Master Thesis

Data Driven Gesture Analysis

Jörg Simon

Graz, 2019

Institute of Interactive Systems and Data Science
Graz University of Technology

Supervisor/First reviewer: Dipl.-Ing. Dr. techn. Eduardo Veas

Abstract (English)

In this thesis, we present a system to recognise natural appearing gestures using a self build smartglove prototype. We explain the nature of gestures and the anatomy of the human arm and go into the theory of gesture recognition. A user study is used as a basis of a data-driven approach to gesture recognition, where all possible features from human activity recognition are generated, and automatic methods to select a good set of features are explored. We extend this approach even further with a novel algorithm for selecting sensors for a specific target system. Recursive Sensor Elimination (*RSE*) selects sensors recursively using a heuristic function to find the best configuration for a given subset of gestures. We explain the use cases, the detail of the *RSE* algorithm and first experimental results. It shows the problems when someone tries to apply the insights of this work to consumer hardware in the form of a smartwatch experiment and which design decision have to be made. Within this experiment, it presents a possible method to augment IMU time series data if the labels are not corrupted by speeding up or slowing down the time series and adding some noise. With this, it is possible to train a simple system to allow steering f.e. a slide set with your watch.

Abstract (German)

In dieser Arbeit stelle ich ein System für die Erkennung natürlich vorkommender Gesten vor, welche sich eines selbst gebauten Smart-Glove Prototyp bedient. Ich gehe auf den Hintergrund von Gesten, die Anatomie der menschlichen Hand und die Theorie von Gestenerkennung ein. Ich stelle eine Studie vor, welche ich mit Kollegen durchgeführt habe und zeige einen eigenen Daten-Getriebenen Ansatz welcher wettbewerbsfähig mit den Resultaten der Studie ist. Ich erweitere diesen Ansatz, um einzelne Sensoren im Handschuh auf ihre Nützlichkeit für die Erkennung zu quantifizieren. Der neu eingeführte Recursive Sensor Elimination (*RSE*) Algorithmus ermöglicht es, Sensoren für ein gewisses Set von Gesten zu selektieren und so das Hardwaredesign datengetrieben weiter zu führen. Man kann dann einen Smart-Glove mit dem reduzierten Sensorset schöner und billiger bauen. Zu guter letzt schaue ich mir die Machbarkeit von Gestenerkennung auf einer Smart Watch an und gehe auf die Einschänkungen ein, wenn man die Erkentnisse der Smart-Glove Studie auf im Konsum erhältlicher Hardware portieren möchte. In diesem Rahmen stelle ich auch eine Methode für die Erschaffung künstlicher Trainingsdaten für IMUs vor. Ich zeige, dass es möglich ist mit Machine Learning ein simples System zu trainieren um eine Präsentation zu steuern. Dank einer entwickelten Datenaugmentierung kann das auch mit vergleichsweise wenig Aufwand im Datensammeln funktionieren.

Acknowledgement

I want to thank especially Prof. Stefanie Lindstaed who brought me to the Know-Center ages ago and supported me and continues to support me trough my carrier there. On the same level I want to thank Dr. Eduardo who is not only a really good Boss and brilliant researcher, but also a very supportive and empathic person. He pushed me to finish this work a lot, and I want to say thanks for the drive you did build up in me. Granit Luzhnica was the one who, with the help of Eduardo, started the journy to the Smart-Glove together with me. He did the first modelling with the dataset and our exchange on this topic and on the topic on machine learning and wearables is always enlighting. I want to thank Vedran Sabol for his support in the team, and his diplomatic feeling. I want to thank Christoffer Öjeling who is now making his career at Google, who brought a lot of clever ideas into the project, and did the hardware assembly. I was always fun discussing with him. I also want to thank all my friends and family and everyone who did provide input and critique but especially I want to thank my wife to support me writing that thesis beside work and was also my first private editor. You are all awesome!

Jörg Simon
Graz, 2019

Statutory Declaration

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 marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Place, Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Ort, Datum Unterschrift

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Research in Human-Computer Interaction (HCI) always strived for systems where you do not need much energy to find out how to perform a specific interaction, coined in the phrase ”don’t make me think” from the same-titled book by Steve Krug [Krug, 2005]. A common hypothesis is, that naturally appearing behaviours like gestures[1] in interaction are a way to achieve that goal.

Natural interactions (like in Latoschik [2005]) through speech and gestures, however, were for a long time not traceable for real systems, as the effort to create such a system limited the tools to particular use cases in the lab. Speech and even gestures are just too complex to be programmed by a classical deterministic algorithm; instead, you need a system which can recognise patterns in data from the interaction/the user and react based on the detected pattern. To perform the interactions based on recognised patterns in data is often called data-driven interactions. Achievements in the field of machine learning, digital signal processing and sensor technology, together with the shrinking of technology f.e. in the form of smartphones, are enablers of such systems of data-driven interactions so that they may become more mainstream.

Gesture recognition is the automatic recognition of a computer system of a gesture performed by a person, often for interaction with a computer system. The basis of this thesis is a project about gesture recognition published in the proceedings of 3DUI’16 [Luzhnica et al., 2016]. This project was done by Granit Luzhnica, Christoffer Öjeling and me under supervision of Eduardo Veas. In it, a custom smartglove is built with many sensors to capture physical motion. In a user study participants repeatedly perform 31 gestures with this glove. With a sliding window approach, specific evaluation and a hand-selected set of features, these 31 gestures (plus a zero class) are recognised with 98.6% accuracy (f1-measure).

1.1. Problem Definition and Solutions

In Luzhnica et al. [2016] there is no in-depth description of the smartglove. The paper explains the number of sensors used and shows that this number was grater than in other similar smartgloves at that time, but it does not describe why exactly the sensors are placed at certain positions. Also, it does not explain the raw data gathered from the user study or explain which steps for data quality are done before feature engineering and modelling. This work complements the paper by providing this missing documentation.

The validation and test split are done by selecting a random subset of the data in Luzhnica et al. [2016]. Since there are overlaps in the windows, this means the training data has ”seen” some of the validation and test data. Do our reported numbers still hold? Also, in the paper, we hand-select features based on our expert knowledge. Are there other, better combinations of features? In this work, a different validation and test split are used, which guarantees that there are no overlaps between train, validation and test set. With

[1] http://www.digitaltrends.com/computing/gesture-control-q-a-thalmic-labs-aaron-grant/

this different split still, similar accuracies are achieved. Feature engineering is also done differently, by creating all kind of features taken from the literature and using algorithmic approaches to select a proper subset of the features which still give good accuracy. I refer to this approach as a data-driven approach to feature engineering. More features than in Luzhnica et al. [2016] do not impact the accuracy of the recognition. I build a system with way fewer features with similar accuracy.

An important issue on building a custom sensor glove is the hardware design. Prototyping hardware is a long and costly effort. There is a tradeoff in building an easy to build system and a system with good accuracy when using data-driven interactions. It is often not clear on what position, how many and which sensors are needed for a specific recognition model. However, modifying a design later because of missing components/sensors is problematic since you have to redo the whole electronics development process[2]. I extend the data-driven approach to feature engineering to make a greedy sensor selection. If the original prototype is one with all sensors and the data collection is one with all gestures this approach can give a set of possible configurations to choose from, which reaches a good tradeoff between model accuracy and sensors needed.

Gesture recognition with a smartglove has one big drawback: There are not many existing smartgloves used outside research. Can the methods of gesture recognition described in Luzhnica et al. [2016] and the extensions be used in some commonly available hardware? Smartwatches are mainstream now. The company IDC[3]forecasts 237M wearables by 2020, of which 82M should be smartwatches. Every smartwatch has similar sensors for motion to our smartglove. In this work, I present a demo system that uses the same methodology to gesture recognition used for the glove. I describe the different constraints of implementing such a system.

The motion sensor of a smartwatch is usually an inertial measurement unit (IMU) with six degrees of freedom (6 DOF), which is similar to the ones we use in the glove. They are not the same so gathering a new training set is needed. Data augmentation is a method to create additional data by transforming the original data without invalidating the labels. I present a method for data augmentation of IMUs which is suitable for the gesture recognition task.

1.2. Contributions

The domain of this work is gesture recognition using physical motion sensors. I base the work on the project/paper we did in Luzhnica et al. [2016]. To this work I complement in this thesis the following:

- Provide more exact documentation of the smartglove (and the placement of sensors) than in Luzhnica et al. [2016]
- Explore the raw data of the study and explain data cleaning
- Validate the results of Luzhnica et al. [2016] with a different validation and test split
- Compare the feature engineering by manually selecting features in Luzhnica et al. [2016] to a data-driven approach of generating many features and use feature selection

I also add the following new aspects to the domain of gesture recognition.

- Provide an approach to sensor selection so a prototype with many sensors can be used to produce a product with the right amount of sensors for the use case
- Demonstrate that it is possible to use the approaches from them smartglove to make an application for a smartwatch
- Implement a data augmentation technique for inertial measurement units for gesture recognition.

[2] https://predictabledesigns.com/how-to-develop-and-prototype-a-new-product/
[3] https://www.idc.com/getdoc.jsp?containerId=prUS41100116

Chapter 2

Anatomy of the Human Arm and Gestures

Parts of the contents of this chapter have been published in the proceedings of 3DUI'16 *[Luzhnica et al., 2016].*

In this thesis, we look at data-driven gesture recognition. Gestures are performed with the human arm, and its freedom in movement and anatomy have essential impacts on gesture recognition. Using models of the human arm and grammar to detect gestures is a traditional way for gesture recognition [Garg et al., 2009]. For the data-driven approach, it is crucial that all movements are captured and possible are redundantly captured by different means of sensors. In the following section, we look into the anatomy of the human arm and its degree of movement. We argue that a data-driven approach based on the dynamics of the arm needs to capture more degrees of freedom that exist on the arm in reality. We then explain what this work understands for a gesture.

2.1. Anatomy of the Human Arm

The human arm is widely used as a communication tool. Most explicitly in sign language, but also an important part of non-verbal communication [Mehrabian, 1972]. Medically the human arm is a limb of the upper body of a human. It consists of the upper arm, the lower arm and the hand. For user use case we will concentrate on the lower arm and the hand. The arm is also an important tool allowing us to build and use tools for daily life. This movement is constraint by the anatomy of the arm.

The human hand consists of 29 bones including the radius and the ulna bone. There are at least 123 named ligaments which define the range of motion in the multitude of joints. Three major nerves supply the 18 muscles in the forearm and the 17 muscles in the palm.

There are five main articulations in the human hand. The wrist or radiocarpal joint which connects the forearm to the hand, the intercarpal and mediocarpal articulations, the metacarpophalangeal articulations where the digits meet the palm and the interphalangeal articulations which are the hinge joints between the bones of the digits.

The movement of the hand can be broken down into many small motions the happen in the articulations mentioned above. Table 2.1 provides an overview of the typical range of motion of the major joints [Grifka and Kuster, 2011; Waldeyer et al., 2012]. Another good source to get an overview is a presentation on the human hand[1].

Based on the insights of anatomy Cobos et al. [2010] developed a mathematical model to capture the kinematics of the hand. It accurately describes the degrees of freedom of the human hand. The full model

[1] http://www.mccc.edu/~behrensb/documents/Structurefunctionofthehandbjb.pdf

Joint	Movement with ROM in degrees
Forearm	Pronation (inwards turn) 70°
	Supination (outwards turn) 85°
Wrist	Extension 70°
	Flexion 70°
	Radial (rotation towards the radial bone) 20°
	Ulnar (rotation towards the ulnar bone) 35°
Thumb basal joint	Palmar Adduction, Contact
	Palmar Abduction, 45°
	Radial Adduction, Contact
	Palmar Abduction, 60°
Thumb interphalangeal joint	Hyperextension 15°H
	Flexion 80°
Thumb metacarpophalangeal joint	Hyperextension 10°
	Flexion 55°
Distal interphalangeal finger joint	Hyperextension 0°
	Flexion 80°
Proximal interphalangeal finger joint	Hyperextension 0°
	Flexion 100°
Metacarpophalangel finger joint	Hyperextension 0°-45°H
	Flexion 90°

Table 2.1.: Typical Range of Motion (ROM)

has 24 *DOF* (figure 2.1). Also, it shows that a model with only 9 *DOF* can simulate a correct hand movement with an error of 5% and one with 6 *DOF* shows a 10% error. This model does not include the 4 *DOF* of the wrist movement. Using sensors which can capture the physical movement, it must be possible to recover the correct physical movement of the hand, a process called inverse kinematics. To fully capture physical movement it makes sense to have sensors which can at least capture this amount of degrees of freedom. The goal of the prototype of the smartglove is to have more sensors than needed and perform sensor selection. The smartglove has 2 *DOF* per finger in flexion sensors and an IMU with 6 *DOF* on the fingertip. It has 3 *DOF* in flexion sensors for the thumb and also a 6 *DOF* IMU. Additionally, it has a 6 *DOF* IMU on the palm and the wrist, a 3 *DOF* magnetometer and two more flexion sensors on the wrist. All these sensors make a sensor setup which can capture 52 *DOF* of movement. The smartwatch on the other side only captures 6 *DOF* and none of the finger or thumb based degrees of freedom are captured by the watch.

At the beginning of our study in [Luzhnica et al., 2016], various data gloves had already been available commercially. All of them emphasise flexion sensors in fingers, and thus focus on hand postures. In contrast, our glove contains both flextion and motions sensors (gyroscope + accelerometer) on each finger, thus focussing more on hand motion, i.e. the dynamic aspects of gestures. The MiMu Glove[2] is used to produce music by some means of gesture detection. It employs one IMU at the wrist, 4 bend sensors at the fingers and vibrators at the underarm to provide haptic feedback.. It is not clear if the IMU is 6 *DOF* or 9 *DOF*. The MiMu glove thos has $10-13$ *DOF*. Fifth Dimensional Technologies[3] offers three gloves that are equipped with bend sensors and abduction sensors between fingers. These 5DT Data Gloves has 5,14 of 16 *DOF*. CyberGlove Systems [4] offers CyberGlove II equipped with two bend sensors on each finger, four abduction sensors, sensors measuring thumb crossover, palm arch, wrist flexion and wrist abduction, giving 18 *DOF*. Virtual Labs[5] offers a range of data glove products (VMG Lite, VMG 10, VMG30, VMG 30 Plus), all of them equipped with bend sensors on the fingers, 9 *DOF* orientation sensors for hand and wrist, as well as tactile feedback vibrators. This gives up to 32 *DOF*.

