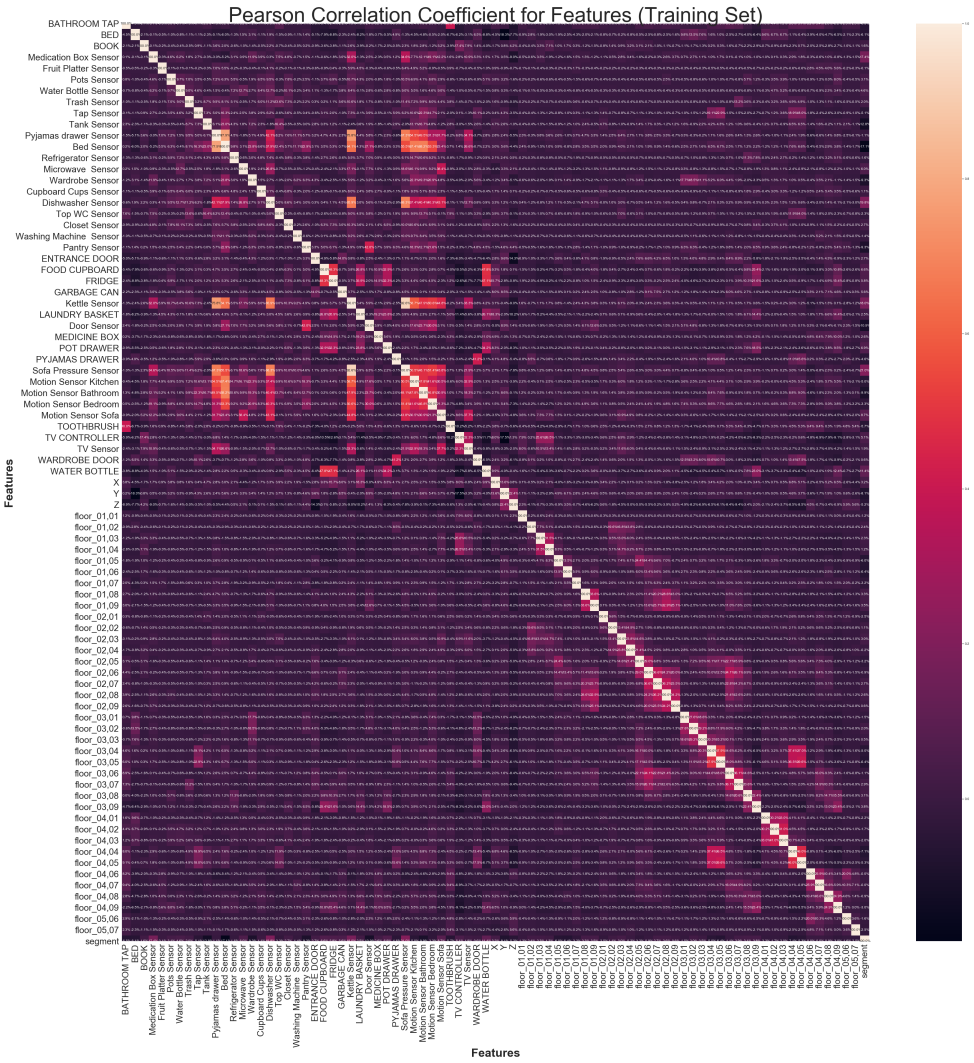


Figure 4.9.: Heatmap of the Pearson correlation of the features training data set. Just by looking at the colours of the matrix it is apparent that only a few features show a correlation.

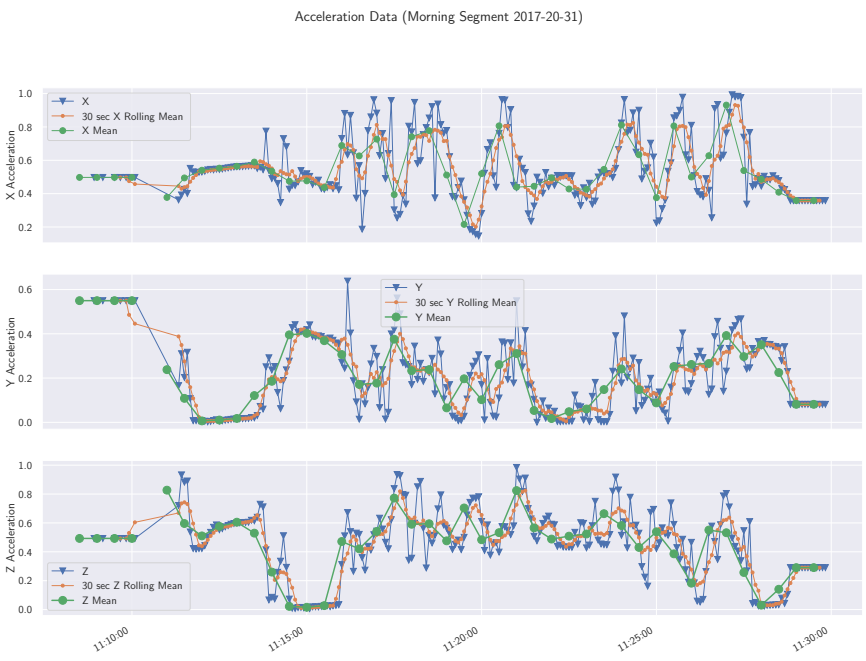


Figure 4.10.: Acceleration data from a wrist watch. All 3 axes (X, Y and Z) are shown with the mean and a rolling mean of 30 seconds.

is never any data coming from the edge tiles near the door: tile “01,10” and “02,10”. Both are excluded from the data set.

Taking the mean of the acceleration data as an additional feature smooths down spikes in the acceleration data while keeping the trend as can be seen in Figure 4.10. This step influences the performance of the classification by a small decrease of the accuracy percentage which is why this statistical feature is not added. Another idea could be to add a rolling mean (aggregating multiple data points to one mean value) of a certain amount of seconds as another feature as it is visualised in the plot.

The feature selection process results in the following 83 features being used for the algorithms described:

- Binary senors: only 26 sensors where selected (binary)
- Intelligent floor: 38 modules (transformed to binary input data) (binary)
- Acceleration from a wrist watch: X, Z and Y axis (scaled between 0 and 1)
- Proximity data from 15 different objects (scaled between 0 and 1, 1 meaning the highest proximity)
- The segment of the day: morning (0), afternoon (1), evening (2): This feature is particularly important to distinguish between breakfast, lunch and dinner due to the time of the day.

The samples are related to each other. To represent the sequential dependence of the samples, lagged features from the sample before are used. This procedure adds the features of forty five second time steps of the previous samples, resulting in 3.403 features per sample. Thus one sample has features from the last 3.33 minutes included.

To reduce features and improve the accuracy, principal component analysis is performed on the data. Nevertheless, the feature reduction only worsens the accuracy and is therefore not chosen as a preprocessing step for the machine learning pipeline of this project.

5. Chosen Classification Algorithms

The problem presented in this academic work is a classification of human activities based on multivariate time series data coming from sensors. The formal problem definition is as follows: For each $(x_i, y_i) \in T \times C$ where x_i is a time series sample observed at time step t originating from the multivariate time series T and y_i is the class label from the class set $C = \{1, 2, 3, ..., 24\}$. Each class in C is an activity of daily living. One sample can only belong to a single class. The task of human activity recognition is to construct a classifier that models a function such as in Equation 5.1.

$$\varphi : T \times C \rightarrow \{1, 2, 3, ..., 24\} \tag{5.1}$$

This chapter focuses on the implementation of the chosen algorithms to solve the problem. The chapter starts with a short description of the implementation tools. Further, twelve different machine learning algorithms based on the already explained theoretical concepts are presented. See Chapter 6 for the evaluation of the their predictive abilities. Nine of the machine learning algorithms are chosen as a baseline comparison. Their results can be compared to other researchers using the same data set and algorithm. Furthermore, the development of the three deep learning models chosen as novel solutions to the problem is described. The hypothesis formed by the literature review of the thesis is that they will show a better predictive performance than the standard algorithms.

5.1. Tools

The chosen programming language is Python[1] including the powerful tools Pandas[2] and NumPy[3] (preprocessing) and scikit-learn[4] (standard classification algorithms and multilayer perceptron) as well as Matplotlib[5] and Plotly[6] (plots for visualisation). The neural network library Keras[7] is used for the CNN and LSTM neural networks. Keras is based on the machine learning platform TensorFlow[8] . The hyperparameter optimisation for the Keras models are implemented using Talos[9]. Node-RED[10] and Flask[11] are used to simulate an IoT system to predict in real time. The application of each tool and its version is listed in the appendix B.

5.2. Standard Classification Algorithms from scikit-learn

A list of nine basic pattern recognition algorithms from the scikit-learn library in Python is chosen to test their predictive accuracy on the human activity recognition data set:

- Logistic Regression
- Gaussian Naive Bayes
- Decision Tree
- Support Vector Machine (radial basis function kernel)
- K-Nearest Neighbors

[1] *Python Programming Language* 2019.
[2] *Pandas - Python Data Analysis Library* 2019.
[3] *NumPy - Python Package for Scientific Computing* 2019.
[4] *scikit-learn - Machine Learning in Python* 2019.
[5] *Matplotlib - Plotting in Python* 2019.
[6] *Plotly - Python Graphing Library* 2019.
[7] *Keras - Library for neural networks* 2019.
[8] *TensorFlow - Open Source Machine Learning Platform* 2019.
[9] *Talos - Hyperparameter Optimization for Keras Models* 2019.
[10] *Node-RED: Low-code programming for event-driven applications* 2019.
[11] *Flask - Python web framework* 2020.

- Random Forest
- Bagging
- Extra Tree
- Gradient Boosting

These machine learning algorithms are selected to compare their predictive abilities with the deep learning algorithms. Some of the options were already tested by other researchers who used the same data set. This way it is possible to validate the preprocessing choices that are made in this thesis and analyse how the results are different from the metrics the other researchers found. The algorithms are executed with the parameters as shown in Listing A.1 in the appendix. Since logistic regression proves to be the best algorithm when analysing the metrics, the hyperparameters of this algorithm are optimised using grid search to achieve the best possible performance.

5.3. Deep Learning Algorithms

Recent development in artificial intelligence explore deep learning and show how neural networks can prove as more powerful solutions than non-deep algorithms. In addition to the standard classification algorithms listed earlier, three deep learning algorithms are selected to analyse their predictive ability: MLP, LSTM and CNN. The classic MLP is chosen since it is an easy start into deep learning. From the literature review, several other researchers mentioned that LSTM networks are a perfect fit for the problem domain. As a third option, the CNN is selected to evaluate if this popular deep learning technique is applicable for human activity recognition.

5.3.1. MLP from scikit-learn

A simple *multilayer perceptron* (MLP) with the parameters listed as in Listing A.1 is chosen to be trained. The MLP hyperparameters are found with grid search. This algorithm uses the categorical cross-entropy loss function and stochastic gradient descent. An adaptive learning rate is added to the model, with the initial learning rate being 0.0001. The activation function is ReLU.