2.2. Gestures

Intuitively many people would define gestures as an embodied means of communication, consisting mainly of arm movements. In an extreme form, this leads to sign languages, which can form complete sentences (see Birk et al. [1997]; Huang and Huang [1998]; Oz and Leu [2011]; Takahashi and Kishino [1991]). Several gloves for auto-translating sign language exist as prototypes[6].

Differentiating between gestures and gesticulation is important. A gesticulation is a form of a freehand movement supporting other expressions like your current speech. Gesticulation is highly personal, meaning the same motion can mean different things for other humans or in other context of speech [Wexelblat, 1995]

A gesture can be a specific posture of the hand conveying a meaning. An example is a thumbs up gesture which can mean approval. A gesture can also be a hand posture combined with arm movement where the exact path of the movement is important. These gestures are often used for direct manipulation. An example would be a pointing gesture (see figure 2.2). Many works list gestures and categorise them into a taxonomy like Bohm et al. [1992]; Koons et al. [1993]; Pavlovic et al. [1997]; Wobbrock et al. [2009]. Many of these gestures are artificial gestures for control. It is possible only to use artificial gestures for interaction with a computer. A study looking at inventing custom gestures Jego et al. [2013] showed, however, that a user can only remember a very small number (about two) of such artificial gestures. Therefore, we are looking at gestures that are widely known, even though there may be cultural differences regarding their popularity and meaning. Additionally, there should be a plausible relationship between the gesture and the resulting interaction between human and computer. We build our set of gestures on the list of 22 natural gestures described in Glomb et al. [2012]. We added the following: The numbers one to five, as they would be useful to select items; popular touch-based swipe gestures such as swipe left, right, up and down, as these would be useful for navigation. Finally, we added lateral grasp (Grasp 2) and palmar grasp (Grasp 1) gestures, as we think that grasping objects would be useful in interaction with 3D virtual objects.

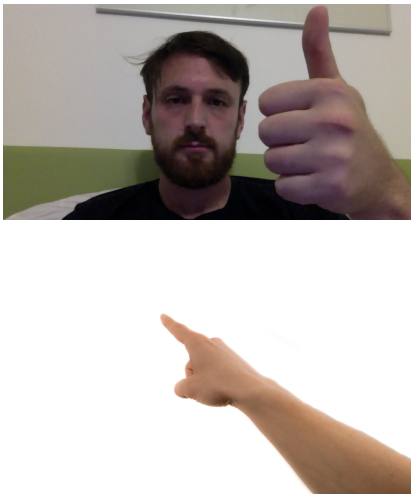


Figure 2.2.: Example of two gestures in our dataset

[6] f.e. `http://enabletalk.com/prototype.html`

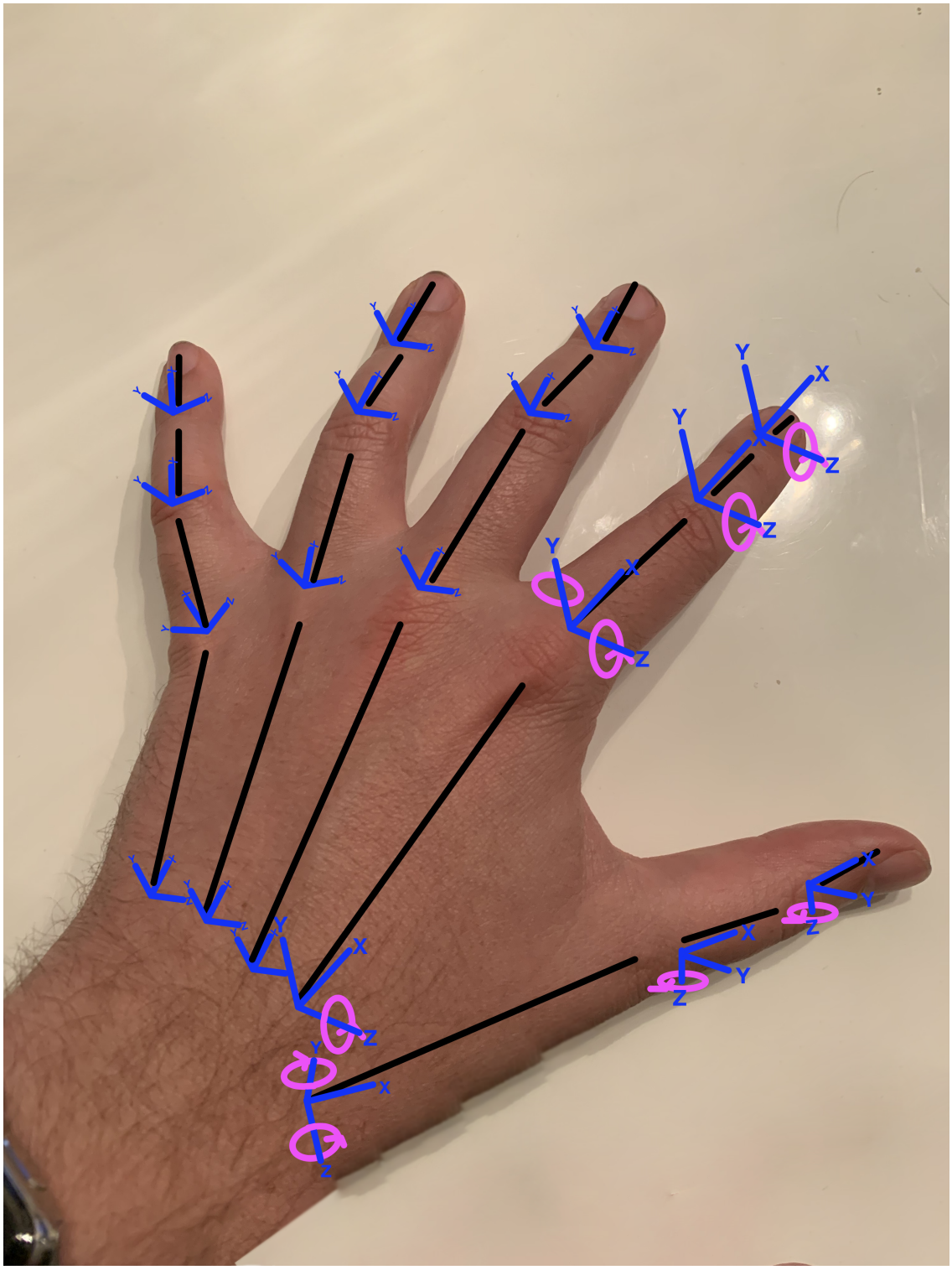


Figure 2.1.: Joints and *DOF* of the human hand based on [Cobos et al., 2010]. The blue arrows show the reference frame. Black lines show a joint or bone inside the hand model. The purple circles show the possible movement for a joint from the reference frame. This circle is also the degree of freedom (*DOF*) of that joint. For clarity only the degrees of freedom of the thumb (4 *DOF*) and the degrees of freedom for one finger (5 *DOF*) is shown, the other fingers have the same DOF. This model has in total 24 *DOF*.

Chapter 3

Gesture Recognition

Parts of the contents of this chapter have been published in the proceedings of 3DUI'16 *[Luzhnica et al., 2016].*

Gesture recognition defines the process of automatically recognising if a human performs a gesture through a computer system. The methods are often dependent on the sensors to capture the gestures. Gesture recognition has gained interest as basis for gesture-based interaction in a wide range of use cases, such as crisis management [Wachs et al., 2011], TV remote controlling [Ren and O'Neill, 2013], interacting with computers [Kenn et al., 2007; Kumar et al., 2012], gaming interfaces [Kulshreshth and LaViola, 2015; Martins et al., 2008; Wachs et al., 2011; Zhang et al., 2009], augmented reality applications [Kavakli et al., 2007; Tsai et al., 2015; Weissmann and Salomon, 1999; Xu, 2006], hands-free interaction in-car driving [Molchanov et al., 2015], providing virtual training for car driving [Xu, 2006] or detecting a driver's fatigue [Lee et al., 2015]. In the medical area, robot nurses are envisioned to detect a surgeon's hand gestures and to assist with the necessary surgical instrument [Wachs et al., 2011]. In another type of use case, computer systems detect gestures to understand user activities. For instance, robots have been envisioned to analyse gestures to track task completion to be able to seamlessly take over with the next steps [Coupeté et al., 2015; Roitberg et al., 2015]. Sometimes, it is useful to only observe and document the gestures, as in the case of assembly lines to document the work for quality assurance [Stiefmeier et al., 2008]. The goal to detect assembly line tasks is an area of active research [Koskim"aki et al., 2013; Ou et al., 2006; Ward et al., 2006; Zappi et al., 2007]. Gesture recognition has also been explored in the context of logging activities of daily life: In [Sen et al., 2015], the authors explore the possibility to detect eating habits via recognising the gestures for eating and drinking (bringing the hand to the mouth). In [Shoaib et al., 2015], activity logging based on both smartwatch and smartphone sensing is used to detect drinking too much coffee or not eating.

Different devices and sensors can be used to capture a gesture and perform gesture recognition. Depth cameras and ordinary cameras are state of the art. Controllers specific for virtual reality (VR) allow gesture-based interactions. There exist more exotic controllers like the infrared-based leapmotion[1]to control a computer. Smartgloves are even less common but offer advantages when you need to move freely and still want the ability to express complex gestures. There exists a smartglove used to analyse combat sports [Navas et al., 2012]. Company's like ProGlove[2]already provide gloves for industrial settings, letting workers scan their tools and materials with NFC/RFID to document work. The MiMu Glove[3]can produce music using gesture recognition. The Keyglove System[4]uses several force sensors to detect touches on the glove to simulate a keyboard. In the medical field, smartgloves are explored to help in the rehabilitation of tremor [Kazi et al., 2010], arthritis [O'Flynn et al., 2013] or to guide blind people [Bernieri et al., 2015; Ugulino and Fuks, 2015]. Another prominent use is the translation of sign language into spoken

[1] https://www.leapmotion.com
[2] http://www.proglove.de
[3] http://mimugloves.com
[4] http://www.keyglove.net/

words, allowing deaf-mute people to communicate more easily (see Praveen et al. [2014] or the Enabletalk Project[5]). This widespread interest means that augmenting gloves with technology and use their data for private or professional activities is not science fiction, but a new trend that is currently happening.

A less explored tool for gesture recognition is the smartwatch. While there exist initiatives like Google's project Soli, which might finally be allowed on the market this year, the sensors generally are not perfectly fit for very expressive gesture recognition. Research usually adds other hardware to the watch like a depth camera to allow gestures performed hovering over the watch [Han et al., 2015] or write text [Van Vlaenderen et al., 2015]. Still, a lot of interesting movements can be recognised [Xu et al., 2015], and while all the other hardware options are not mainstream, smartwatches and the, from the sensor perspective, similarly capable fitness trackers, are.

3.1. Camera Based Gesture Recognition

Typically a camera is mounted in the environment to record human hands. The system extracts features from the individual frames of the recording. Sometimes there is a filtering process involved which removes unwanted objects like, f.e. heads from the image or video [Dardas and Georganas, 2011]. There are two options to proceed: Traditionally the system is composed of modules. A modular system first predicts postures [Chen et al., 2007; Dardas and Georganas, 2011] and then a grammar of sequences of postures is constructed to recognise gestures [Garg et al., 2009]. For instance, in Dardas and Georganas [2011], the authors first detect and track hands and then recognise ten postures with an accuracy of 96% in camera images with a multi-class support vector machine (SVM).

Similarly, in Chen et al. [2007] a single webcam is used as a source, from which the authors extract Haar-like features and use AdaBoost to discriminate between four postures: two fingers, palm, fist, little finger. The authors achieve an accuracy of over 90%. In Birk et al. [1997], the authors achieve an accuracy of 99% by using principle component analysis (PCA) and a Euclidian distance based classifier to recognise 25 international hand alphabet postures from images of the gestures. An alternative vision based approach is to use coloured gloves in which different parts of the hand are marked with different colours, making it much easier to track gestures [Hasan and Mishra, 2012]. Using a depth camera is also an option.

More recent systems learn the gestures end-to-end. Yin and Davis [2014] uses a Microsoft Kinect, and Hierarchical Hidden Markov Models (HHMMs) to model gestures end to end. End to end modelling has recently become more popular with deep learning. First works on deep learning in gesture recognition still used some preprocessing. Lin et al. [2014] first learns to transform the skin colour to a neutral colour and translates the hand into a neutral position. The system then uses a convolutional neural network (CNNs) to perform gesture recognition with 95.5% accuracy using seven gestures from seven subjects. Huang et al. [2015] uses a 3D CNN to learn American sign language with a Microsoft Kinect. Núñez et al. [2018] uses a convolutional neural network combined with a long short-term memory network (CNN+LSTM) over a video stream to perform human activity recognition (HAR) and gesture recognition. They analyse different gesture datasets and achieve accuracies between $79\% - 86\%$ using datasets of around $20 - 30$ gestures. Yang et al. [2018] allows social robots to track gestures in real time by first extracting a region of interest (ROI) with the OpenCV library and then performing gesture recognition with a CNN. Perera et al. [2019] shows that on a dataset of 13 gestures used to control unmanned aerial vehicle (UAVs) like drones it is possible to achieve 91.9% accuracy using a CNN. Using an event-based camera and an implementation of a CNN for spiking neural networks Amir et al. [2017] could implement a system which reaches 96.5% out-of-sample accuracy with 11 gestures from 29 subjects.