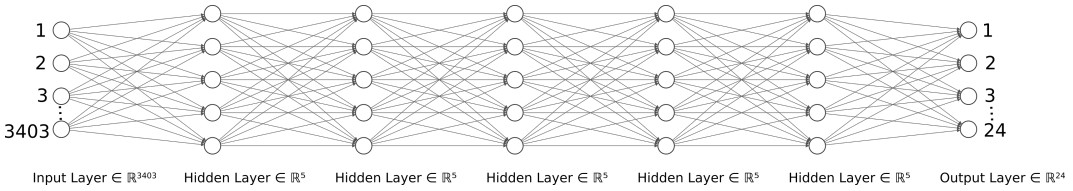


Figure 5.1.: The network architecture of the MLP neural network consists of five layers with five neurons in each layer. The input contains the features for a single data point and the output layer consists of one probability per class.

The hyperparameter *shuffle* is set to false since the samples should stay in the sequence they were captured in. Early stopping is applied to prevent overfitting and stop when the validation score is not improving. The neural network is built with five hidden layers, each of them having five neurons. 20% of the training data set are held back for validation. The batch size is set to a rather small number - five. As it can be seen in Figure 5.1, the MLP has five layers with five neurons in each layer. The choice of layers and their neurons is based on trials with high as well as small numbers of layers and neurons. The models were evaluated simply on their training and testing accuracy. The input contains the features for a single data point and the output layer consists of one probability per class.

5.3.2. Keras Models: CNN and LSTM

The CNN and LSTM network are implemented with Keras. The input for the neural networks is a tensor with the shape samples x time steps x features per step (see Figure 5.2). For one iteration, the depth of the tensor is reduced to the batch size. 40 lagged time steps are used, meaning that the input data includes data from 41 time steps in total. Each time step consists of 83 features. The output of the deep learning networks has the dimensions batch size x classes using the softmax activation function. Per class, the predicted probability (a value between 0 and 1) of the sample belonging to the class is given. The maximum value is evaluated to determine the most probable class. The ground truth of the neural networks is a one-hot encoded matrix with the classes as columns (filled with binary values), as it is required by the algorithm.

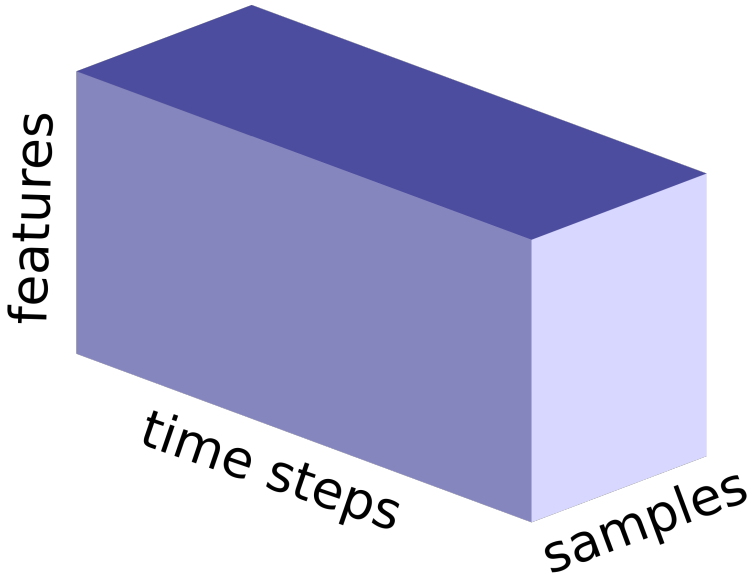


Figure 5.2.: The input for the CNN and LSTM network is a 3D tensor with the shape samples x time steps x features per step.

The amount of the hidden layers and the hidden neurons in each layer is first tested with a deep network with many layers and then with a more shallow approach with only a small amount of layers. The complexity increases the more hidden layers are present. Many hidden layers affect the training duration since more time is necessary to backpropagate through all those neurons. Both algorithms rely on the categorical cross-entropy loss function and stochastic gradient descent. As an optimizer, Nadam is selected with the suggested default parameters according to the Keras documentation. The *shuffle* flag of the Keras models is set to false as with the MLP. The learning rate is set to a small value because it shows a higher accuracy. Batch normalisation applied to the input speeds up the training, but when it is applied to the CNN and LSTM in this project, the model does indeed learn faster but the accuracy drops significantly. Therefore batch normalisation is not used.

LSTM

The hyperparameters for the LSTM network are chosen as shown in Listing A.2 in the appendix after tweaking them with various trial and error runs. The network consists of the following layer setup: LSTM layer with 8 neurons → fully connected layer (dense layer) with 8 neurons and ReLU as activation → softmax output layer. L2 regularisation is used against overfitting. 50 epochs are used for the training. Early stopping is applied with 5 epochs of patience, meaning that if the validation loss is not changing for 5 consecutive epochs, the training is stopped. 20% of the input data is set up for the validation of the model. The batch size is 4 samples. To combat overfitting, L2 regularisation is applied to the LSTM layer and fully connected layer. Furthermore, dropout is tried but the accuracy of the model drops and thus dropout is removed in the end version of the model.

CNN

The network architecture is shown in Figure Figure 5.4. The network consists of the following layer setup: 1D convolution (64 filters, kernel size 2, ReLU activation) → 1D max pooling with size 2 → flatten → fully connected layer (64 neurons, ReLU activation) → dropout → softmax output layer. 1D convolution is chosen since it is the default for time series classification. The filter of the convolution is one-dimensional.

Since the CNN proves to be the best performing algorithm, its hyperparameters are tuned to get the best possible model. Using a hyperparameter optimisation for Keras called Talos[12], the hyperparameters for the CNN are found (see in Listing A.2 in the appendix). The Talos code is available in Listing A.5. The batch size is set to 5, epochs and early stopping are the same as with the LSTM network. 25% of the training samples are held back for validation. L2 regularisation results in a slightly lower accuracy which is why it is omitted.

The layers of the CNN and the dimensions of the inputs and outputs are visualised in Figure 5.5. The 1D convolution layer has 64 filters (a kernel size

[12] *Talos - Hyperparameter Optimization for Keras Models* 2019.

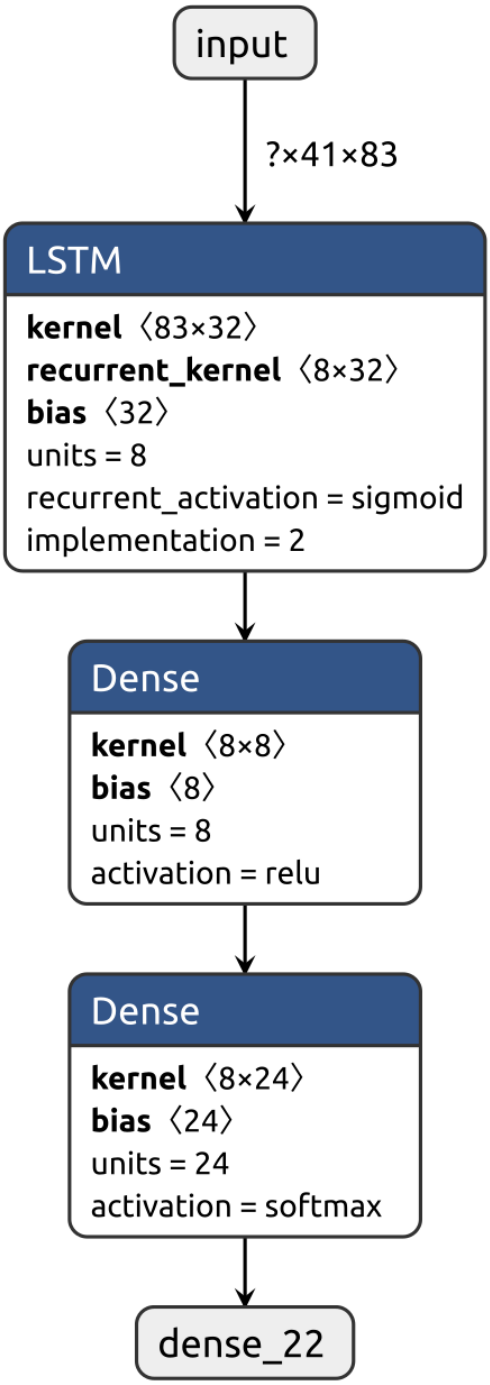


Figure 5.3.: The network architecture of the LSTM consists of: LSTM layer with 8 neurons → fully connected layer with 8 neurons ReLU as activation → softmax output layer.

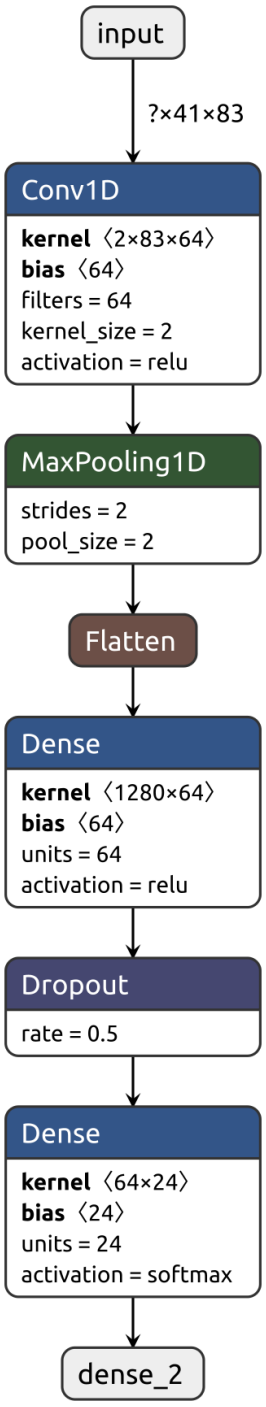


Figure 5.4.: The network architecture of the CNN consists of: 1D convolution (64 filters, kernel size 2, ReLU activation) → 1D max pooling with pooling window size 2 → flatten → fully connected layer (64 neurons, ReLU activation) → dropout → softmax output layer.

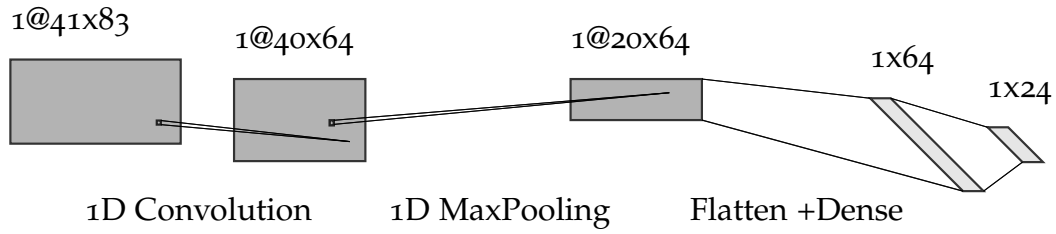


Figure 5.5.: In this graph, the layers of the CNN are visualised with the input and output sizes of the tensors. For simplicity, the batch size is displayed as 1 in the graph.

of 2 is applied to each filter). With 64 filters, the layer can detect 64 different features. With an input matrix height of 41 and a kernel size of 2 (length of the 1D convolution), the sliding window of the convolution processes 40 steps (41-2+1). The resulting output matrix is batch size x 40 x 64. The single convolution layer is already enough to learn the features. The 1D max pooling layer slides a window of height 2 across the output data from the convolution layer and replaces it with the maximum value of the values in the pool of the window. Thereby max pooling reduces the likelihood of overfitting of the learned features since the maximum value is taken by the sliding window approach. The output is a matrix with the dimensions batch size x 20 x 64, halving the input. To prepare the output as an input for the fully connected layer, the matrix is flattened. The layer has 64 neurons and uses the ReLU function for activation. To reduce overfitting even more, dropout with a rate of 0.5 is applied to the output layer. This means that 50% of the neurons of the layer are chosen at random to drop out. The input units of those neurons are set to 0 at each update during training which reduces the chances of the network to become overly sensitive to smaller variations of the data. Finally, the output layer calculates the probability distribution for 24 classes with a softmax activation function. Naturally, all probabilities add up to the sum of 1. The categorical cross-entropy loss is then calculated by comparing the probability scores and true labels of the samples.