[5] http://enabletalk.com/prototype.html

3.2. Wearable Based Gesture Recognition

The majority of wearable sensor systems for gesture detection are gloves equipped with sensors. In most research endeavours, gloves are custom-built. In Murakami and Taguchi [1991], a recurrent neural network (RNN) is used to recognise the following Japanese sign language gestures: father, mother, brother, sister, memorise, forget, like, hate, skilled and unskilled. The gestures are 42 postures representing Japanese letter alphabet with an accuracy of 96%. The data were generated using a VPLDataGlove. In Xu [2006], the authors use a feed-forward neural network capable of distinguishing between 15 gestures with an accuracy of 98%. They record the data with a CyberGlove[6]with 18 sensors. More recently, a feed-forward neural network was used to construct a hand gesture recognition system for interacting with robots [Neto et al., 2013]. Using data from CyberGlove II (providing 22 joint-angle measurements), the authors were able to recognise 10 different artificial gestures with an accuracy of 99.8% and 30 gestures with an accuracy of 96.3%. In this work, a first step is segmenting the data stream into segments of different size so that each segment contains exactly one gesture. These segments are then used for the classifier. In Zhang et al. [2009], data from electromyography (EMG) sensors and a wrist-worn accelerometer were used to build a system that recognises 18 gestures with an accuracy of 91.7%. The defined gestures were used to play a virtual rubric's cube game. In Romaszewski et al. [2014], authors used a list of 22 natural hand gestures. The gestures originally stem from Glomb et al. [2012]. While analysing the data, the authors first resampled and interpolated the data. They then used LDA to discriminate between the resampled segments with an accuracy of 92.8%. This changing of segments shows that there is a clear separability between those gestures. However, in a live recognition system, this resampling and interpolation is not possible unless the start and end time of the gesture that needs to be recognised can be detected.

Recently, there has been growing research interest in gesture recognition based on consumer good technology like smartwatches (e.g., [Han et al., 2015; Van Vlaenderen et al., 2015; Xu et al., 2015; Zhao et al., 2015]). In Xu et al. [2015], the authors report the classification of 37 interaction oriented gestures, i.e. gestures that are intended to be used for controlling other devices (turning the arm, simulating a click, pinch to zoom, etc..). The gestures are detected based on smartwatch sensor data only and with an accuracy of 98% by using the Naive Bayes algorithm. The authors report different numbers in a subsequent demo paper, namely 27 gestures with 96% accuracy using Logistic Regression or Decision Trees [Zhao et al., 2015]. However, the data in the latter paper were collected only by a single participant, and the participant performed gestures from a fixed arm position.

When comparing camera and wearable based approaches, vision-based systems are more sensitive to the environment. Lighting conditions, scene and background details are issues that affect such systems [Wachs et al., 2011]. In the case of cameras, there might also be privacy issues; and different countries have different regulations concerning video recordings in non-private environments. Wearable sensor-based systems, primarily glove based ones, can be uncommfortable [Wachs et al., 2011] or even pose a hygienic problem [Kiruthika et al., 2014]. On the other hand, wearable technologies provide, in principle, the possibility for higher privacy as the data are a priori more anonymous than pictures or videos. When it comes to accuracy, many authors report very high accuracies (in the higher 90s) for the selected set of gestures using either technique [Birk et al., 1997; Dardas and Georganas, 2011; Neto et al., 2013; Xu, 2006].

3.3. Human Activity Recognition (HAR) Based Gesture Recognition

In the paper of Luzhnica et al. [2016] and this thesis, we take a wearable sensors approach. In contrast to some other works, we emphasise capturing the dynamics of the gestures, and present gesture recognition using a custom data glove. Our work also differs from previous research since we take a sliding window approach in combination with dual labelling in the test set. A sliding window is a technique for data preprocessing in which information is extracted (statistics, aggregates, features, etc...) over a "sliding window" that contains a fixed number of samples. To the best of our knowledge, this paper was the first using a sliding window approach for gesture recognition. However, this approach is very common in human activity

[6] http://www.cyberglovesystems.com/

recognition (HAR) [Koskimaki et al., 2009; Krishnan and Cook, 2014; Ortiz Laguna et al., 2011].

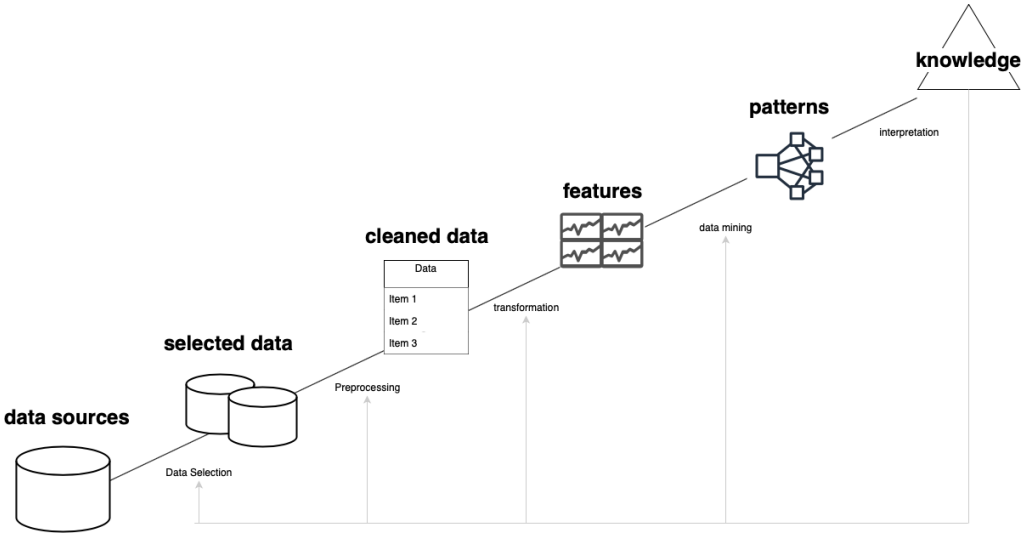


Figure 3.1.: Knowledge Discovery and Data Mining Process (KDDM)

There are several descriptions of the steps on how to build any recognition system with machine learning. The most general process is the process of knowledge discovery and data mining (KDDM). The KDDM process (see figure 3.1) describes the data perspective of the process [Fayyad et al., 1996].

In the first step, a data set is selected from various sources. We want to use a model with more physical motion sensors than needed. Such a dataset is not available, so we build our own in a user study described later. To have a comparison to the smartglove, the smartwatch data is collected similarly to the original user study.

The data we collect comes in the form of time series. Time series data is data where instances are data points at a specific time. These data points are multi-dimensional vectors with each data point corresponding to channels at that specific time [Zhou and De la Torre, 2012]. Framed differently, a data point has the particular sensor values sampled at that specific time point. This form of the dataset also means that instances in the dataset are not independent of each other, which is often a basis for the classification. Time series data is captured in our case in a data matrix $D \in \mathbb{R}^{(n_d,m_d)}$ with n_d being the number of data points captured in time and m_d being the number of channels captured by the sensors. The subscript d denotes that these are the raw data dimensions, instead of the dataset after feature engineering. Additionally a vector $\vec{y} \in \mathbb{R}^{(n_d,1)}$ exist which annotates each timestep with the dependent variable like the class type.

$$D \in \mathbb{R}^{(n_d,m_d)} = \begin{pmatrix} x_{(0_d,0_d)} & x_{(0_d,1_d)} & \cdots & x_{(0_d,m_d)} \\ x_{(1_d,0_d)} & x_{(1_d,1_d)} & \cdots & x_{(1_d,m_d)} \\ x_{(2_d,0_d)} & x_{(2_d,1_d)} & \cdots & x_{(2_d,m_d)} \\ \cdots & \cdots & \cdots & \cdots \\ x_{(n_d,0_d)} & x_{(n_d,1_d)} & \cdots & x_{(n_d,m_d)} \end{pmatrix} \tag{3.1}$$

The next step is to perform preprocessing on the data to get clean data. Preprocessing means removing data which does not meat certain quality criteria, performing outlier detection [Breunig et al., 2000; Liu

et al., 2012] and filling missing values [Barnard and Meng, 1999]. All this needs to be done with the dataset from the study and the dataset from the smartwatch. Since it is possible to interrupt the data collection at the user study with the smart glove it is especially important to see if the labels are valid and if not remove that data. In the smartwatch case there can be artefacts of earlier recordings and timing issues which need to be removed.

Machine learning models often need the data in a certain form so they can find patterns. For our task

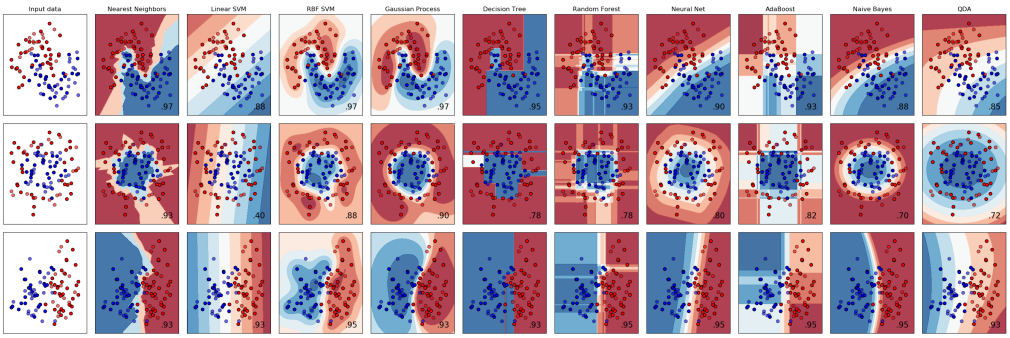


Figure 3.2.: Comparison of what decision boundaries a model can learn given data. The image is taken from scikit-learn.org

to recognise gestures, the model is a classification model. Given data, it predicts which kind of class (gesture) the data is from. It does so by learning a decision boundary separating instances of the different classes in data. Linear methods (see Bishop [2006] Chapter 4.3.4 or Crammer et al. [2006]; Xu [2011]; Zhang [2004].) need the data to be linearly separateable. The machine learning model can find these linear hyperplanes in the data, but it can not transform the data itself except for methods which employ kernels [Boser et al., 1992; Rasmussen and Williams, 2005; Vapnik and Lerner, 1963] to do part of the feature engineering. Clustering methods need data of classes to be within a certain distance or density. Some methods like linear discriminant analysis (LDA) and principal component analysis (PCA) have problems if channels of the data are linearly dependent on each other or if data is highly redundant. The transformation of the clean data is called feature engineering and feature selection. The process is a different dataset consisting of features. Figure 3.2 shows the different decision boundaries a classifier can learn given data with two features.

In our specific case, we employ feature engineering which is used in Human Activity Recognition (HAR). Human Activity Recognition defines the process of automatically recognising activities performed by a human being. Examples of such activities are differentiating the type of transportation (walking, running, cycling, driving a car) [Cardoso et al., 2016], home behaviour analysis [Vepakomma et al., 2015] or detecting work tasks [Koskimaki et al., 2009]. A widespread approach in HAR is to transform the time-based dataset into an instance based dataset using a sliding window approach [Koskimaki et al., 2009; Krishnan and Cook, 2014; Ortiz Laguna et al., 2011]. The dataset matrix $D \in \mathbb{R}^{(n_d,m_d)}$ is transformed to a feature matrix $D' \in \mathbb{R}^{(n_w,m_f)}$ where n_w is the number of windows over the data and m_f the number of features. This transformation is done by taking a specific consecutive number of time steps/instances (the window size) and extracting features from this data. Then the window advances to the next set of instances (the step size or stride). Commonly the window does advance a smaller amount of time steps than the window size, so the windows overlap. Some publications state the percentage of overlapping instead of the stride number [Koskimaki et al., 2009]. An example is that in a matrix with 1000 timesteps and 6 channels a sliding window of size 200 timesteps is used to extract 55 features, the windows overlap 50%, so the stride is 100 timesteps. This transforms the matrix $D \in \mathbb{R}^{(1000,6)}$ to the feature matrix $D' \in \mathbb{R}^{(16,55)}$. This process is depicted in equation 3.2. We see that a sliding window approach often reduces the number of instances available for the algorithm drastically and at the same time increases the number of features. In HAR there exist a large number of possible features to extract which I explain in the next section.

With the sliding window also the vector $\vec{y} \in \mathbb{R}^{(n_d,1)}$ needs to be transformed. The same window size and stride is used to transform the vector into $\vec{y'} \in \mathbb{R}^{(n_w,1)}$ or $\vec{y'} \in \mathbb{R}^{(n_w,2)}$. In this transformation a window can have a mixture of values of the dependent variable y. One now needs to decide how to deal with this. In human activity recognition time steps often correspond to one specific class or the zero class (the class corresponding to an unknown activity), but there is not overlap of classes for one time steps. If the window is small enough this means the a window can only be of one class, the zero class or a mixture of the zero class. In case $\vec{y'} \in \mathbb{R}^{(n_w,1)}$ a strategy can be to always take the dominant class as the dependen variable. The case of $\vec{y'} \in \mathbb{R}^{(n_w,2)}$ is called dual labeling and allows a second aptional class which is also right. A strategy is to only allow one class if only one class is present in the window or both possible classes if the window captures a mixture of those classes.

$$D \in \mathbb{R}^{(n_d,m_d)} = \begin{pmatrix} x_{(0_d,0_d)} & x_{(0_d,1_d)} & \cdots & x_{(0_d,m_d)} \\ x_{(1_d,0_d)} & x_{(1_d,1_d)} & \cdots & x_{(1_d,m_d)} \\ x_{(2_d,0_d)} & x_{(2_d,1_d)} & \cdots & x_{(2_d,m_d)} \\ \cdots & \cdots & \cdots & \cdots \\ x_{(n_d,0_d)} & x_{(n_d,1_d)} & \cdots & x_{(n_d,m_d)} \end{pmatrix} \implies D' \in \mathbb{R}^{(n_w,m_f)} = \begin{pmatrix} x_{(0_w,0_f)} & x_{(0_w,1_f)} & \cdots & x_{(0_w,m_f)} \\ x_{(1_w,0_f)} & x_{(1_w,1_f)} & \cdots & x_{(1_w,m_f)} \\ x_{(2_w,0_f)} & x_{(2_w,1_f)} & \cdots & x_{(2_w,m_f)} \\ \cdots & \cdots & \cdots & \cdots \\ x_{(n_w,0_f)} & x_{(n_w,1_f)} & \cdots & x_{(n_w,m_f)} \end{pmatrix} \tag{3.2}$$

The feature extraction over a sliding window allowes us to tread each datapoint in the feature matrix D' as an indpendent sample to classify. With overlapping windows some of the data is seen by several windows. This is not seen as a problem for modelling (see f.e. Koskimaki et al. [2009]) but care should be taken that when splitting validation and test set no overlapping data is used. Another positive aspect is that by ignoring the exact values of the time series but only using features over a window a lot of the noise in the time series is automatically removed.

3.4. Feature Engineering - Digital Signal Processing

In machine learning, feature engineering is the heart of the process. Depending on the classification model used, the data must be in a certain form. Linear methods (see Bishop [2006] Chapter 4.3.4 or Crammer et al. [2006]; Xu [2011]; Zhang [2004].) need the data to be linearly separateable. The machine learning model can find these linear hyperplanes in the data, but it can not transform the data itself except for methods which employ kernels [Boser et al., 1992; Rasmussen and Williams, 2005; Vapnik and Lerner, 1963] to do part of the feature engineering. Clustering methods need data of classes to be within a certain distance or density. Some methods like linear discriminant analysis (LDA) and principal component analysis (PCA) have problems if channels of the data are linearly dependent on each other or if data is highly redundant.

For this, the features are extracted from the data. Additionally to that the signal often has to be transformed into alternative representations of the same signal (Fourier and wavelet transformations) or into a different signal which better represents an aspect of the signal (linear and gravitation force). These fundamental transformations are part of the process of digital signal processing and complement the later feature extraction.

3.4.1. Splitting Gravitation Force and Linear Force

It is possible to predict the pose of a hand just by knowing the direction of gravitation in space of the accelerometer. If the arm is hanging down the accelerometer would have an x value in the range of -5 to $-15 m/s^2$, a y value 0 to $-5 m/s^2$ and a z value 6 to $-1 m/s^2$ in case of a smartwatch sensor. Linear force, on the other hand, can be harnessed for a gesture where the movement mainly defines the gesture, like waving.

A raw accelerometer does not differentiate between gravitation and movement force, although some modern system also do the splitting on the sensor using some digital signal processing (DSP) electronics. Splitting this signal with one axis sensor alone is extremely hard. When using many sensors, this splitting if feasible by a process called sensor fusion. When this term is used within the context of signal processing, sensor fusion can be used to manipulate signals directly. Incorporating different sensor information in machine learning is also often called sensor fusion. We mainly use this term for digital signal processing. In sensor fusion, the different characteristics of one sensor are used to correct the other one. Splitting linear and gravitational force in an inertial measurement unit (IMU) is a common form of sensor fusion for digital signal processing. Some modern smartwatches also give you the signal already split.

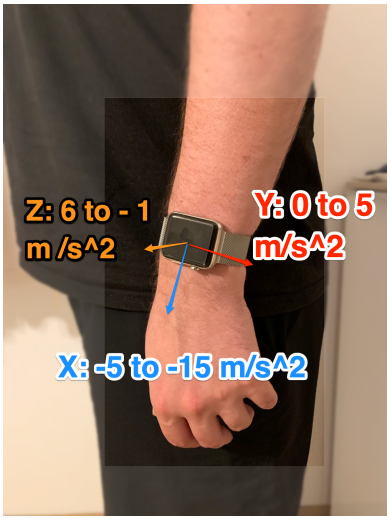


Figure 3.3.: Typical gravitational forces on a hanging arm with a smartwatch

In most cases however, you need to do the splitting yourself. The best accuracy usually is achieved by an extended Kalman filter (see chapter 15 in Russell and Norvig [2010] or the original linear algorithm in Kalman [1960]). It needs to be an extended one since the movements are highly non-linear in case of gesture recognition. An easier way is to use a complementary filter. A complementary filter uses a combination of gyroscope and accelerometer information to compute the current orientation angle Higgins [1975]. From that, you can calculate the gravitation component of each sensor axis. An excellent basis for implementing such sensor fusions is the project FSensor[7]. The basic formula of a complementary filter is pretty easy and can be seen in equation 3.5

$$\phi_n = \alpha * (\phi_{n-1} + \vec{\omega} * \Delta t) + (1 - \alpha) * \vec{a} \tag{3.3}$$

Here ϕ is the angel or heading of the sensor. The $\vec{\omega}$ is the angular velocity of the gyroscope in deg$/s$, which gets us the angle by multiplying it with Δt as the time between two samples. A fraction of the accelerometers readings $\vec{a}$ corrects this angle. The $\vec{a}$ means the accelerometer data expressed as the direction cosine. The result of the complementary filter is the current orientation angle. To split the force, this angle must be projected back on the axes and multiplied by the gravitation unit g to get the gravitation force. The linear force is simply the difference between the raw accelerometer signal and the gravitation signal.

$$\begin{aligned}
\alpha = \cos a &= \frac{\mathbf{v} \cdot \mathbf{e}_x}{\|\mathbf{v}\|} = \frac{v_x}{\sqrt{v_x^2 + v_y^2 + v_z^2}} \\
\beta = \cos b &= \frac{\mathbf{v} \cdot \mathbf{e}_y}{\|\mathbf{v}\|} = \frac{v_y}{\sqrt{v_x^2 + v_y^2 + v_z^2}} \\
\gamma = \cos c &= \frac{\mathbf{v} \cdot \mathbf{e}_z}{\|\mathbf{v}\|} = \frac{v_z}{\sqrt{v_x^2 + v_y^2 + v_z^2}} \\
\vec{a} &= [\alpha, \beta, \gamma]
\end{aligned} \tag{3.4}$$

The complementary filter needs a hyperparameter α which defines how the accelerometer influences the angle. The gyroscope has low noise but drifts over time. The accelerometer has much noise but low drift.

[7] https://github.com/KalebKE/FSensor

By mixing the sensors, these effects are removed. A usual value of α reaching this effect is 0.98. A concrete implementation of the complementary filter might look like equation 3.5.

$$\phi_n = 0.98 * (\phi_{n-1} + \text{gyr Data} * \Delta t) + 0.02 * (\text{accData}) \tag{3.5}$$

3.4.2. Absolute Energy Signal

The absolute energy within is the sum of squares of the directed impulses of movement within the window. It is an orientation independent measure of the activity within the signal [Fortino et al., 2015]. If gestures are different already by their amount of movement, this transformation can yield features which can help to find that. The transformation is simple to use the raw accelerometer and transform each time-step by $x'_t = x_t^2 + y_t^2 + z_t^2$.

3.4.3. Fourier Transformation

A Fourier transformation decomposes a signal as a combination of several periodical signals/frequencies. In human activity recognition, this is used for several tasks like detecting assembly tasks [Koskimaki et al., 2009] or detecting daily live achievements [Leutheuser et al., 2013].

We use the implementation of numpy to calculate the Fourier transformation. Numpy uses Equation 3.6 based on Cooley and Tukey [1965] for the discrete Fourier transformation (DFT) as basis of its implementation.

$$A_k = \sum_{m=0}^{n-1} a_m \exp\left\{-2\pi i \frac{mk}{n}\right\} \quad k = 0, \ldots, n-1 \tag{3.6}$$

3.4.4. Wavelet Transformation

The wavelet transformation (WT) has similarities to the Fourier transformation. It transforms a signal to a time-frequency domain. Compared to the Fourier transformation it can handle non-stationary characteristics of a signal like an impulse. The WT can transform the signal to a time-frequency domain, conserve the signal energy in the transformed domain, compress the signal and with it de-noise it. It is reversible and can handle non-stationary signals with periodic/quasi-periodic impulse trains. The WT exists as a continuous wavelet transform (CWT) shown in equation 3.7 and a discrete wavelet transform (DWT) shown in equation 3.8. The DWT is a transformation in which a subset of wavelet basis functions form an orthogonal basis and can be seen as a sampling of the CWT. After the transformation, like in the Fourier transformation, a typical feature engineering/feature selection step is to keep the most significant coefficients for the transformation. The wavelet transformation is a good feature transformation for accelerometer data [Suh et al., 1999].

$$CWT(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \psi^*(\frac{t-b}{a}) dt \tag{3.7}$$

$$DWT(j,k) = \frac{1}{\sqrt{2^j}} \int_{-\infty}^{\infty} x(t) \psi^*(2^{-j}t - kb_0) dt \tag{3.8}$$

3.5. Feature Engineering - Extracting Features from Windows

There exists a multitude of possible feature transformations for working with physical motion sensors in the human activity recognition literature. A window slides over the data performing the extraction. Before this sliding of the window, an algorithm splits the accelerometer signal into linear and gravitation force. It comes either as a preprocessing step over the whole signal (i.e. in the smartglove) or directly from the hardware (i.e. the smartwatch). For the smartglove, before the sliding window is applied, the absolute energy signal is computed. The frequency representation (Fourier transform) and time-frequency representation

(wavelet transform) are computed per window.

There exist several representations of the signal channels in a window. From all this data specific data points are now extracted for the later modelling. These points are used to describe the physical activity in a way a machine learning model can understand. In all following equations, the w_{start} and w_{end} stands for a current start and the end row of the sliding window over the data, and each extraction is evaluated over all possible sliding windows.

3.5.1. Statistical Features over the Window

In approaches using a sliding window it is very common to take statistical features from the data and computed features within the window. For acceleration this is f.e. done by Abdic et al. [2016]; Fortino et al. [2015]; Koskimaki et al. [2009]. Usually at least the minimal value within a window, the maximal value, the range or amplitute (maxminal - minimal), the mean, the median, standart error and the variance are present. This work uses the formulae in equation 3.9.

$$\begin{aligned} range(x, w_{\text{start}}, w_{\text{end}}) &= \max(x, w_{\text{start}}, w_{\text{end}}) - \min(x, w_{\text{start}}, w_{\text{end}}) \\ mean(x, w_{\text{start}}, w_{\text{end}}) &= \frac{\sum_{i=w_{\text{start}}}^{w_{\text{end}}} x_i}{w_{\text{end}} - w_{\text{start}}} \\ std(x, w_{\text{start}}, w_{\text{end}}) &= \sqrt{\frac{\sum_{i=w_{\text{start}}}^{w_{\text{end}}} (x_i - mean(x, w_{\text{start}}, w_{\text{end}}))^2}{w_{\text{end}} - w_{\text{start}}}} \end{aligned} \tag{3.9}$$

Beside these standart statistical features, some research reports using extended statistical measures like the skewness and kurtosis (i.e. in gait detection from Camps et al. [2018]). We include these measures computed over each window (see equation 3.10). All of these statistics make not only sense in the time domain, but are also useful features in the frequency domain (see Fourier transformation 3.5.5), often taken over the amplitudes of f.e. the acceleration values (see f.e. Samà et al. [2018]).

$$\begin{aligned} \mu = mean(x, w_{\text{start}}, w_{\text{end}}), &\sigma = std(x, w_{\text{start}}, w_{\text{end}}), \\ N &= w_{\text{end}} - w_{\text{start}} \\ skew(x, w_{\text{start}}, w_{\text{end}}) &= \frac{1}{N} \sum_{i=w_{\text{start}}}^{w_{\text{end}}} (\frac{x_i - \mu}{\sigma})^3 \\ kurtosis(x, w_{\text{start}}, w_{\text{end}}) &= \frac{1}{N} * \sum_{i=w_{\text{start}}}^{w_{\text{end}}} (\frac{x_i - \mu}{\sigma})^4 \end{aligned} \tag{3.10}$$

3.5.2. Zero Crossing

The number of zero crossing in an accelerometer or gyroscope signal is the number of times the signal changes its sign per sensor. Fortino et al. [2015] uses this to detect handshakes.

$$\begin{aligned} zero\_crossing(x, w_{\text{start}}, w_{\text{end}}) &= \frac{\sum_{i=w_{\text{start}}}^{w_{\text{end}}} |\text{sign}(x_i) - \text{sign}(x_{i-1})|}{2} \\ \text{sign}(x) &= \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \leq 0 \end{cases} \end{aligned} \tag{3.11}$$

3.5.3. Peaks

Detecting the peaks was used as a feature for detecting different work tasks in Ward et al. [2006]. F.e. manual screwing gives fewer peaks than hammering or holding an electric drill. There are many ways to do a peak detection. The most common method is to compare if the current point is higher than a point in the future and in the past ($p = x_t > x_{t-1} \wedge x_t > x_{t+1}$). This method assumes a clear signal or a good smoothing. I use the implementation of scipy's peak detection with a *CWT* based on the algorithm from Du et al. [2006].

3.5.4. Total Energy

The total energy is the sum of the absolute acceleration force within a window and acts as an index of the activity within the signal [Fortino et al., 2015]. If gestures are different already by their amount of movement, this transformation can yield features which can help to find that.

$$energy(x, w_{\text{start}}, w_{\text{end}}) = \sum_{i=w_{\text{start}}}^{w_{\text{end}}} x_i^2 + y_i^2 + z_i^2 \tag{3.12}$$

3.5.5. Fourier Transformation Based Features

After the channels of a window are transformed into their corresponding frequency representations, features are extracted from them. Like with any other representation of the channels it makes sense to extract all the statistical features from it. The sum of the smaller frequencies is used for recognition of assembly tasks in Koskimaki et al. [2009]. Taking a set of n amplitutes is also a common procedure. Typically, either the n first or the n largest (by amplitude) coefficients are used [Mörchen, 2003]. Another important feature is the spectral centroid. The spectral centroid ($\mathbb{C}$) denotes the centre of the frequencies and is computed by taking the (weighted) mean of a Fourier transformation. Another common derived feature of the Fourier transformation is the bandwidth. The bandwidth ($\mathbb{B}$) is the difference between the higher bound and lower bound of frequencies, expressed in Hertz. Spectral centroid and bandwidth are used in Leutheuser et al. [2013] to classify daily life activities. The (Power) Spectral Entropy ($\mathbb{H}$) denotes the amount of entropy in the frequency spectrum. It is used in Bao and Intille [2004] to classify several daily live activities. The equation 3.13 end equations in 3.14 are used to extract those features.

$$\mathbb{C} = \frac{\sum_{n=0}^{N-1} f(n)a(n)}{\sum_{n=0}^{N-1} a(n)} \quad a(n) = \text{amplitute at pin n} \tag{3.13}$$

$$\begin{aligned} S(m) &= |X(m)|^2 \\ P(m) &= \frac{S(m)}{\sum_i S(i)} \\ \mathbb{H} &= -\sum_{m=1}^{N} P(m)\log_2 P(m) \\ \mathbb{B} &= \max(\lambda) - \min(\lambda) \end{aligned} \tag{3.14}$$

3.5.6. Pairwise Features

A common praxis it not only to calculate features per source channel but already in the feature engineering compute features which represent interactions in the channel. The total energy is already a specific example of the interactions between all three accelerometer axis channels. Besides these specific ones there exist more generic approaches by computing features of pairs of channels.