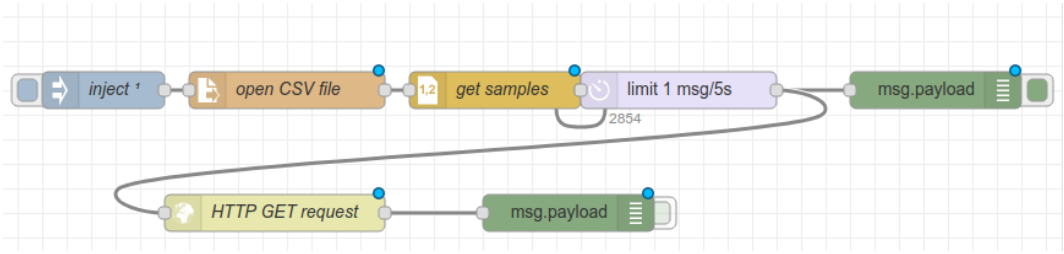


Figure 5.6.: A Node-RED flow to simulate sensor data coming in from the devices and being transmitted to a REST API is implemented. The flow reads in the input data from a CSV, selects one data point every 5 seconds and sends it to the API with a HTTP GET request. The API returns the predicted class.

5.4. On the Fly Prediction with Node-RED Environment

An IoT system using one of the classifiers as listed above is simulated using Node-RED and Flask. A REST API is built with Flask which loads the saved model from the training phase (the code is available in Listing A.4 in the appendix). To simulate the sensor input, a flow in Node-RED (see Figure 5.6) is set up to send JSON data to the API. For simplicity, the input data for the live prediction is read in from a CSV file by a Node-RED node and contains the testing data, already set up with 40 lagged features attached to one data point. One sample is transmitted via a HTTP GET request to the API every 5 seconds. As a classifier, the best CNN model of the ten training experiments is taken. The API loads the trained model, takes the received data point for prediction and returns the predicted class as a string.

6. Evaluation

In this chapter, the results of the machine learning algorithms used for the classification of different activities of daily living are compared. The chosen metrics were accuracy, recall (weighted), precision (weighted), F1 score (weighted), Matthews correlation coefficient (MCC) and categorical cross-entropy loss. Recall, precision and F1 score were weighted to deal with the class imbalance of the data set. The definitions of these metrics are given in Chapter 2.1. Importantly, the tests with the deep learning models from the Keras library did not converge at the same result each run - 10 tests had to be executed and averaged to get a stable comparison with the standard machine learning results which showed constant results. The experiments were conducted with the following system:

- *Operating System:* Ubuntu 18.04.3 LTS
- *CPU:* Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
- *RAM:* 8GB

In addition to the evaluation of the experiments, a short comparison of the used machine learning libraries Keras and sci-kit learn is part of this chapter.

6.1. Comparison of Machine Learning Algorithms

The metrics of the results of the machine learning algorithms are displayed in Figure 6.1 and in Table 6.1. The best classification result from the experiments in this thesis was obtained by the CNN model with an accuracy of 70.06%. The training accuracy and training accuracy are similar for all algorithms

except for the CNN. A higher training accuracy compared to the testing accuracy is often a sign of overfitting.

From analysing the metrics, it becomes apparent that the precision metric has a higher spike in the bar chart than the rest of the metrics. With the formula in mind, the precision measurement illustrates that the models did not classify a high amount of false positives in comparison to the true positives. Most of the true positives were simply never predicted. For the support vector machine and random forest algorithms, the recall was higher than precision. The standard algorithm that resulted in the overall best performance was logistic regression with an accuracy of 65.63%. This means that using the model obtained from logistic regression, 65.63% of the samples can be classified correctly. The worst result came from the gaussian naive bayes algorithm. Naive Bayes works with the assumption that all input features are conditionally independent. Since some features have a strong strength of association according to the Pearson correlation, this might be the reason that the algorithm performs poorly. Logistic regression still works well when features are correlated. Interestingly, Jiménez and Seco, 2018 reported an accuracy of 60.5% when using a multi-event naive bayes classifier. As in this thesis, the developers also incorporated all four sensor sources, but with different preprocessing choices. Ceron, López, and Eskofier, 2018 developed several different methods and noted the following training accuracy when applying 10-fold-cross validation for the same algorithms used in this project:

- Support vector machine: 89.4%
- Random forest: 90.3%
- Bagging: 89.8%

The best result when training was gained by the AdaBoostM1 algorithm which was why this algorithm was explored further. In the end, the testing accuracy for classifying activities of the test data set was only 60.1%, 62.77% without the zero class (class used for samples that are not associated to any out of the 24 defined classes). The developers decided to aggregate the activities related to eating meals of the day into one single event. For example, "dinner", "lunch" and "breakfast" were summarised as the activity "eating". This step was also performed to the activities related to preparing those meals. Merging those activities increased the classification accuracy by

Algorithm	Train Accuracy	Test Accuracy	Recall	Precision	F1 Score	MCC	Loss
Logistic Regression	66%	65.63%	67.02%	71.44%	63.11%	64.48%	3.98
Gaussian Naive Bayes	34%	33.54%	43.33%	55.75%	39.64%	32.13%	22.95
Decision Tree	43%	42.93%	42.93%	48.39%	40.03%	40.27%	19.71
Support Vector Machine	53%	52.73%	61.39%	47.93%	50.23%	51.98%	1.15
K-Nearest Neighbors	35%	34.92%	42.99%	47.16%	31.54%	28.46%	20.86
Random Forest	46%	46.10%	61.17%	50.00%	50.90%	44.51%	2.28
Bagging	48%	47.69%	48.70%	55.31%	43.36%	45.81%	10.77
Extra Tree	56%	55.38%	56.50%	62.15%	51.30%	55.18%	9.22
Gradient Boosting	56%	56.18%	57.36%	64.41%	51.62%	55.31%	2.45
MLP	50%	50.31%	51.37%	58.71%	48.66%	47.67%	2.29
LSTM	57%	57.41%	60.68%	63.26%	56.53%	55.15%	1.65
CNN	83%	70.06%	71.87%	73.36 %	69.80%	68.13%	1.19

Table 6.1.: Metrics of different machine learning algorithms

13%. For the data preparation steps in this thesis project, the activities were not merged and the segment of the day was added to provide a guide for the model how to differentiate between those activities. Furthermore, the zero class was omitted from the data points. When comparing the results from this thesis to Ceron, López, and Eskofier, 2018, it seems that the models by the other researchers suffered from overfitting. The experiments with the standard algorithms described in this chapter do not show hints of overfitting.

A remark on the preprocessing choices is that changing the ways of preprocessing (sampling the sensor data in time slots in particular) can have a significant

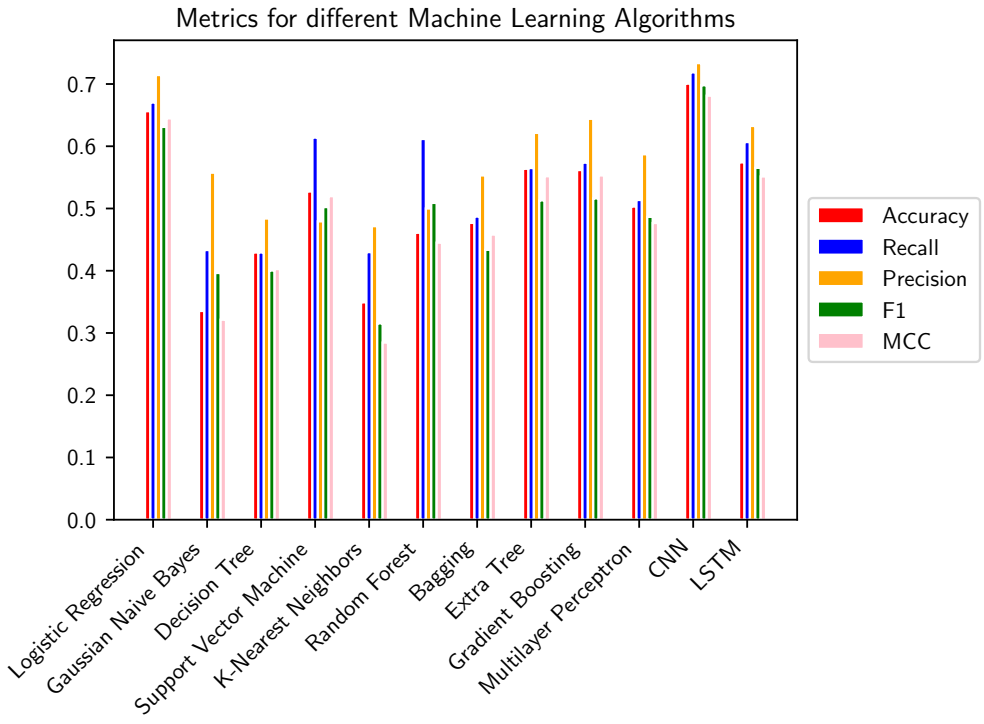


Figure 6.1.: Bar chart of metrics of different machine learning algorithms performed on the data set.

effect on training time and outcome. Furthermore, the problem with uneven class distribution in the different input data sets has to be addressed. Especially if classes are not present in the same amount in training and validation set, the model has a hard time learning.