The angle $\angle$ between two signals is often used as a measure of the similarity of those signals. An angle of 0 means both signals are the same, and an angle of Π means the two signals are as different as possible. We also use the angle in the frequency domain. The angle between consecutive windows has been used in Bieber et al. [2014]. Another common feature is the correlation between two channels. I use the Spearman rank order correlation coefficient ρ from Zwillinger and Kokoska [2000] for the correlation between two channels. Also computing the difference in the means d of the series is a possible feature [Suh et al., 1999].

$$\begin{aligned} \angle(\vec{v}_1, \vec{v}_2) &= arccos(\vec{v}_1 \bullet \vec{v}_2) \\ \rho &= \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}, \quad x_i \in \vec{v}_1, y_i \in \vec{v}_2 \\ d &= \bar{x} - \bar{y}, \qquad\qquad x_i \in \vec{v}_1, y_i \in \vec{v}_2 \end{aligned} \tag{3.15}$$

3.6. Feature Selection

The basis of a data-driven approach to gesture recognition is not only to find patterns in data but also to select the right features for this. In the data-driven approach, algorithms help in finding the correct features in a huge search space. This approach is called feature selection [Guyon and Elisseeff, 2003]. Too many features are not only slow, very often the accuracy of the systems are even slower, as the algorithm is more likely to find spurious correlations or overfit to the dataset. This is also called the curse of dimensionality.

There exist various techniques for feature selection and it has been applied in various domains. There are two classes for feature selection: Filters compute metrics directly from the features and sometimes the dependent variable. An example for this is removing features which are constant or have a low variance [Van Hulse et al., 2012]. Another example is interpreting each feature dimension as a random variable (often entropy is approximated for this feature [Kozachenko and Leonenko, 1987; Kraskov et al., 2004]) and then compute the mutual information for f-ANOVA to the dependent variable y to each feature as a selection criterium [Van Hulse et al., 2012].

Compared to filters, wrappers are a more powerful model-based method. Instead of using simple metrics a wrapper trains a machine learning model and uses aspects of that model to perform feature selection. A support vector machine (SVM) can rank features by calculating a distance to its hyperplane [Chang and Lin, 2008], Lasso and Elastic Net regularization compute a weight for each feature [Baraniuk, 2007]. If this weight goes to zero, the feature is removed. Trained only once these methods often just remove a small amount of features or hurt performance. An often used approach is to iteratively train a model and then use the ranking of the model to remove the lowest x percent of features. The iteration is stopped according to a criterium like a threshold on change in accuracy. This process is called recursive feature elimination (RFE) [Guyon and Elisseeff, 2003; Guyon et al., 2002]. Another approach is always evaluating which one feature to add best to the current best configuration, and then use this configuration as the new basis for the search. This is called best first search for feature selection [Xu et al., 1988].

Recursive feature elimination together with a support vector machine (RFE+SVM) is used in Yan and Zhang [2015] to find a good set of features for breath analysis. They introduce a correlation bias reduction method (CBR) to further improve the results. Park et al. [2018] uses RFE+SVM to investigate features for monitoring long term or short term stress with heart rate variability (HRV) measures using a electrocardiogram (ECG) sensor. This improves the accuracy of detection to 93.11% and allows to discuss the influence of HRV features on the detection. Mustapha et al. [2018] uses a multiclass SVM together with RFE to detect 9 different cracks in steel or healthy steel with an accuracy of over 95% using a network of piezoelectric (PZT) wafers.

3.7. Machine Learning Models

The next step after feature engineering and selection is creating a model using machine learning. Machine learning means a system which improves with experience E over some task T with a performance measure P [Mitchell, 1997]. A recommended introduction is the visual guide to machine learning[8]. In gesture recognition with a sliding window, this experience E is the features of a window and the assigned class. The task T is to learn a mapping from the features to the categorical variable y. Learning the mapping to dependent categorical variable is also called classification. The performance measure P depends on the way the mapping is learned. An example is the cost function $J(\omega)$ from logistic regression $J(\omega) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\omega\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\omega\left(x^{(i)}\right)\right)\right]$. For the person interpreting the algorithm usually additional performance metrics are displayed like the accuracy of the system in per cent.

The models considered for gesture recognition are models for classification. The model learns a mapping from features which can be of different data types (categorical, discrete, continuous) to a categorical variable. So instead of learning a function which best fits the data (regression) it must learn decision boundaries which best separate the data into their classes (classification). Most classification algorithms are, in their basic form, only suited to separate between two classes. This work wants to separate 31 gesture and the zero class. To do that there are two options:

Learning n classifiers of the same type all learning to map to one specific class vs all the other classes. At prediction time the classifier with the highest confidence is chosen to perform the mapping. This strategy is called one vs rest.

The other strategy is to learn $\frac{n(n-1)}{2}$ classifiers where each learns to separate two classes, and all combinations are covered. At prediction time a majority vote is performed among the classifiers. This method is called one vs one [Aly, 2005; Bishop, 2006]. In the case of classifying 31 gestures and the zero class, the one vs one strategy means that you need to learn 465 instead of 31 classifiers, which is much slower. However, this is still the chosen approach. The problem in the one vs rest classification is that the rest class is dramatically higher than the one class to predict, which is a hard problem for the classifiers, resulting in bad performance.

The performance metric for a model depends on how the model learns the mapping. For the person interpreting the model there exists a standard set of metrics to interpret the result of the model. These metrics are standard metrics in statistics. The most common metric is the metric of the overall accuracy of the system defined by $\frac{\text{correct prediction}}{\text{all predictions}}$. In most cases of activity recognition and for gesture recognition this is a poor metric. There are two reasons: If all the classes are balanced, but there are many of them, like 31, the algorithm can have a high accuracy for perfectly detecting 25 of them and ignoring 6 of them which would lead to 80% accuracy. Even more problematic is that in activity and gesture recognition there exists a big imbalance between classes. Whenever the user is not performing an activity, the class is recorded as the zero class. This class is dominant and appears a magnitute more often in these datasets than the other classes. If the algorithms conservatively detect everything as zero class and sometimes detect some gestures right, it yields an accuracy of $80-90\%$ of a still unusable system.

There exist several metrics which address this problem. A true positive is a correctly predicited class [Rijsbergen, 1979]. A true negative is the correct prediction that an example is an example of the other class. By sepearating the accuracy into precision, which is the accuracy to be right if a decision is made ($\frac{\text{true positives}}{\text{all predictions}}$) and the ability to find all instances of the classes ($\frac{\text{true positives}}{\text{everything classified for that specific class}}$) one can compute the f1 $F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$ measure which is a statistical accuracy measure which accounts better to class imbalances derived from Rijsbergen [1979] effectiveness measure. Another option is the receiver operating characteristic (ROC) curve, which plots true positives on the y axis and true negatives on the x axis for binary classification. For multi class, the individual classifiers must be binarised. By changing the confi-

[8] http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

dence threshold of binary classification, the algorithm can be biased towards finding more true positives or finding in total more of the example of the class by making more errors. The area under the curve (AUC) of the ROC curve is often used as a performance measure. For a multiclass problem, these AUC curves of each classifier are avaraged. Macro avaraging gives equal weight to each class, so it is important to use micro avaraging which incoporates the class weights for activity and gesture recognition problems.

Machine learning models learn a set of parameters to perform the mapping. In linear models, they learn a set of weights ω_i to find a linear combination of data siutable for task T. There exists another set of parameters which change the behaviour of the algorithm but are not learned by the algorithm itself. To differentiate from the parameters of the model they are called hyperparameters. Hyperparameters are the maximal depth of a decision tree, the type of kernel or other parameters I explain in the following section. Hyperparameters most often influence how complex a model can be, often also called the capacity of the model, and which basic assumption the model has. Therefore the hyperparameter tuning has a huge influence on the performance of the learned models. A support vector machine without hyperparameter tuning taken from scikit-learn [Pedregosa et al., 2011] performs very poorly on the smartwatch dataset. With tuning the contamination parameter C and the kernel, it gives almost perfect accuracy on the validation set (99%).

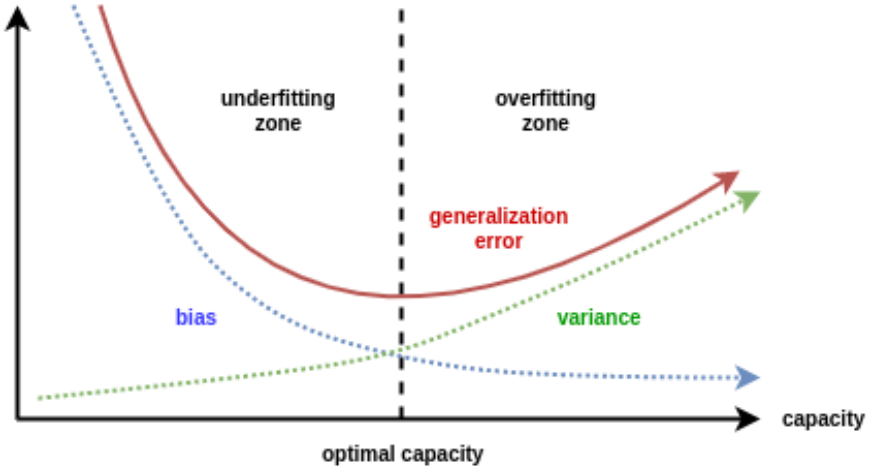
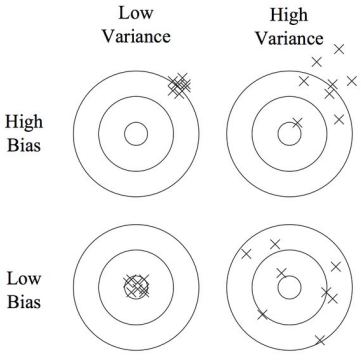


Figure 3.4.: Variance and bias as a function of model capacity/model complexity

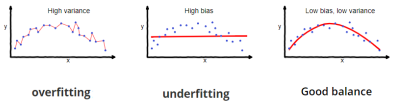
The complexity of a model defines its ability to fit data. In case it can not fit data there are two main causes for that. Either the model takes some things too much for granted. F.e. it can just always choose one class of two classes because this one class is overrepresented. This problem is called bias. The other option is that the algorithm is complex enough to learn data by heart. It can isolate single datapoints and say what they are. Complex models often lead to 100% accuracy on the training data but terrible results on unseen data. The problem is called overfitting, and for the model, we say it has too much variance. Another good way to explain the bias vs variance problem is dart throwing (from Domingos [2012]).

Most of the hyperparameters allow you to tune the so called bias / variance tradeoff. Common options are to not train the model too long (early stopping), making the model simple, penalising the parameters of the model to not become too complex by regularisation or using more data [Chokkanathan and Koteeswaran, 2018; Domingos, 2012].

There exist very many models even for classification alone. In his work "Do we Need Hundreds of Classifiers to Solve Real World Classification Problems" Fernández-Delgado et al. [2014] compares 179 different classification algorithms over 121 datasets. The paper concludes that the Random Forrest (RF) is the best one yielding supperior results in almost all cases. In competitions with machine learning however[9] most of



(a) Bias and variance in dart throwing from [Domingos, 2012]



(b) Good vs. bad fit

the time other algorithms win. The reason is that the paper only uses the default hyperparameters for modelling, showing again the importance of hyperparameters. In this work hyperparameter tunig is especially explored in the smartwatch case. Another insight is that there are too many classifiers to evaluate all in this work. This work concentrates on the classifiers available in scikit-learn [Pedregosa et al., 2011] which implements all the common classifiers.

3.7.1. Linear Models

Linear Models are models who learn a linear combination of the inputs to predict a dependent variable in the form of $\hat{y}(w,x) = w_0 + w_1x_1 + \ldots + w_nx_n$. This is the form for regression. The learning is done by minimising a cost function. For classification in a two class case the learned line must separate classes. The combination is changed by adding a sigmoid function and mapping to the categorical variable y binarised to either 0 or 1. The linear combination is changed to the hypothesis function $h_\omega(x) = \frac{1}{1+e^{w_0+w_1x_1+\ldots+w_nx_n}}$. The cost function changes then to $J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$ [Bishop, 2006]. This classifier is called logistic regression (LR). If the model is not learned by looking at all the data at once, but always by a set of sampled batches of data, the algorithm is called stochastic gradient decent classifier (SGD classifier) [Xu, 2011; Zhang, 2004]. Another linear model is the passive aggressive model, which uses closed form solutions. They are optimised for large datasets [Crammer et al., 2006].

If the values of the weights ω_i become high, the model can form very complex hyperplanes. The solution to this is to add a regularisation term to the basic logistic regression (LR) classifier. Regularisation of the parameters is adding the parameters themselves to the cost function. Since the target of the optimisation is getting the cost function to a minimum value, it now needs to balance the accuracy of the mapping with the values of the parameters, thus penalising the model for complexity. Several options exist for this penalising [Friedman, 1989; Friedman et al., 2010; Fu, 1998; Rifkin and Lippert, 2007; Zou and Hastie, 2005]. In the Ridge case, the sum of squares of the weights is added $+\alpha\|w\|_2^2$. This term is also called the ℓ_2 regularisation term [Rifkin and Lippert, 2007]. In the Lasso case, the absolute value of the weights is added $+\alpha\|w\|^1$. This term is also called the ℓ_1 regularisa-

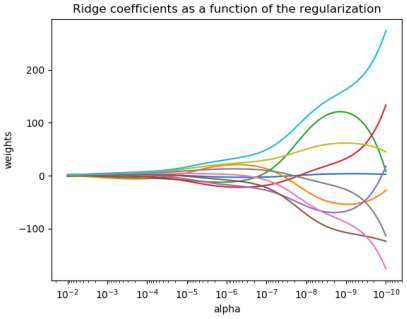
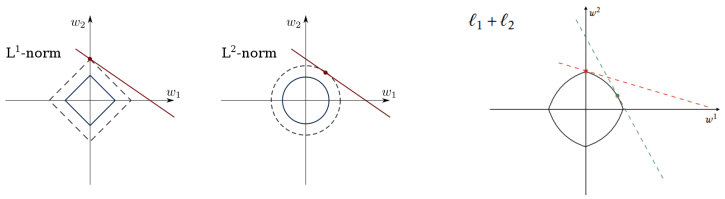


Figure 3.6.: Effect of l2/Ridge on the weights. Image taken from the scikit-learn HP.

[9] https://www.kaggle.org

tion term [Bishop, 2006; Friedman et al., 2010; Fu, 1998; jean
Kim et al., 2007]. The elastic net adds a weighting between those two options $+\alpha\rho\|w\|_1 + \frac{\alpha(1-\rho)}{2}\|w\|_2^2$. In all cases, the hyperparameter α determines how complex the model can become and is an important parameter to tune. In the case of the elastic net, the hyperparameter ρ determines which type of regularisation to use. For this work, which makes a data driven approach to feature engineering, the Lasso and Elastic Net regulisers are of special interest as the ℓ_1 regularisation tends to set specific weights ω to zero if there is a good setting of the hyperparameters α. Setting a weight to zero is the same as removing that feature from the dataset and thus can be used for feature selection [Zou and Hastie, 2005]. In this work, the effect of ℓ_1 regularisation is used for the feature selection algorithm.



(a) Constrain Areas of Ridge and Lasso

(b) Constrain Areas of Elastic Net

Figure 3.7.: Comparison of the Area the Regularisation wants to push the weights to. Image taken from Wikimedia Commons

logistic regression, together with decision trees, have been applied to smartwatch gesture recognition in the demo paper of Zhao et al. [2015].

3.7.2. Support Vector Machines

SVMs [Boser et al., 1992; Vapnik and Lerner, 1963] are classifiers which behave similar to linear or logistic regression. In contrast to logistic regression it optimises a different cost function which forces the linear decision boundary to have a maximal margin between the two classes instead of any linear decision boundary which satisfies the criterium of splitting the data. It does so by choosing points which define a vector on the outside boundary of the data distributions (so-called support vectors, hence the name) and puts the decision boundary in a place in the middle of those two vectors of each class. With this approach alone the SVM would not converge if the data is not completely separateable similar to the famous perceptron algorithm. However a hyperparameter C controls the amount of contamination between classes which is an important parameter for overfitting.

There exists a set of variants of a SVM. The general iterative algorithm to find the decision boundary stays the same. The difference is how the data is treated. In case of a linear SVM, data is just treated as a point in feature space similar to LR. Another option is to use kernels [Bishop, 2006; Boser et al., 1992], transformations which sit on the features, to transform the space even more. A kernel is basically a function which exists for each data point, and computes f.e. the distance to each other datapoint. Since it is a function, this mapping can be arbitrarily complex. The Radial Basis Function (RBF) is the best known kernel, and uses a Gaussian distribution to transform the data at each data point. Depending on the data function which is used, a SVM is either called linear SVM or SVM(RBF) or any other kernel. Each kernel exposes its own set of hyperparameters.

I use the scikit-learn provided implementations of the SVM algorithm. They are the C-Support Vector Classification (SVC), Nu-Support Vector Classification (NuSVC) and LinearSVC algorithms [Pedregosa et al., 2011]. SVC and NuSVC are general SVM algorithms, where you can use different kernels. The LinearSVC is a version which is optimised for the linear kernel. A standard SVM algorithm like the SVC uses the C penalty parameter to decide the tradeoff between finding a perfectly separating hyperplane for the data and between maximising the margin while allowing some data points outside the decision area. A value of $C = 0$ means that all data must be perfectly separated or the algorithm might not converge. Higher

values allow to find probably more generalisable hyperplanes on the cost of some misclassification on the training data. A SVM does classification by calculating the distance of a point to the hyperplane, and then basically looking at the sign of this distance. Points on one side have a positive, on the other side a negative sign. That also means that a SVM has no direct method for classifying multiple classes at once, as it is the case in our experiments. SVMs can deal with that in two ways: Either by building one classifier which detects if a point is a part of its class or of any other class (one vs. rest, or "ovr" in scikit learn) or by building a set of pairwise classifiers for each tuple combination of classes present (one vs. one or "ovo" in scikit learn).

SVM have been shown to work well in HAR and gesture recognition. In Dardas and Georganas [2011] SVMs are used on images to detect gestures. In HAR a good feature representation SVM-RBF has shown to give very good result in Parkinson's disease walk detection (see Camps et al. [2018]). Tuning the parameter C and choosing a good kernel (RBM) is a vital task to get a SVM to work on the smartwatch.

3.7.3. Naive Bayes

Naive bayes is a simple classifier which tries to predict the probability of a class from its feature $p(C_k|x_1,\dots,x_n)$. However, naive bayes assumes that there is no dependence between the features, and therefor can approximate the likelihood of each feature with $p(x_i|x_{i+1},\dots,x_n,C_k) \approx p(x_i|C_k)$ (see f.e. [Russell and Norvig, 2010]). It can then compute the class probability using equation 3.16.

$$\begin{aligned} p(C_k|x_1,\dots,x_n) &\propto p(C_k,x_1,\dots,x_n) \\ &\approx p(C_k)\,p(x_1|C_k)\,p(x_2|C_k)\,p(x_3|C_k)\cdots \\ &= p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \end{aligned} \tag{3.16}$$

In our case the data is continuous, so the parameter estimation $p(x=v|C_k)$ is computed by a Gaussion distribution (see equation 3.17).

$$p(x=v|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}} \tag{3.17}$$

Naive Bayes is used in smartwatch based gesture recognition for detecting 37 interaction based gestures (turning the arm, simulating a click, pinch to zoom, etc..) with an accuracy of 98% by Xu et al. [2015].

3.7.4. Gaussian Processes

Gaussian processes in machine learning (see Rasmussen and Williams [2005]) are a method to fit function data. There are infinite many functions possible to go trough a set of points in training data. A Gaussian process models this infinite possible amount of functions in form of kernels. It assumes that the parameters of these kernels are Gaussian distributed. It learns the Gaussian parameters from the data, and assigns inference by sampling from the distributions and giving a probability back.

3.7.5. Clustering and Distance-based Methods

There exist several methods which perform a clustering by measuring the distance of each point and finding the center of those points (Nearest Centroid classifier [Tibshirani et al., 2002]). The algorithm assumes one cluster per class. Classification is then done by assigning the point with the smallest distance. Another option is to save the whole labeled training dataset and look at a set of datapoints around the point to classify and take a majority voting. This model is the Nearest Neighbours [Coomans and Massart, 1982; Cover and Hart, 1967] model.

The Nearest Neighbours model is also called k-Nearest Neighbours (kNN) for its important hyperparameter k, which decides on how many neighbours are considered for the classification. If you choose too little

the algorithm might overfit to complex decision boundaries as outliers define an own region for its class. If you choose too many you might bias the algorithm towards major classes.

A very imporant hyperparameter for both, Nearest Centroid and Nearest Neighbour is the distance function. The default of this hyperparameter is to use the euclidean distance. For image based gesture recognition this is reported to work well [Birk et al., 1997]. However, if the clusters are skewed distributions this might lead to a very bad performance. In this case the machalanobis distance [Chandra et al., 1936; Mclachlan, 1999] is a better choise. Other options are taking the humming or edit distance.

In the smartglove case the dataset is very large. The kNN model needs the whole data to perform inference. With all features this algorithm performs too slow to be used in that case. It starts to work with datasets below 1000 features. The Nearest Centroid classifier works with every feature size.

3.7.6. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a classifier which learns a linear decision boundary by projecting data onto hyperplanes similar to PCA (see the comparison from Martinez and Kak [2001] or explanations in Bishop [2006]), but takes the class centers into account when doing the factoring. It was originally developed by Fisher [1936]. This can also be used for dimensionallity reduction. It is similar to ANOVA but ANOVA maps categorical independent variables to a continuous dependent variable while LDA maps continuous indipendent variables to a categorical dependent variable. LDA makes the assumption that the covariances of the individual classes are the same and full rank. A generalisation of the algorithm is the quadratic discriminant analysis (QDA) which does not make that assumption and can learn a quadratic decision boundary.

In gesture analysis LDA was used with a smartglove in Romaszewski et al. [2014] to detect 22 natural gestures from Glomb et al. [2012] by resampling and interpolating the data and then perform LDA on it. They achieve an accuracy of 92.8%.

3.7.7. Decision Tree

A decision tree is a classifier which uses information theoretic metrics (f.e. the information gain) to split the data at a certain threshold into two branches [Bishop, 2006; Hastie et al., 2009; Russell and Norvig, 2010]. It iteratively does this until the data is entirely separated, leading to pairwise linear decision boundaries. To avoid overfitting leaves which only have a few data points inside are removed and merged with their neighbours, a process called pruning. There exist different variants of decision trees using different metrics for the splitting, like C4.5, MARS and CART.

Decision trees, together with logistic regression, have been applied to smartwatch gesture recognition in the demo paper of Zhao et al. [2015].

3.7.8. Ensamble Methods

Ensamble methods are methods where many models are combined to predict the final outcome [Rokach, 2005; Russell and Norvig, 2010]. This combination is often a majority vote. This is based on Condorcet's Jury Theorem from Marquis de Condorcet, 1784. The theorem states that if an individual juror is more likely to be right than wrong, increasing the number of jurors can only help. More formally given N jurors with the avarage probability p of being right, the total probability μ is defined as $\mu = \sum_{i=m}^{N} \left(\frac{N!}{(N-i)!i!} \right) (p)^i (1-p)^{N-i}$. If $p > 0.5$ than $\mu > p$ and $\mu = 1$ if $p > 0.5$ and $N \to \inf$. This is used f.e. in bagging algorithms like a random forrest. Instead of voting, avaraging the models or doing a weighted avarage is used in many boosting algorithms.

3.7.9. Bagging

Bagging is a non-iterative ensamble method where many models are learned in parallel. Bagging stands for Boodstrapped AGGregatING. Instead of using one dataset in Bagging several datasets are created. Each dataset is of the same size as the original and created by sampling from the original dataset with replacement. A variant of that is to clone the original dataset several times instead of sampling, but weighting the features different in each clone, a process called wagging. Then the model is trained on each of the generated datasets [Breiman, 1996; Rokach, 2005].

The most famous model is the random forrest (RF). A random forrest is an extension to a decision tree. Instead of training one tree you train many trees each trained on its own dataset generated by bagging. In the version of random forrests only a subset of features are selected for each dataset for the sampling process to assure different trees. The prediction is done by majority voting [Breiman, 2001; Ho, 1995].

If there are very many features it is likely that many of them are highly correlated. In this case a random decision of features does not change the difference of each dataset a lot, resulting that the random forrests behave more like the same tree trained on the same data [Hastie et al., 2009].

Extremly randomised trees (Extra Trees) from Geurts et al. [2006] deal with this problem by adding splits in the tree which are completely random. They perform good on large and/or noisy datasets [Lawson et al., 2017]. The effect on a large amount of features is well observable in the smartglove case. With all the generated features the Extra Tree is the best classifier, it is also the best classifier globally. After a lot of features are removed by feature and sensor selection the random forrest is the best classfier given that data.

3.7.10. Boosting

Boosting is an iterative ensamble method [Bishop, 2006; Freund et al., 1999; Rokach, 2005; Schapire, 2003]. The idea is to iteratively first learn a set of several simple classifiers (also called weak learners in the boosting literature) and combine them to one decision. In each iteration, a simple classifier is learned to make an easy decision on the features.

AdaBoost [Freund and Schapire, 1997] is a classifier where a set of weak learners (you can choose which they are) solve easier problems than the original problem and then are combined to a final model. In each iteration, a learner learns to sparate some classes in an easy way. After that the data is re-weighted, with the wrongly classified data boosted to have a higher weight, so the new weak learner needs to emphasise a decision boundary which separates these. All these classifiers are then combined in the end according to weights proportional to the weight change of their reweighting step. AdaBoost is applied to camera based gesture recognition in [Chen et al., 2007] after transforming the image with Haar wavelets.

GradientBoost is like AdaBoost a variant of a boosting algorithm. That means that iteratively a set of weak classifiers are learned (with gradient boost often decision trees) and after learning the data is reweighted, so the next weak learner (f.e. tree) learns a different separation. The result is a weighted avarage of those weak classifiers. In Gradient Boosting the data is changed so the next weak learner $h(x)$ tries to classify the data in the error/residual of the first learner $h(x) = y - F_m(x)$ [Friedman, 2002].

3.8. System Perspective

So far I described the process of gesture recognition from the data perspective based on the KDDM process and human activity recognition. Another view is to view the system with its components. This work takes a wearable approach to gesture recognition in the form of a smartglove and in the form of a smartwatch. Body sensor networks (BSN) [Fortino et al., 2015] describe systems with sensors and some central processing which collects the data and performs feature extraction. The analysis layer uses models described above to perform predictions. In the dissemination layer, the predictions are used for interaction.

In the case of the smartglove there is a sensing layer and data collection on the glove. The data streams to a computer and the modelling is done offline. The demo system with the smartwatch describes a more complete BSN system. The sensors of the smartwatch are sent to a phone. The phone uses models which are created similarly as the gesture recognition models of the smartglove to perform predictions on the sensor stream in batches as they arrive. These predictions are then sent to a system to steer a slideshow.

The central elements not described in the process so far are the sensors used for capturing physical motion and the dissemination layer. The dissemination layer is explained in more detail in the smartwatch experiment. The sensors are described in this section now.