Figure 6.2 shows the training time in seconds of the algorithms. The training duration varies greatly. It is apparent that the training time for the LSTM was much longer than the training time of the CNN. Notably, the CNN converged faster but showed the better accuracy. From looking at the standard algorithms, it seems that a shorter training time does not immediately mean a better performance. The training times for gaussian naive bayes, decision tree, k-nearest neighbors and extra tree were extremely short, nevertheless the best model performance from the standard algorithms

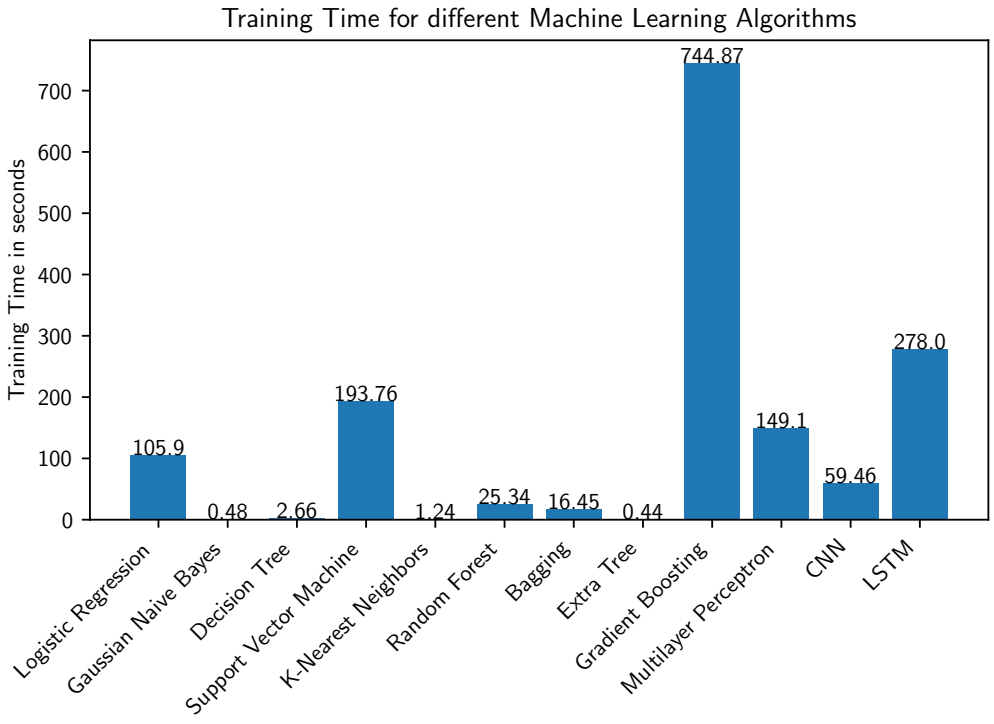


Figure 6.2.: Bar chart of average training times in seconds for different machine learning algorithms performed on the data set.

was from logistic regression. The MLP took about double the time to train than the CNN. In general, the solution must not rely on faster computation (which can be achieved by hardware improvements), it also should look for a better solution to solve the problem.

The categorical cross entropy loss is compared in Figure 6.3. From studying the chart, the loss was the lowest for the support vector machine, followed by the CNN and LSTM. The highest loss was reported for the gaussian naive bayes algorithm which also showed weak performance. The loss of the gaussian naive bayes was actually not that high compared to the other algorithms even though its accuracy was the lowest.

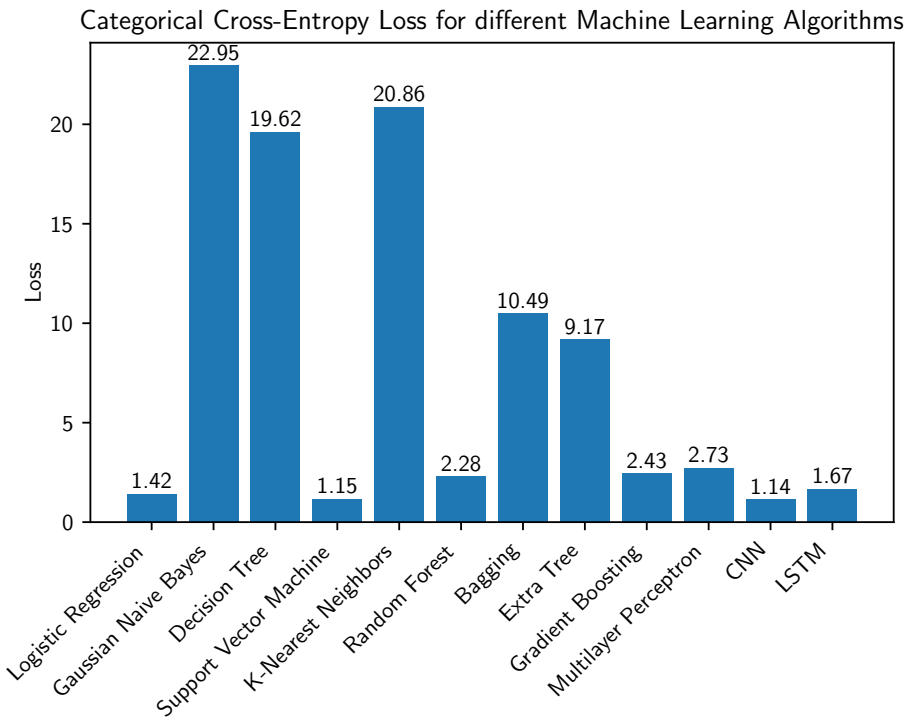


Figure 6.3.: Bar chart of cross entropy loss of different machine learning algorithms performed on the data set.

6.1.1. Logistic Regression from sci-kit learn

Logistic regression was the best performing standard algorithm according to the experiments. The confusion matrix for the logistic regression algorithm is available in Figure 6.4. Most of the activities had a high chance of being predicted correctly. The activities "wash dishes" and "Visit in the SmartLab" were not predicted for any of the testing samples even though there were data points present from those classes. Other standard algorithms showed difficulties of predicting those two classes. The confusion matrix illustrates that the biggest problem for the model was to spot the activity "relax on the sofa". This activity was often misclassified as "lunch", "dinner" and "watch TV". This is probably because the correlation of the kettle sensor and the the sofa pressure sensor that was seen in the Pearson correlation already. Those sensors were mainly involved in the four activities mentioned. The confusion matrices for the other models show as well that the models had difficulties classifying the activity "relax on the sofa" correctly. Interestingly, "play a video game" was nearly always wrongly classified as "watch TV". There was only one occurrence of this activity in the test and in the training set which might be the reason why the model cannot learn the patterns of this activity. Furthermore, the activities are fairly similar. "Use the toilet" was recognised as the activity "dinner" for most of the samples. The activity "wash dishes" was mostly confused with "put waste in the bin".

The logistic regression classification algorithm calculates the probability of each activity and then takes the label which results in the maximum of those probabilities. In Table 6.2 shows the probabilities of one sample. The highest value result was returned for activity "wash hands", but it was actually activity "brush teeth". For real life samples where there is no clear majority vote for a single class and the model is unsure of the right label, the IoT system could give the top candidate activities.

6.1.2. MLP from sci-kit learn

The multilayer perceptron performed on the data set with an accuracy of 50.31%. Nevertheless, its predictive ability was beaten by the CNN and LSTM models and several of the standard machine learning algorithms.

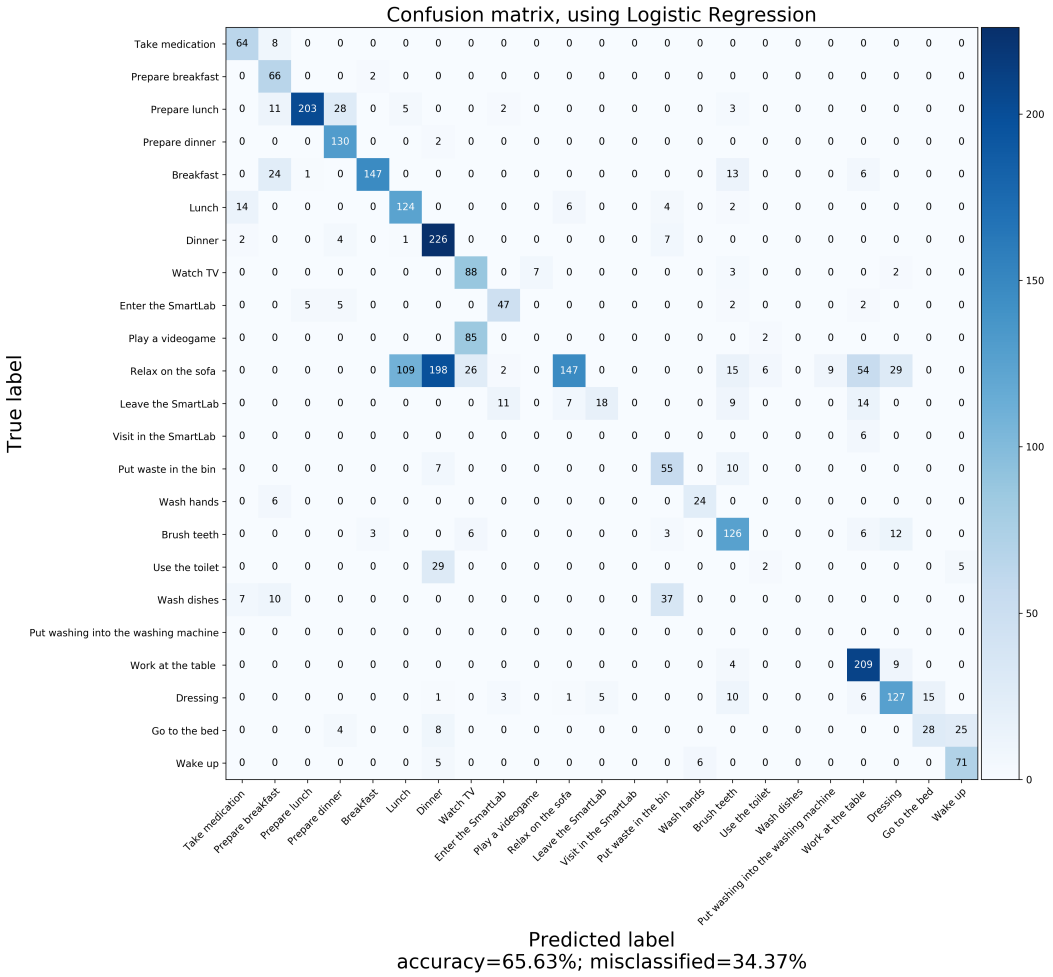


Figure 6.4.: Confusion matrix for the classification results of Logistic Regression performed on the data set using scikit-learn.

Activity	**Probability**
Take medication	0.00
Prepare breakfast	0.00
Prepare lunch	0.00
Prepare dinner	0.00
Breakfast	0.00
Lunch	0.00
Dinner	0.00
Eat a snack	0.00
Watch TV	0.00
Enter the SmartLab	0.00
Play a video game	0.00
Relax on the sofa	0.00
Leave the SmartLab	0.00
Visit in the SmartLab	0.00
Put waste in the bin	0.00
Wash hands	0.03
Brush teeth	0.93
Use the toilet	0.03
Wash dishes	0.00
Put washing into the washing machine	0.00
Work at the table	0.00
Dressing	0.00
Go to bed	0.00
Wake Up	0.00

Table 6.2.: Class probabilities for one sample using logistic regression

The confusion matrix is displayed in Figure 6.5. It is visible that the model misclassified many activities, especially the activity "relax on the sofa" which posed a problem to the other classifiers as well. As in the case of logistic regression, the activities "wash dishes" and "Visit in the SmartLab" were not predicted for any of the testing samples. Looking at the metrics, the model's precision value was much higher than the other four metrics, highlighting that the percentage of true positives was high in comparison to false positives.

6.1.3. Detailed Comparison of the Deep Learning Models LSTM Network and CNN from Keras

Surprisingly, the results from the LSTM model were not outperforming the standard algorithms nor the MLP and CNN as it was concluded from the literature review. The average accuracy was 57.41% (+/-13.4) over 10 runs. The averaged loss is 1.65 (+/-0.51). The cause for the weak performance might be that the training data set is fairly small for a deep learning problem (5.344 samples for training and 2.920 samples for the testing), leading to overfitting. Several ways to combat the overfitting of the model were tried, but it seems that the model is simply not a good fit since other models performed significantly better. Figure 6.6 displays the confusion matrix of the LSTM model. The model plotted in the confusion matrix could not recognise the class "put washing into the washing machine" even though it was present in the testing data. As the other confusion matrices have shown, the activity "relax on the sofa" is misclassified by the LSTM as well. Instead the predictions were "breakfast","lunch" and "work at the table".