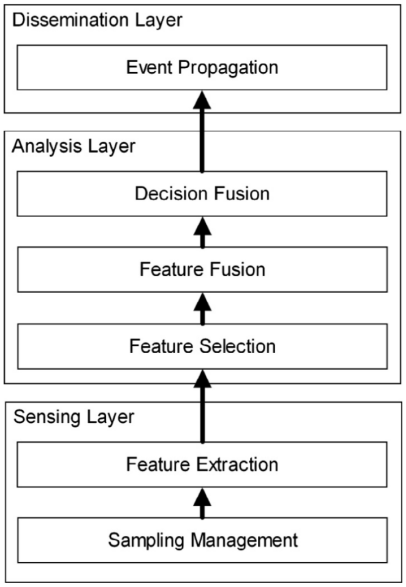


Figure 3.8.: Three-Layer Architecture for a BSN as described in Fortino et al. [2015]

A smartglove is a glove with control electronics and sensors. It is the sensors which allow the glove to recognise its environment. From a data perspective, the glove is a whole device capturing a (complex) movement and each sensor data stream is a channel (or several channels depending on the definition) of that movement. From these channels then features are generated. I now want to list the sensors used in this work.

3.8.1. Inertial Measurement Unit

IMU stands for inertial measurement [Woodman, 2007] unit and is a combination of several individual (inertial) sensors like accelerometers, gyroscopes or magnetometers. Depending on how many and in which axis these individual sensors are placed we speak of different degrees of freedom (DOF). An often used IMU has 6 DOF because it has three accelerometers, each in one axis (x,y,z) capturing linear acceleration and three gyroscopes, each in one axis (x,y,z) capturing rotational speed. The measurement is usually done with a low-cost Micro Electro Mechanical System (MEMS) [Jurman et al., 2007]. IMUs can be employed to create an inertial navigation system (INS) which navigates an object in a 3D space like in dead reckoning. The error of MEMS-based IMUs propagates over time, which means that you can be about 150m offset of where you calculated to be in $60s$ if you are unlucky [Woodman, 2007]. An interesting property is that the IMU measures the movement with different sensors. Therefore the errors of the individual sensors (noise and drift) can be reduced using sensor fusion with methods like a nonlinear Kalman filter, a Madgwick algorithm or a complementary filter[10]. The combined / sensor fused data of an IMU represents movement in a 3D space (depending on the usage of the magnetometer the coordinate system is fixed differently). There are several options to present movement in 3D space: Using the direction cosine matrix (DCM), Euler angles or quarternions [Jurman et al., 2007]. .

3.8.2. Accelerometer

An accelerometer is a sensor measuring proper acceleration, that is the gravitational force among axes in m/s^2. F.e. a resting accelerometer that is placed with the x-axes upwards will measure $1g$ upwards (approximately $9.81m/s^2$) on the x-axis and about 0 on the other axes. An accelerometer sensor measures on one axis only. If you want another axis you add another accelerometer placed accordingly, although this is often built into one electronics chip. Most consumer good devices contain accelerometers that are Micro Electro Mechanical System (MEMS). They are made of a movable plate that is fixed to a reference frame through a suspension system [Cai, 2014; Lyshevski, 2002]. A common problem for accelerometers

[10] http://www.starlino.com/imu_guide.html

is noise, especially when lying still. A moving average filter is a good first step to lower this noise (see Chapter 15 Moving Average Filters in [Smith, 1997]).

3.8.3. Gyroscope

A gyroscope measures how fast an object rotates in 3 coordinate planes, that is the XY, YZ, and the ZX plane. The measurements are usually in degree per second (deg/s) per plane. Many manufacturing variations of gyroscopes exist, although MEMS gyroscopes are the most used in consumer goods or hobby electronics, and use a micro-vibrating element as a pendulum. Gyroscopes are less prone to noise, but therefore have a problem with drift. This complementary nature of gyroscope and accelerometer is used in the complementary filter for IMUs [Woodman, 2007].

3.8.4. Magnetometer

A magnetometer measures the strength and direction of the surrounding magnetic field. The measure is done in degrees (deg) for each axis. Most smartphone users know it as a compass or sometimes as a metal detector. In smartphones and hobby electronics solid state technology is used to create miniature Hall-effect magnetometers, which produce a voltage proportional to the magnetic field [Woodman, 2007].

3.8.5. Flex Sensors

Flex sensors are also called flexion sensors or bend sensors. They measure the deflection caused by bending the sensor, in form or output voltage. If the sensor is fixed to one angle (like a finger joint in our case) you can approximate with the measurement the angle of the joint.

The usual ways to create flexion sensors are conductive ink based, fibre optic based, or conductive fabric based. In our experiment, we used conductive ink-based flexion sensors. They are relatively cost-effective compared to fibre optic, and clearly more reliable than conductive fabric based, and very popular in hobby electronics. They come with a start resistance, and on bending the resistance can increase up to 10x the start resistance [Dunne et al., 2007].

3.8.6. Force Sensitive Resistors

A Force Sensitive Resistor (FSR) is sometimes also called a pressure sensor. It actually changes its internal resistance based on the pressure applied to the sensor. On pressure, a conductive film comes into contact with an area with many conductors. The more pressure, the more contact, lowering the resistance. FSRs usually have a very high resting resistance (about $1M\Omega$), and only after some force starts with about $100k\Omega$ to a lower value. More detail on the workings of FSRs can be found in Burdea [1996].

Chapter 4

The Gesture Glove Experiment

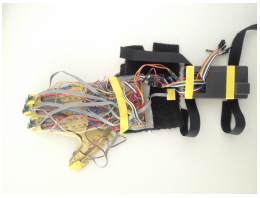
Parts of the contents of this chapter have been published in the proceedings of 3DUI'16 *[Luzhnica et al., 2016].*

The idea for the smartglove was to produce a system which allows recognising 31 natural gestures plus the zero class. We are interested in a general-purpose gesture alphabet with which to control computers and communicate with them. Essentially, it would be possible to develop a completely novel gesture language for such a purpose. A study looking at inventing custom gestures [Jego et al., 2013] showed however, that a user could only remember a very small number (about two) of such gestures. Therefore, we are looking at gestures that are widely known, even though there may be cultural differences regarding their popularity and meaning. Additionally, there should be a plausible relationship between the gesture and an interaction between human and computer.

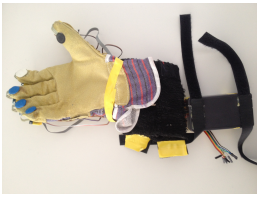
These criteria result in the following 31 hand gestures (see table 4.1 and figure 4.1). Our gesture set was initially based on the list of 22 natural gestures described in Glomb et al. [2012]. We added the following: The numbers one to five, as they would be useful to select items; popular touch-based swipe gestures such as swipe left, right, up and down, as these would be useful for navigation. Finally, we added lateral grasp (Grasp 2) and palmar grasp (Grasp 1) gestures, as we think that grasping objects would be useful in interaction with 3D virtual objects.

4.1. The Hardware of the Custom Smartglove and Data Collection Software

The mentioned gestures vary a lot in their dynamics: Some gestures contain a lot of complex motions (e.g continue) whereas some are very close to a posture (e.g. numbers one, two, ...).
The smartglove emphasises motion detection of the fingers (which implies that we would have motion sensors on the fingers); as well as hand postures (which implies that we would use bend sensors). The glove is depicted in Figure 4.2.



(a) Top View



(b) Bottom View

Figure 4.2.

We placed two flexion sensors on each finger. The upper sensor measures the bending (which translates to angle) of the finger relative to the hand, whereas the lower sensor measures the bending between the middle segment and the base segment of the finger. Another flexion sensor is placed between the thumb and index finger in order to measure the distance/angle between them. Two more flexion sensors (in opposite directions) are placed on the wrist in order to be able to measure wrist flexion/extension. Overall, this gives 13 flex-

Gesture	Description
(1) One	Number one by extending index finger
(2) Two	Number two by extending index and middle finger
(3) Three	Number three by extending index, middle and ring finger
(4) Four	Number four by extending all fingers except thumb
(5) Five	Number five by extending all fingers
Thumbs up	Thump stretched pointing up, other fingers form fist
Thumbs down	Thump stretched pointing down, other fingers form fist
Point to self	Pointing at self with thumb
Shoot	Hand in form of a gun and then vibrate
Scissor	Stimulating scissors with two fingers
Cutthroat	Using index finger
Continue	Waving like circular motion with the flat hand
Knocking	Forming a fist and moving the fist up and down
Waving	Shaking the flat hand left and right
Come here	Flat hand with palm upwards: Simultaneous flexing the all fingers but the thumb
Go away	Hand with palm downwards, all fingers but thumb flexed. Simultaneously stretching them
Push away	Flat hand with palm pointing forwards, then moving the whole hand forward
Never mind	Flat hand with palm pointing left above the head, then moving the whole hand left
Talking	Thumb and 4 fingers pointing forward. Then moving 4 fingers up and down
Calling	Hand is a fist, but thumb and small finger are extended
Walking	Hand is a fist, but making a walking motion with the index and middle finger
Shoulder pat	patting with the open hand on a virtual shoulder
Point	Pointing in front with index finger
Swipe left	Stretched hand with palm pointing left, flexing it completely to the right first, then flexing it to the left, in a circular motion
Swipe right	swiping with palm pointing right, and left to right motion
Swipe up	swiping with palm pointing up, and bottom to top motion
Down	swiping with palm pointing down, and top to bottom motion
Turn	Hand rotation
Zoom	Reverse pinch using index finger and thumb
Grasp 1	Palmar grasp (in the experiment we used a glass)
Grasp 2	Lateral grasp (in the experiment we used a pen)

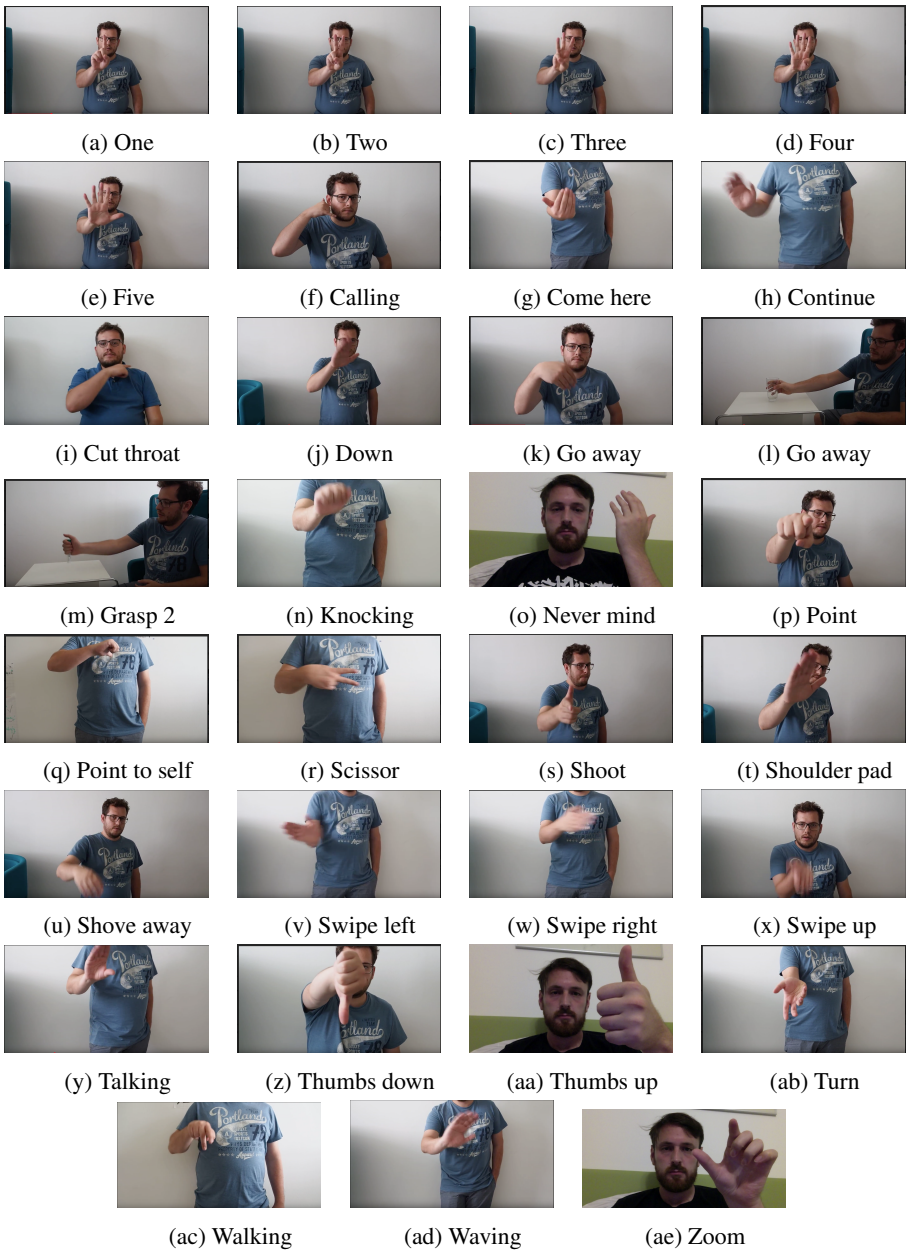
Table 4.1.: List of 31 interaction-oriented hand gestures.

ion sensors. Additionally, each fingertip is equipped with a pressure sensor
(5 pressure sensors). Furthermore, 7 IMUs with 6 *DOF* are placed, one at the top of each finger, one on the back of the hand and one on the wrist. The wrist IMU is placed exactly at the position where a watch would be. This placement allows the data recorded with the glove also to be treated as if it came from a smartwatch by simply ignoring the input from other sensors. Finally, a magnetometer is placed on the back of the hand.

This setup gives us a smartglove with 52 *DOF* which is more than other gloves had at the time of building the glove and more than needed for capturing the kinematics of a hand (see chapter 2 for reference).

From a modelling perspective, it makes sense to know the value ranges and noise of a sensor. Often models trained with one type of noise are not easily transferred to another. Also, it makes sense to have full documentation of the hardware setup of the smartglove. This documentation describes the technical aspects of the sensors, the central data collection unit in the form of an Arduino DUE and how sensors and the Arduino are connected.