The performance of the CNN was similar to the best results from the standard algorithms with an average of 70.06% (+/-2.3) of accuracy in 10 runs. The averaged loss is 1.19 (+/-0.11) which is the second lowest after the support vector machine algorithm. Notably, the training accuracy was much higher than the testing accuracy with 83%. This can be a sign of overfitting of the model. Figure 6.7 displays the confusion matrix of the CNN model. The model plotted in this heatmap was unable to predict the activity "eat a snack" at all, even though it was present in the testing set. The plot shows that the model had the same problem as the other algorithms - it could

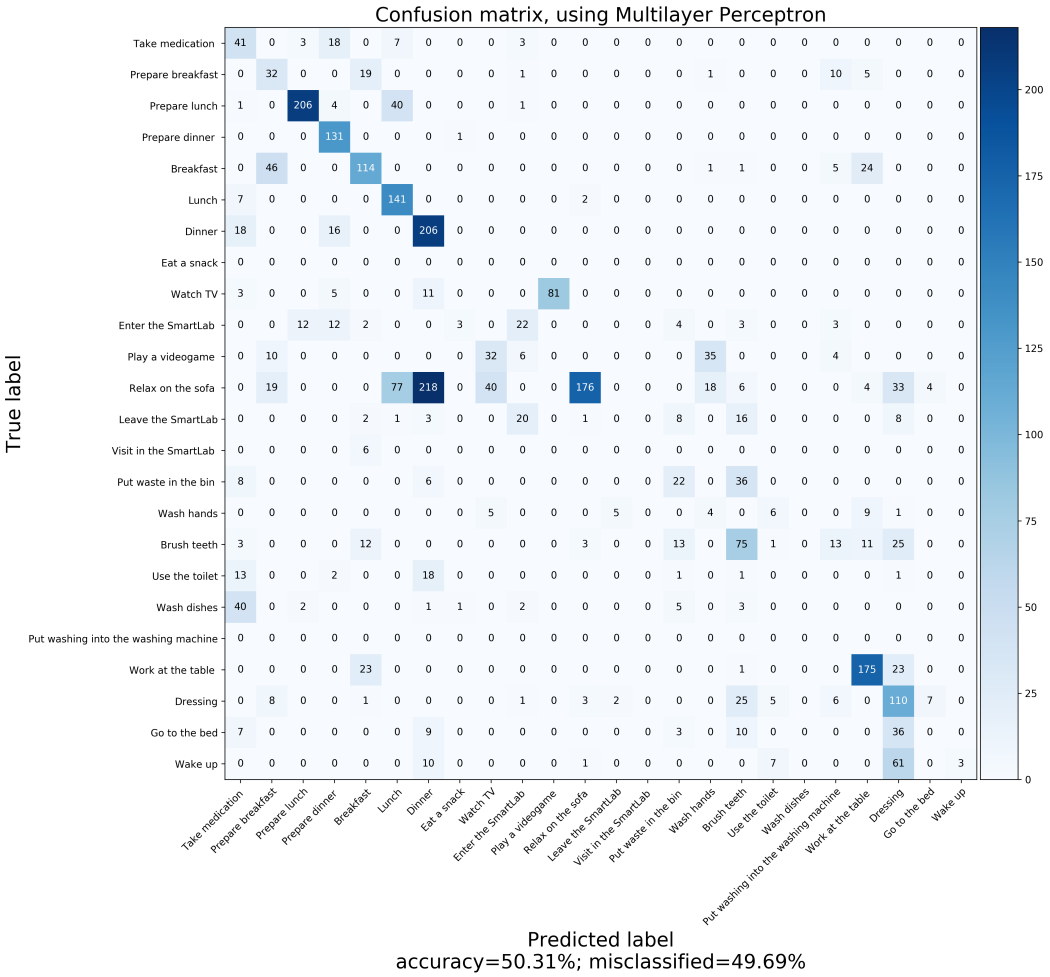


Figure 6.5.: Confusion matrix for the classification results of the multilayer perceptron performed on the data set using sci-kit learn.

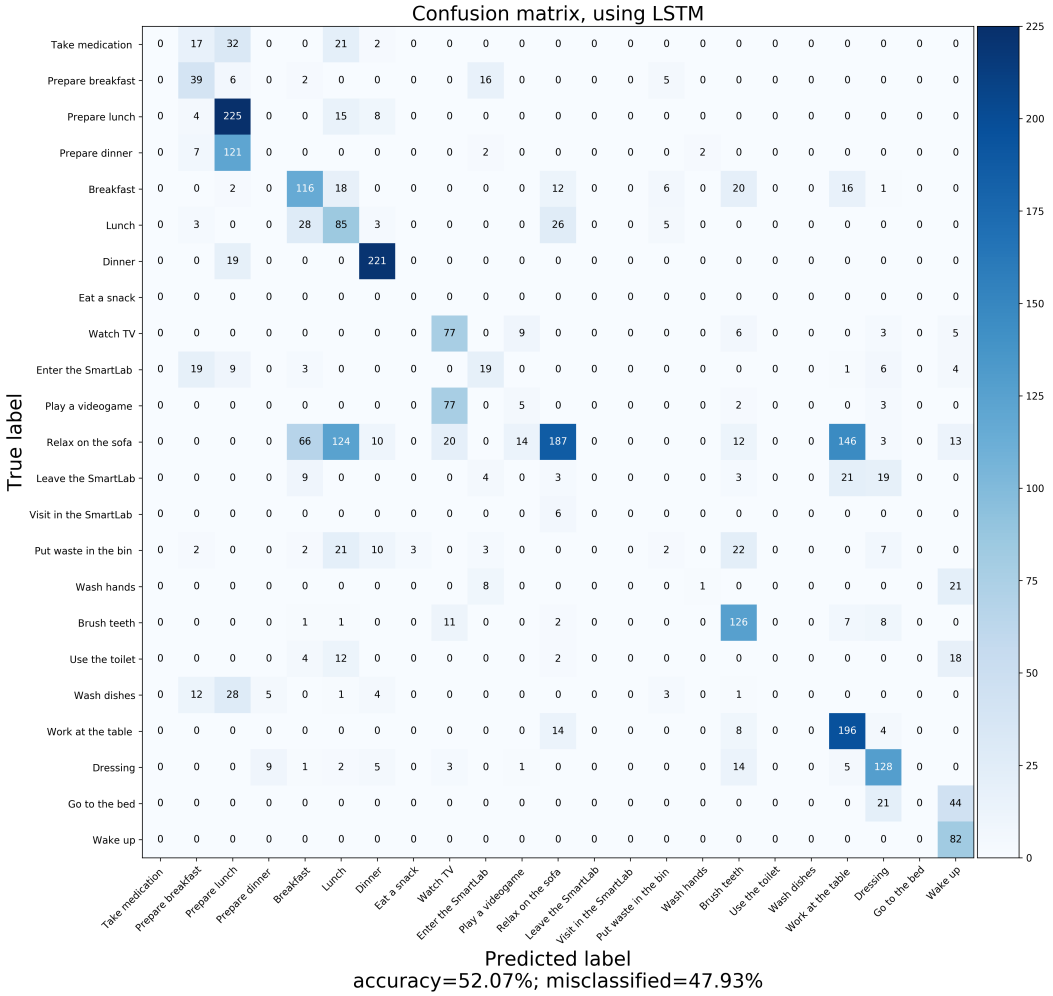


Figure 6.6.: Confusion matrix for the classification results of the deep learning model LSTM performed on the data set using Keras.

hardly classify the activity "relax on the sofa" and predicted "dressing", "lunch"and "dinner" instead. Many methodologies to combat overfitting were tested. Batch normalisation applied to the input layer ends up with an accuracy of 25%, much lower than without. L2 regularisation resulted in 61%. L2 regularisation was thus not used, but applied to the LSTM since there it did result in a slightly better performance. From the success of the CNN model over the other trained classifiers it is concluded that the input data is more similar to an image with one channel, showing spatial features in a single dimension.

Figure 6.8 illustrates how training and validation loss behave over the epochs of the CNN and LSTM model. The training loss for the first couple of epochs shows up slightly higher than the validation loss. This means that the model is better at predicting unseen data than on the training data which could be caused by a validation set with very different data. The training accuracy shows a well-formed curve close to 90%. For the LSTM, the accuracy improved linearly over the epochs, but the validation loss did not seem to become significantly smaller, but instead increased slightly. The training loss nevertheless decreased steadily.

The deep learning algorithms from Keras were run 10 times to get an averaged result since the outcome can be different each run. The runs are compared in Figure 6.9. These graphs were created using the Tensorboard visualisation[1]. There are two clusters visible - one is the CNN and one is from the LSTM. For the CNN, the different functions for each run were similar and nearly logarithmic. The LSTM runs showed a higher variance in their performance. The LSTM took more time to train than the CNN and the function has a more linear growth. From this analysis, it is clear that the CNN converged faster than the LSTM since early stopping is used. Early stopping which forces the deep learning algorithm to stop learning before the model overfits was applied to the CNN and LSTM. For the CNN, the training was stopped at epoch 21 on average. In case of the LSTM, the early stopping specifications were several times not met and the LSTM ran for the full 50 epochs.

The random probability of predicting the right class would be 4.16% with balanced occurrences of classes in the data set. Taking this as a baseline,

[1] *TensorBoard - Visualisation for Tensorflow and Keras* 2019.

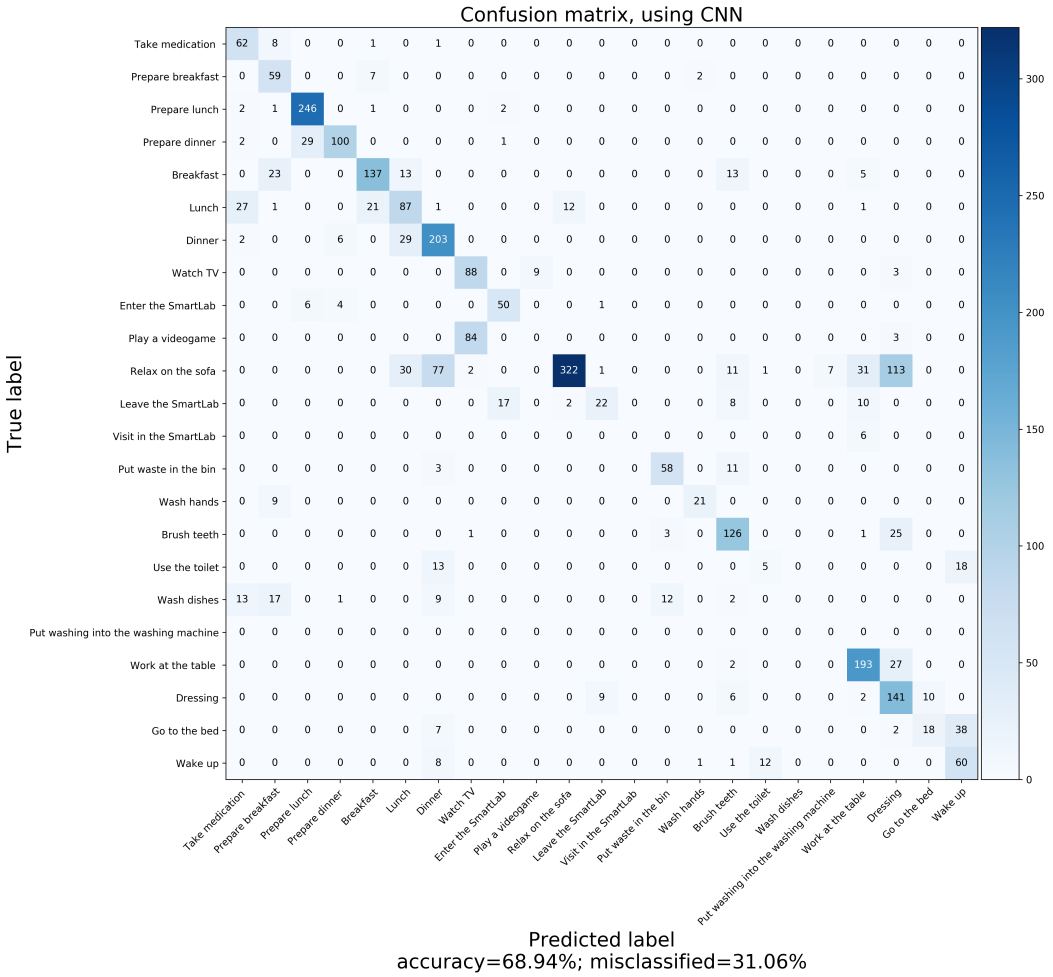
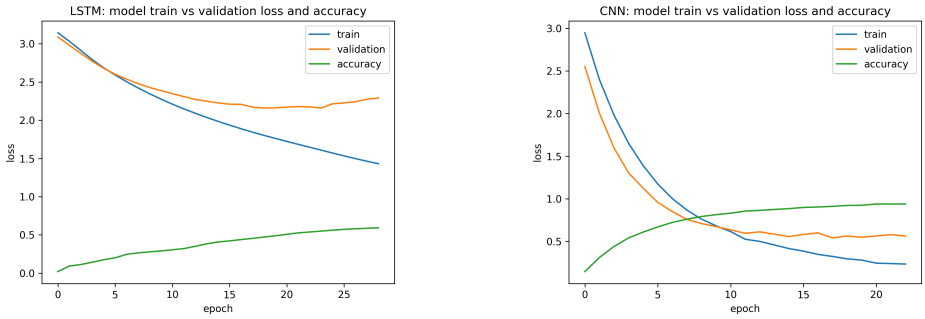


Figure 6.7.: Confusion matrix for the classification results of the deep learning model CNN performed on the data set using Keras.



(a) Training loss, validation loss and training accuracy curves of LSTM model

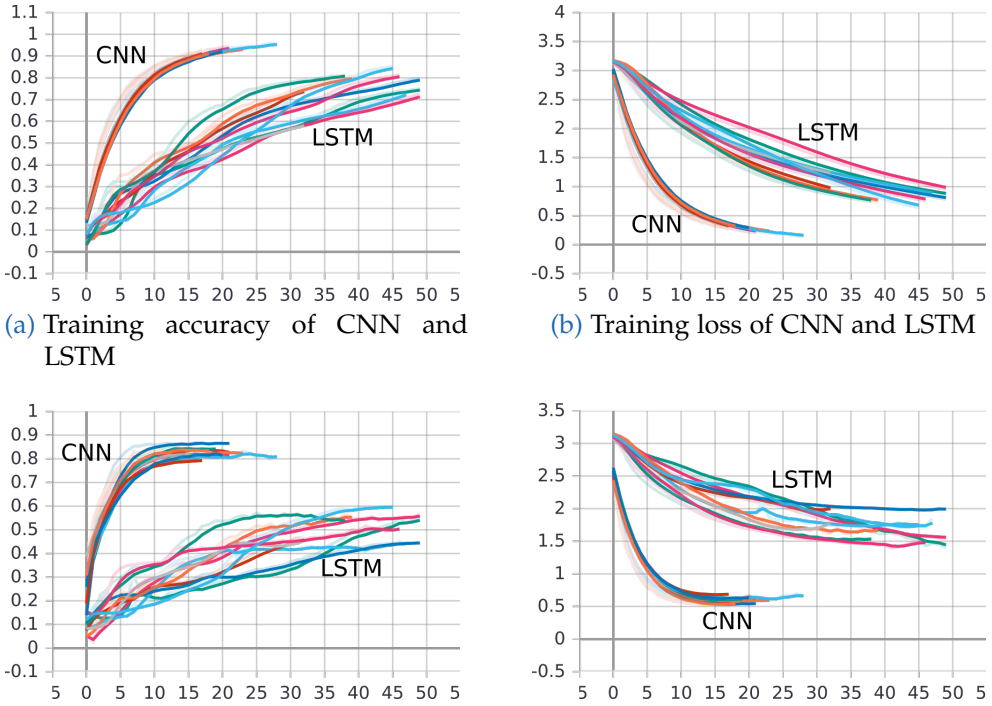
(b) Training loss, validation loss and training accuracy curves of CNN model

Figure 6.8.: Different metrics of the deep learning models LSTM and CNN over the epochs.

the overall performances of the algorithms in this thesis provide a much more accurate prediction. For this particular data set, it seems a simple architecture is already good enough, no need to add more complexity to the model. Arguably, logistic regression proves its predictive ability, trains fast and and is easy to implement.

6.2. Evaluation of Tools

Two different machine learning libraries were used in this project: Keras (built on top of TensorFlow) and sci-kit learn. TensorFlow is already a highly optimised library for deep learning. Nevertheless, Keras goes a step further and brings a new level of usability with it. It is much easier for developers to understand the code and there is less need to fully grasp the mathematics behind the called functions due to how easy the built-in functions are. There is an extensive documentation and a growing user base. The sci-kit learn library exists longer than Keras. The library has it's focus on standard machine learning algorithms and meets its users with a well-written documentation and great community support. Additionally, sci-kit learn possesses preprocessing tools. All in all, both libraries are excellent for machine learning, but they do serve different purposes with



(a) Training accuracy of CNN and LSTM

(b) Training loss of CNN and LSTM

(c) Validation accuracy of CNN and LSTM

(d) Validation loss of CNN and LSTM

Figure 6.9.: Different metrics of the 10 runs of the experiments using LSTM and CNN over the epochs.

their distinguished set of available algorithms. An exception would be the multilayer perceptron. This model can be built with both libraries.

7. Conclusions

This thesis provides an evaluation of different machine learning techniques for classifying activities of daily living in a smart home. This chapter concludes with a final statement about the the main results from this project and recommendations for further work on this topic.

To sum up the observations of this thesis, the best performing standard algorithm in terms of accuracy was logistic regression with 65.63%. Compared to Ceron, López, and Eskofier, 2018 who worked on a similar solution ending up in 60.1% accuracy, the results of the static machine learning methods of this thesis show a performance improvement. Additionally, it must be noted that Ceron, López, and Eskofier, 2018 merged the meal activities into a single activity which, of course, increased the accuracy because there are less classes to predict. The multilayer perceptron reported an accuracy of 50.31%. The LSTM model showed an accuracy of 57.41% (+/-13.4), the CNN in 70.06% (+/-2.3) on average - resulting in only slightly higher scores than the best standard algorithm logistic regression. Surprisingly, the deep learning classifiers were not able to significantly outperform the standard algorithms used in this project.

These results can either be a sign that there was only a weak pattern to be learned from the input data or that the algorithms were not a good choice. From the literature review, it became apparent that human activities of daily living can barely be judged from examining static input samples including aggregated data from a few seconds. The features identifying a certain activity might not all be present in that single sample, but might have been included in a previous sample or are part of future samples. Static machine learning algorithms like logistic regression have difficulties with detecting patterns in sequences of sensor data. With this in mind, it is surprising that the LSTM (which has a recurrent factor included) performed so badly.

Improvements of the hyperparameters for the better performing CNN model were intensively investigated. Unfortunately, it became clear that the data set led to overfitting, no matter which hyperparameters were tried for the deep learning models. The reasons were most likely the insufficient quantity of data and the class imbalance. Additional factors might have been non-representative training data or irrelevant features.

The key drawback of finding a fitting machine learning algorithm to solve a problem such as the one presented in this thesis is that there is no straightforward, trivial solution. Machine learning is a powerful tool, but the right technique and method has to be found for each particular problem. Experiments have to be conducted to empirically evaluate the results and fine tune the preprocessing and hyperparameters. Every researcher in the field of machine learning is met with the challenge of interpreting the performance metrics such as training and testing accuracy correctly and adapting the chosen machine learning method to build a better model. Investigating a variety of different options takes a substantial amount of time, thus already existing experiments with the same data set are valuable to build on.

Next to introducing deep learning to human activity recognition, this project also evaluated existing machine learning libraries. Until now, the library scikit-learn has reached a mature level of documentation and possesses a substantial community. TensorFlow and especially Keras have just evolved recently, setting their focus on deep learning. Scikit-learn proves especially useful due to its simplicity. With Keras, the implementation of the model has gotten majorly easier than with TensorFlow, but is still more complex than with scikit-learn.

To summarise, it is shown that the applicability of deep learning for human activity recognition is given and it is a suitable classifier. In general, the volume of reviewed literature showed that not much progress has been made with using deep learning for the problem domain, but will most definitely be made in the future. Some directions for further research are proposed in the following section.

7.1. Future Work

Future work should concentrate on finding other data sets for HAR and thus validate the results of this thesis. Finding a good solution that applies to many different HAR data sets will be a challenge for a long time. Furthermore, here is a vast amount of possible activities in the context of daily living which are not part of the UCAMI Cup 2018 data set.

The preprocessing methods used in this project leave room for improvement. Not much time was invested in the feature selection for the presented solution. Future work might take up on this issue and perform a more extensive analysis of the feature extraction and assess which combinations of feature inputs improves the model's prediction ability.

Extensions on the presented models could by done by experimenting more with other deep learning architectures such as a VGG or GoogLeNet or trying a hybrid model using CNN layers paired with LSTM layers. For the LSTM in particular, the hyperparameters could be explored further.

Concerning the implementation using Node-RED, additional real-life plots of the model could be included in the Node-RED dashboard. Actual devices could be connected instead of simulating the data inputs. Of course, a whole system with an actual user interface and the sensor infrastructure should be implemented for a smart home using the proposed machine learning algorithms.

Concluding, there is much left for other researchers to investigate and build on the approaches used in this thesis.

Appendix

Appendix A.

Code Snippets

Listing A.1: The machine learning algorithms from the scikit-learn library in Python are executed using mostly default parameters.

```
#sci-kit learn machine learning algorithms
self.regressor = LogisticRegression(random_state=0, solver='lbfgs',
    multi_class='multinomial', max_iter=5000)
self.gnb = GaussianNB()
self.dt = tree.DecisionTreeClassifier()
self.svc = SVC(C=1.0, decision_function_shape='ovr', degree=3, gamma='
    auto_deprecated', kernel='rbf', probability=True)
self.knn = KNeighborsClassifier(n_neighbors=5)
self.rf = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state
    =0)
self.b = BaggingClassifier()
self.et = ExtraTreesClassifier()
self.gb = GradientBoostingClassifier()
self.ada = AdaBoostClassifier()
self.mlp = MLPClassifier(activation='relu', early_stopping=True,
    hidden_layer_sizes=(5,5), max_iter=500,
                        shuffle=False, solver='sgd', validation_fraction=0.2,
                        batch_size=5, learning_rate='adaptive',
                            learning_rate_init=0.0001)
```

Listing A.2: The hyperparameters for the long short-term memory recurrent neural network using the Keras library are executed with the parameters shown in this code sample.

```
n_outputs = 24  # number of classes

model = Sequential()
model.add(LSTM(8, input_shape=(train_X.shape[1], train_X.shape[2]),
    kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dense(8, activation='relu', kernel_regularizer=regularizers.l2
    (0.0001)))
model.add(Dense(n_outputs, activation='softmax'))
optimizer = optimizers.Nadam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e
    -07, schedule_decay=0.004)
```

```
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['
    acc'])
# early stopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=4, verbose=2,
    shuffle=False, validation_split=0.2, callbacks=[es])
```

Listing A.3: The hyperparameters for the convolutional neural network using the Keras library are executed with the parameters shown in this code sample.

```
n_outputs = 24  # number of classes
epochs = 50

model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(
    train_X.shape[1], train_X.shape[2])))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_outputs, activation='softmax'))
optimizer = optimizers.Nadam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e
    -07, schedule_decay=0.004)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['
    acc'])
# early stopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
# fit network
history = model.fit(train_X, train_y, epochs=epochs, batch_size=5, verbose=2,
    shuffle=False, validation_split=0.25, callbacks=[es])
```

Listing A.4: A Node-RED flow to simulate sensor data coming in from the devices and being transmitted to a REST API is implemented. The flow reads in the input data from a CSV, selects one data point every 5 seconds and sends it to the API with a HTTP GET request. The API returns the predicted class.

```
import flask
from tensorflow import keras
import pandas as pd
from flask import request, jsonify
from pandas.io.json import json_normalize

app = flask.Flask(__name__)
app.config["DEBUG"] = True


@app.route('/api/prediction', methods=['POST'])
def precict():


    # Validate the request body contains JSON
    if request.is_json:
```

```
# Parse the JSON into a Python dictionary
req = request.get_json()
sample_df = json_normalize(req)

timesteps = 40
#sample_df = sample_df.drop(["TIMESTAMP"], axis=1)
sample_df = sample_df.astype(float)

x_test, y_test = sample_df.iloc[:, :-1], sample_df.iloc[:, -1]

n_features = 83
x_test_reshaped = x_test.values.reshape(x_test.shape[0], timesteps +
    1, n_features)


optimizer = keras.optimizers.Nadam(lr=0.0001, beta_1=0.9, beta_2
    =0.999, epsilon=1e-07, schedule_decay=0.004)

model = keras.models.load_model('models/CNN-1.h5', compile=False) #
    todo: get right model
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
    metrics=['acc'])


y_pred = model.predict(x_test_reshaped)
y_class = y_pred.argmax(axis=-1)
y_class = y_class + 1

y_pred_pd = pd.DataFrame(y_class, columns=["class"])
y_test_pd = pd.DataFrame(y_test.tolist(), columns=["class"])

activity_map = {0: "no␣activity", 1: "Take␣medication", 2: "Prepare␣
    breakfast", 3: "Prepare␣lunch", 4: "Prepare␣dinner",
                5: "Breakfast", 6: "Lunch", 7: "Dinner", 8: "Eat␣a␣
                    snack", 9: "Watch␣TV",  10: "Enter␣the␣SmartLab",
                11: "Play␣a␣videogame", 12: "Relax␣on␣the␣sofa", 13: "
                    Leave␣the␣SmartLab", 14: "Visit␣in␣the␣SmartLab",
                15: "Put␣waste␣in␣the␣bin", 16: "Wash␣hands", 17: "
                    Brush␣teeth", 18: "Use␣the␣toilet", 19: "Wash␣
                    dishes",
                20: "Put␣washin␣into␣the␣washing␣machine", 21: "Work␣
                    at␣the␣table", 22: "Dressing", 23: "Go␣to␣the␣bed"
                    ,
                24: "Wake␣up"}
predicted_class = y_pred_pd["class"].map(activity_map)
y_test_pd = y_test_pd.astype(float)
actual_class = y_test_pd["class"].map(activity_map)

prediction_result = "The␣new␣data␣point␣is␣predicted␣to␣be␣the␣
    activity␣{}␣({}).␣The␣ground␣truth␣activity␣is␣{}␣({}).␣".format(
    predicted_class[0], y_class[0], actual_class[0], int(y_test[0]))
if(y_class[0] == int(y_test[0])):
    prediction_result += "The␣system␣predicted␣correctly!␣"
else:
    prediction_result += "The␣system␣predicted␣wrong!␣"
```

```
        print(prediction_result)

        # Return a string along with an HTTP status code
        return prediction_result, 200

    else:

        # The request body wasn't JSON so return a 400 HTTP status code
        return "Request was not JSON", 400

app.run()
```

Listing A.5: The hyperparameters for the CNN model were optimised using Talos.

```
def evaluate_CNN_model_talos(self, train_X, train_y, x_val, y_val, params):

    n_output = 24  # number of classes

    model = Sequential()
    model.add(Conv1D(params['first_neuron'], kernel_size=2, activation=params[
        'activation'], input_shape=(train_X.shape[1], train_X.shape[2])))
    model.add(MaxPooling1D(params['pool_size']))
    model.add(Flatten())
    model.add(Dense(params['second_neuron'], activation='relu'))
    model.add(Dropout(params['dropout']))
    model.add(Dense(units=n_output, activation='softmax'))
    model.compile(loss=params['losses'], optimizer='adam', metrics=['acc'])
    # early stopping
    es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)


    print(model.summary())


    # fit network
    out = model.fit(train_X, train_y, epochs=50, batch_size=5, verbose=2,
        shuffle=False, validation_data=[x_val, y_val], callbacks=[es])
    return out, model
```

Appendix B.

Library Versions of Used Tools

The chosen programming language is Python[1] including the powerful tools Pandas[2] and NumPy[3] (preprocessing) and scikit-learn[4] (machine learning algorithms) as well as Matplotlib[5] and Plotly[6] (plots for visualisation). In terms of deep learning, the neural network library Keras[7] are used. Keras is based on the machine learning platform TensorFlow[8] . The hyperparameter optimisation for the Keras models are implemented using Talos[9]. Node-RED[10] and Flask[11] are used to simulate an IoT system to predict in real time. The application of each tool and its version is listed in Figure B.1. The source code of this thesis is available under the MIT license on Github: `github.com/kolbl/HAR_master_thesis`.

[1] *Python Programming Language* 2019.
[2] *Pandas - Python Data Analysis Library* 2019.
[3] *NumPy - Python Package for Scientific Computing* 2019.
[4] *scikit-learn - Machine Learning in Python* 2019.
[5] *Matplotlib - Plotting in Python* 2019.
[6] *Plotly - Python Graphing Library* 2019.
[7] *Keras - Library for neural networks* 2019.
[8] *TensorFlow - Open Source Machine Learning Platform* 2019.
[9] *Talos - Hyperparameter Optimization for Keras Models* 2019.
[10] *Node-RED: Low-code programming for event-driven applications* 2019.
[11] *Flask - Python web framework* 2020.

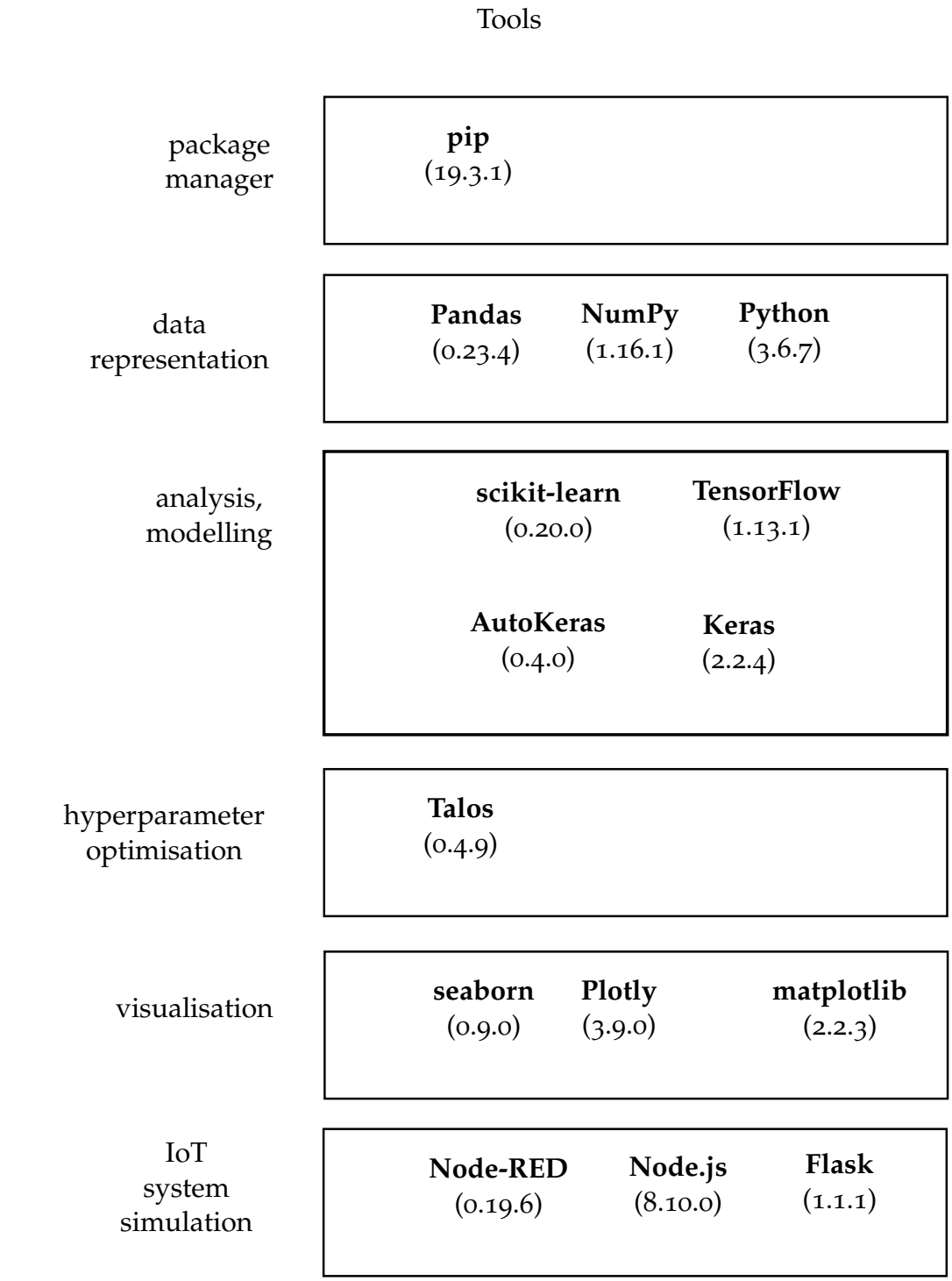


Figure B.1.: Implementation tools used for the project of this thesis.

Bibliography

Ann, Ong Chin and Bee Theng Lau (Nov. 2014). "Human activity recognition: A review." In: *2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014)*. Penang, Malaysia: IEEE, pp. 389–393. DOI: 10.1109/ICCSCE.2014.7072750 (cit. on p. 1).

Balas, Valentina et al. (2019). *Internet of Things and Big Data Analytics for Smart Generation*. Vol. 154. Intelligent Systems Reference Library. Springer International Publishing. ISBN: 9783030042035. DOI: 10.1007/978-3-030-04203-5.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult." In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. DOI: 10.1109/72.279181 (cit. on pp. 30, 31).

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN: 9780387310732 (cit. on p. 22).

Ceron, Jesus, Diego López, and Bjoern Eskofier (Dec. 2018). "Human Activity Recognition Using Binary Sensors, BLE Beacons, an Intelligent Floor and Acceleration Data: A Machine Learning Approach." In: *The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018)*. Vol. 2. 19 1265. Punta Cana, Dominican Republic: MDPI. DOI: 10.3390/proceedings2191265 (cit. on pp. 41, 43, 44, 80, 81, 97).

Chollet, François (2018). *Deep Learning with Python*. Manning. ISBN: 9781617294433 (cit. on pp. 2, 14, 19).

Deloche, François (n.d.[a]). *LSTM Cell*. Wimikedia. URL: commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg (cit. on p. 33).

Deloche, François (n.d.[b]). *RNN Graph*. Wimikedia. URL: commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg (cit. on p. 31).

Dozat, Timothy (2016). "Incorporating Nesterov Momentum into Adam." In: URL: cs229.stanford.edu/proj2015/054_report.pdf (cit. on p. 28).

Espinilla, Macarena, Javier Medina, and Chris Nugent (Dec. 2018). "UCAmI Cup. Analyzing the UJA Human Activity Recognition Dataset of Activities of Daily Living." In: *The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018)*. Vol. 2. 19 1267. Punta Cana, Dominican Republic: MDPI. DOI: 10.3390/proceedings2191267 (cit. on pp. 2, 3, 54, 55).

Flask - Python web framework (2020). URL: fullstackpython.com (visited on 01/04/2020) (cit. on pp. 70, 107).

Géron, Aurélien (2017). *Hands-On Machine Learning with Scikit-Learnand TensorFlow*. O'Reilly. ISBN: 9781491962299. URL: oreilly.com/library/view/hands-on-machine-learning/9781491962282/ (cit. on pp. 15, 18, 20, 22, 37).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press. URL: www.deeplearningbook.org (cit. on p. 33).

Google Scholar (2019). URL: scholar.google.co.uk/ (visited on 04/09/2019) (cit. on p. 39).

Hammerla, Nils Y. and Shane Halloran Thomas Plötz (2016). "Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables." In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, pp. 1533–1540. URL: ijcai.org/Proceedings/16/Papers/220.pdf (cit. on p. 45).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory." In: *Neural Computation* 9.8, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. on p. 31).

Jiménez, Antonio R. and Fernando Seco (Dec. 2018). "Multi-Event Naive Bayes Classifier for Activity Recognition in the UCAmI Cup." In: *The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018)*. Vol. 2. 19 1264. Punta Cana, Dominican Republic: MDPI. DOI: 10.3390/proceedings2191264 (cit. on pp. 41, 43, 80).

Karvonen, Niklas and Denis Kleyko (Dec. 2018). "A Domain Knowledge-Based Solution for Human Activity Recognition: The UJA Dataset Analysis." In: *The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018)*. Vol. 2. 19 1261. Punta Cana, Dominican Republic: MDPI. DOI: 10.3390/proceedings2191261 (cit. on pp. 40–42).

Keras - Library for neural networks (2019). URL: keras.io (visited on 06/05/2019) (cit. on pp. 70, 107).

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization." In: URL: arxiv.org/pdf/1412.6980.pdf (cit. on p. 28).

Lago, Paula and Sozu Inoue (Dec. 2018). "A Hybrid Model Using Hidden Markov Chain and Logic Model for Daily Living Activity Recognition." In: *The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018)*. Vol. 2. 19 1266. Punta Cana, Dominican Republic: MDPI. DOI: 10.3390/proceedings2191266 (cit. on pp. 40, 41).

Machine Learning Mastery - How to choose loss functions when training deep learning neural networks (2019). URL: machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ (visited on 12/23/2019) (cit. on p. 26).

Machine Learning Mastery - Machine Learning Algorithms for Human Activity Recognition (2019). URL: machinelearningmastery.com/evaluate-machine-learning-algorithms-for-human-activity-recognition/ (visited on 11/29/2019) (cit. on p. 20).

Matplotlib - Plotting in Python (2019). URL: matplotlib.org (visited on 05/23/2019) (cit. on pp. 53, 70, 107).

Node-RED: Low-code programming for event-driven applications (2019). URL: nodered.org (visited on 10/11/2019) (cit. on pp. 70, 107).

NumPy - Python Package for Scientific Computing (2019). URL: numpy.org (visited on 11/08/2019) (cit. on pp. 70, 107).

Pandas - Python Data Analysis Library (2019). URL: pandas.pydata.org (visited on 05/23/2019) (cit. on pp. 53, 70, 107).

Paola, Alessandra De et al. (Jan. 2017). "A Context-Aware System for Ambient Assisted Living." In: *Ubiquitous Computing and Ambient Intelligence: 11th International Conference (UCAmI 2017)*. Vol. 10586, pp. 426–438. ISBN: 978-3-319-67584-8. DOI: 10.1007/978-3-319-67585-5_44 (cit. on p. 12).

Patterson, Josh and Adam Gibson (2017). *Deep Learning*. O'Reilly. ISBN: 9781491914250 (cit. on pp. 26, 35, 36, 38).

Plotly - Python Graphing Library (2019). URL: plot.ly (visited on 11/08/2019) (cit. on pp. 70, 107).

Priyadarshini, Sushree Bibhuprada B., Amiya Bhusan Bagjadab, and Brojo Kishore Mishra (2019). "The Role of IoT and Big Data in Modern Technological Arena: A Comprehensive Study." In: vol. 154. Intelligent Systems Reference Library. Springer International Publishing. Chap. 2,

pp. 13–25. ISBN: 9783030042035. DOI: https://doi.org/10.1007/978-3-030-04203-5_2 (cit. on p. 51).

Python Programming Language (2019). URL: python.org (visited on 05/23/2019) (cit. on pp. 53, 70, 107).

Rashid, Tariq (2017). *Neuronale Netze selbst programmieren*. O'Reilly. ISBN: 9783960090434 (cit. on pp. 22, 28, 29, 37, 38).

Razzaq, Muhammad Asif et al. (Dec. 2018). "Multimodal Sensor Data Fusion for Activity Recognition Using Filtered Classifier." In: *The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018)*. Vol. 2. 19 1262. Punta Cana, Dominican Republic: MDPI. DOI: 10.3390/proceedings2191262 (cit. on pp. 41, 43).

Razzaq, Muhammad Asif et al. (n.d.). "Multimodal Sensor Data Fusion for Activity Recognition Using Filtered Classifier." In: DOI: 10.3390/proceedings2191262 (cit. on pp. 44, 45).

Rolls, Edmund T. and Alessandro Treves (1998). *Neuronal Netzworks and Brain Function*. Oxford University Press. ISBN: 9780198524335 (cit. on pp. 18, 23).

Rosenblatt, Frank (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6, pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: dx.doi.org/10.1037/h0042519 (cit. on p. 23).

Salomón, Sergio and Cristina Tîrnăucă (Dec. 2018). "Human Activity Recognition through Weighted Finite Automata." In: *The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018)*. Vol. 2. 19 1263. Punta Cana, Dominican Republic: MDPI. DOI: 10.3390/proceedings2191263 (cit. on pp. 41, 42).

scikit-learn - Machine Learning in Python (2019). URL: scikit-learn.org (visited on 05/23/2019) (cit. on pp. 70, 107).

Semantic Scholar (2019). URL: semanticscholar.org (visited on 04/09/2019) (cit. on p. 39).

Sensors - MDPI Open Access Journal (2019). URL: mdpi.com/journal/sensors (visited on 04/09/2019) (cit. on p. 39).

Sharma, Neha, Madhavi Shamkuwar, and Inderjit Singh (2019). "The Histroy, Present and Future with IoT." In: vol. 154. Intelligent Systems Reference Library. Springer International Publishing. Chap. 3, pp. 27–51. ISBN: 9783030042035. DOI: https://doi.org/10.1007/978-3-030-04203-5_3 (cit. on pp. 11–13).

Singh, Rajesh et al. (2019). "Internet of Things Enabled Robot Based Smart Room Automation and Localization System." In: vol. 154. Intelligent Systems Reference Library. Springer International Publishing. Chap. 6, pp. 105–133. ISBN: 9783030042035. DOI: https://doi.org/10.1007/978-3-030-04203-5_6 (cit. on p. 12).

Talos - Hyperparameter Optimization for Keras Models (2019). URL: github.com/autonomio/talos (visited on 06/14/2019) (cit. on pp. 70, 74, 107).

TensorBoard - Visualisation for Tensorflow and Keras (2019). URL: www.tensorflow.org/guide/summaries_and_tensorboard (visited on 06/27/2019) (cit. on p. 91).

TensorFlow - Open Source Machine Learning Platform (2019). URL: tensorflow.org (visited on 11/08/2019) (cit. on pp. 70, 107).

The 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018) (Dec. 2018). Vol. 2. 19. Punta Cana, Dominican Republic: MDPI.

Towards Data Science - Data Types in Statistics (2019). URL: towardsdatascience.com/data-types-in-statistics-347e152e8bee (visited on 12/12/2019) (cit. on p. 22).

Turing, Alan M. (1950). "Computing Machinery and Intelligence." In: *Mind* 59.October, pp. 433–60. DOI: 10.1093/mind/LIX.236.433 (cit. on p. 2).

Ulster Institutional Repository, Ulster University (2019). URL: uir.ulster.ac.uk/ (visited on 04/09/2019) (cit. on p. 39).

Weisstein, Eric W. (n.d.[a]). *Heaviside Step Function*. From MathWorld - A Wolfram Web Resource. URL: mathworld.wolfram.com/HeavisideStepFunction.html (cit. on p. 25).

Weisstein, Eric W. (n.d.[b]). *Sigmoid Function*. From MathWorld - A Wolfram Web Resource. URL: mathworld.wolfram.com/SigmoidFunction.html (cit. on p. 25).

Weka 3: Data Mining Software in Java (2019). URL: cs.waikato.ac.nz/ml/weka/ (visited on 04/09/2019) (cit. on p. 43).

Wirth, Rüdiger and Jochen Hipp (2000). "CRISP-DM: Towards a Standard Process Model for DataMining." In: *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pp. 29–39. URL: pdfs.semanticscholar.org/48b9/293cfd4297f855867ca278f7069abc6a9c24.pdf (cit. on p. 43).

Zhou, Yi Tao and Rama Chellappa (1988). "Computation of optical flow using a neural network." In: *IEEE 1988 International Conference on Neural Networks* 2, pp. 71–78 (cit. on p. 34).