Flexion sensors are probably the most used sensors for smartgloves (see Alapati and Yeole [2017]). Also all gloves described in this thesis use them. We use a standard conductive ink-based 2.2" (so 5.8cm) flexion sensor. These sensors are relatively cheap compared to the other options (fibre optic and conductive thread/polymer based). Also, they are less ergonomic, especially compared to conductive thread. They are



(a) One (b) Two (c) Three (d) Four

(e) Five (f) Calling (g) Come here (h) Continue

(i) Cut throat (j) Down (k) Go away (l) Go away

(m) Grasp 2 (n) Knocking (o) Never mind (p) Point

(q) Point to self (r) Scissor (s) Shoot (t) Shoulder pad

(u) Shove away (v) Swipe left (w) Swipe right (x) Swipe up

(y) Talking (z) Thumbs down (aa) Thumbs up (ab) Turn

(ac) Walking (ad) Waving (ae) Zoom

Figure 4.1.: 31 gestures used in this work. Additional explanation in the text.

longlived, and the accuracy suffices for the prototype. The internal pull-up resistor of an Arduino is enough for connection.

For the IMU we used the MPU-6050 chip with a GY-512 breakout board. This chip is a 6 *DOF* IMU with a digital motion processor (DMS) on one small chip. It operates at a voltage of 2.3 to 3.4V and sends its data over I^2C. The DMS is not sufficiently documented for hobby users. Some used reverse engineering to control it properly, others (as we did) just used the raw values of the sensors. The chips sensitivity for the sensors can be set by the developer, and ranges from $+/-2g$, $+/-4g$, $+/-8g$, or $+/-16g$ for the accelerometer and $+/-250\circ/s$, $+/-500\circ/s$, $+/-1000\circ/s$, or $+/-2000\circ/s$ for the gyroscope. We used the default setting which is the lowest of each ($+/-2g$, $+/-250\circ/s$). Some resources, like the

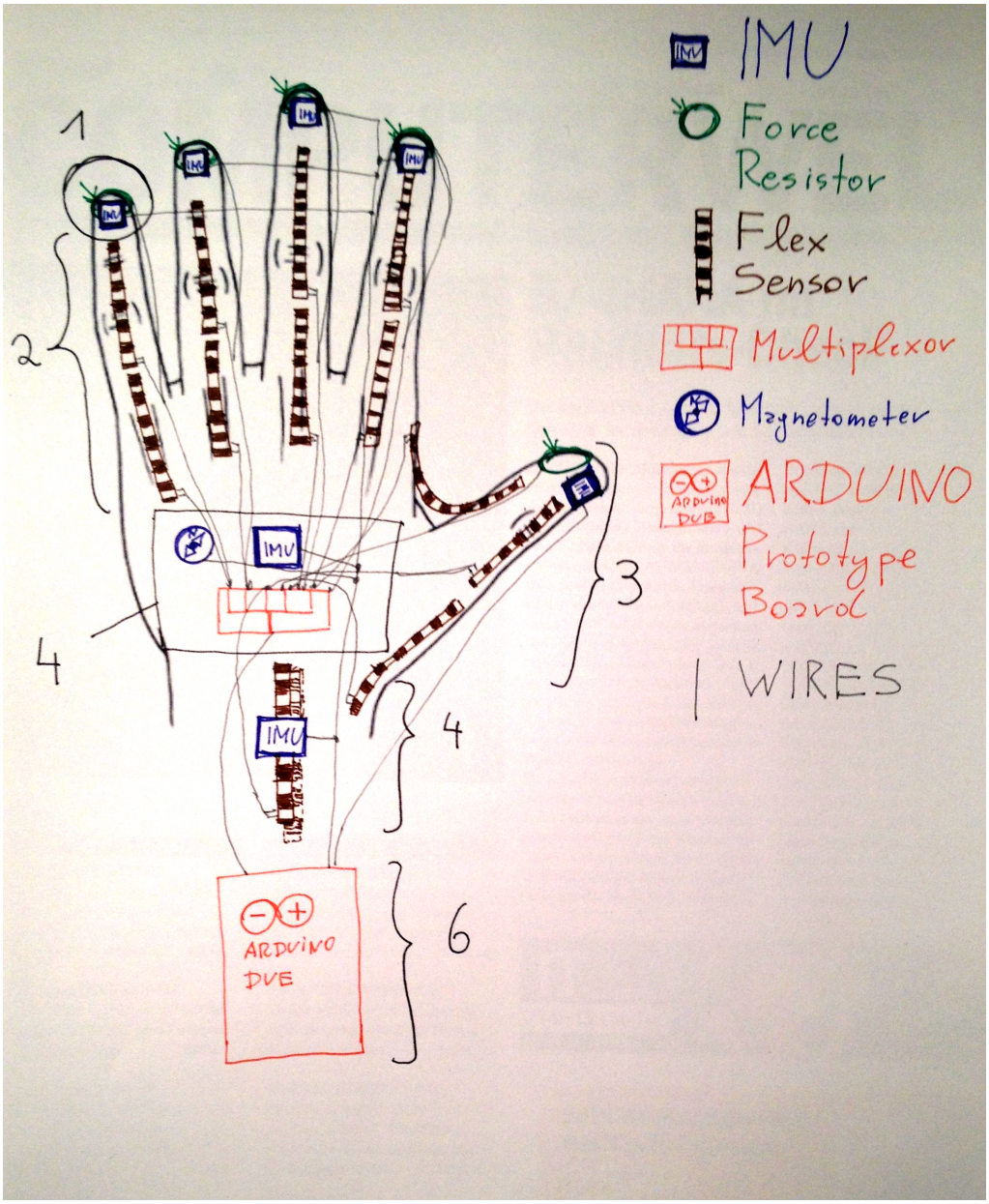


Figure 4.3.: A schematic view on the smartglove.
1. Each fingertip has an IMU on the top and a force resistor on the bottom. 2. Each finger has two flexion sensors to measure the flexion of two parts of the finger. 3. The thumb has three flexion sensors, the additional one measures the opposition of the thumb to the hand. 4. On the top of the palm, we placed an IMU, a magnetometer and an analogue multiplexer. 5. On the wrist, we placed a flexion sensor and additionally an IMU at a position where a smartwatch would usually be placed. 6. All the sensors are connected to an Arduino DUE board. The board is held by an armband with a velcro fastener
Christoffer Öjeling did the main assembly and C programming with input of Eduardo Veas, Granit Luzhinca and me.

manufacturers home page[1], the data sheet[2]or Arduino tutorials[3]exit.

We use the HMC5883L magnetometer. This magnetometer operates with $3.3V$. It communicates over the I^2C protocol. The measuring rate is $\pm 1.3 - 8Gs$ gauss.

The pressure sensor is the interlink force sensitive resistor FSR 402 Short. It has a diameter of $0.58mm$, and a sensitivity of $0.2N - 20N$ and is therefore optimised for human touch control. In our first experiments, the angle of the touch had a big influence. Therefore we put sugru glue over them so the force of touch is applied more evenly. The resistance range is well suited to work with the internal pull-up resistor of an Arduino DUE.

All sensors with an I^2C protocol are connected directly to the Arduino DUE. However, the Arduino DUE only has 12 analogue inputs, and we have 17 devices. Therefore we use a 16 channel analogue multiplexer from SparkFun (CD74HC4067). Five digital pins are connected to control the multiplexer which acts as a rotary switch for the analogue signals. The multiplexer connects to all the flexion sensors. An implication is that in each sending circle the Arduino has to sample the sensors for each transmission.

On the forearm, a plastic box is fastened with velcro. The box is the casing for an Arduino DUE. The Arduino DUE is used to control the electronics and sensors, and collect the data. The device operates on $3.3V$ compared to the $5V$ of popular AVR based boards like the Rasperry Pi. Care must be taken to provide the correct voltage as too much voltage damages the board. Connecting the Arduino DUE per USB is enough to power the board. The Arduino DUE is based on the Atmel SAM3X8E ARM Cortex-M3 CPU which has 32 bit. The board's clock is set to 84 Mhz. It was the only 32 bit ARM Arduino at the time we built the glove. The Arduino DUE has 54 digital IO pins, of which 12 can be used with PWM. There are 12 analogue inputs. The ADC and PWM resolution is supported up to 12 bit. The Arduino DUE also has some other electronics we do not use. The board has two USB connections. One goes directly to the CPU, the other, the programming USB goes to the ATMEL 16U2 which acts as a USB-to-Serial converter.

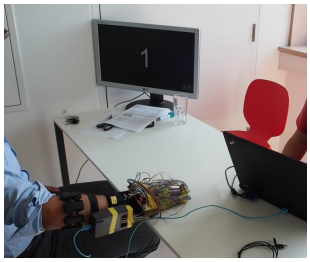
We used the serial connection protocol over USB (the same you use to program the Arduino) to collect the data. Every 0.012s one loop through the sensors was finished and sent back to the connected computer. A C program on the Arduino initialises the sensors on startup and in the loop instructs the multiplexer to switch between the flexion sensors and take the current value. Also, it reads all the pressure sensors over analogue ins and the data of all IMUs and the magnetometer over I^2C. One loop through all the sensors takes 12 milliseconds. Every 12 milliseconds ($83.\dot{3}Hz$) the data is sent over USB serial connection to a PC if connected. Since the frequency of human movements is low and mostly under $20Hz$ [Camps et al., 2018], this is more than sufficient.

Additionally, Christoffer built scripts to control the glove from the PC. The `calibrate_analog.py` python script calibrates the gyroscope of the glove and the *min*. and *max*. values of the flexion sensors. It is advised to call that always before an experiment and save the resulting offset.txt files. Two visualisation scripts allow seeing either the centre of the gyroscopes or a simple 3D joints visualisation of the hand. Finally, there is one basis script to test the data collection, and one script for the data collection we used in our experiment.

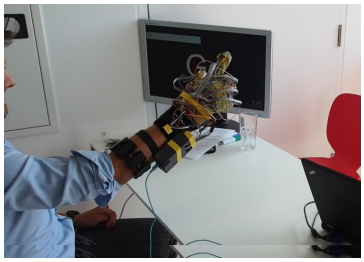
The finished version of the glove is shown in figure 4.2, a schematic view in figure 4.3. Our custom smartglove is a hardware prototype and as such it has some limitations, mainly regarding usability. For long-term wearing, the glove should for instance be made of more comfortable material, be made of smaller and not visible electronic components, should be available in different sizes, and be wirelessly connected to the computing unit.

[1] `http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/`
[2] `https://www.cdiweb.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf`
[3] `http://playground.arduino.cc/Main/MPU-6050`

4.2. Data Collection Experiment



(a) Countdown to perform a gesture



(b) Participant performs the "One (1)" gesture while the progressbar is on the screen

Figure 4.4.

We collected sensory data annotated with gesture names in the subsequently described data collection experiment. 27 people participated in the experiment. On some we had to interrupt in the middle because of hardware problems or timing of the participant or discarded the data for technical reasons: The accelerometer sensitivity was very low and therefore much clipping occured. Clipping means that, due to measurement errors, there were long times of maximum acceleration. For subsequent experiments, the acceleration sensitivity settings were changed to allow higher (but less precise) acceleration readings. We recorded the whole experiment with 24 people. After data cleaning, two more people had to be excluded from the study. In total, we collected data from 22 healthy adults: 12 males and 8 females. Participants were aged between 24 and 40 years. In the original paper of Luzhnica et al. [2016], we were more conservative in the exclusion, using 18 people for the data analysis.

The overall process for one participant is shown in figure 4.5. (1) Before starting with the data recording, the purpose of the experiment was explained. (2) For each participant, we collected demographic data and data about the hand. Participants were asked to remain seated during the experiment in an office chair. In front of them (on the desk), a monitor was placed. (3) We helped with putting on the data glove and connected the system to the computer. (4) We explained the process and let the user try out one set of repetitions of gestures without recording (5 and 6). Then the data collection started (7).

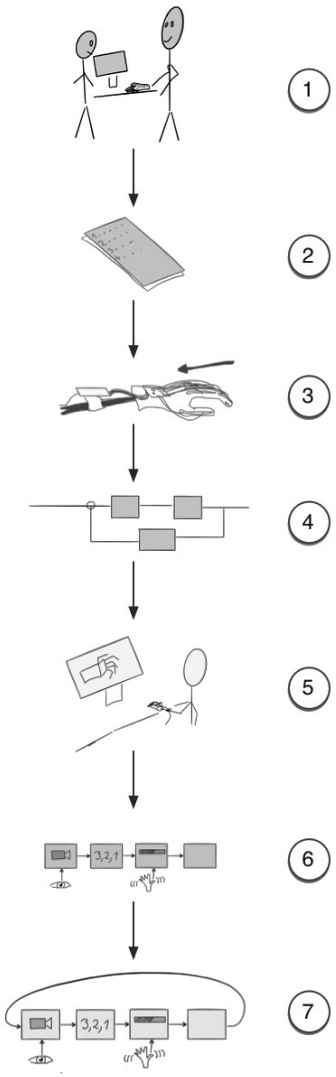


Figure 4.5.: The overall process of the study

A schematic of the data collection is shown in figure 4.6. Every gesture was performed several times by every participant (5 or 10 times depending on the willingness of the participant) in a row. The gesture name and the video of the actor performing the gesture (steps 1 and 2) were shown only for the first repetition of the gesture, whereas the counter, progress bar and labelling (steps 3-7) were the same in every repetition. Figure 4.4 shows one participant while performing the "One" gesture.
For each gesture, the following steps were performed in the given sequence (see figure 4.6):

1. The name of the gesture was shown on the screen for 2*s*, then a video was displayed; it showed an actor performing the gesture (without a glove; 6*s*-7*s*)

2. A counter was shown on the screen alarming the participants that the recording was about to start (3*s*).

3. The participant was asked (audio and text on screen) to start performing the gesture. A progress bar was displayed on the screen, indicating the time the participant had to finish the gesture (3*s*). The appearance of the progress bar started the time window called "automatic labelling".

4. When the participant started the gesture, the experiment observer pressed a button on the keyboard. This button press indicated the start point of the time window called "dynamic labelling". When the participant ended the gesture, the experiment observer released the button. This was the end point of the time window called "dynamic labelling". In case that the gesture contained a stationary state/posture, the experimenter noted the start time and end time of the stationary part as well (by pressing and holding another button until it was over)

5. When the progress bar ended, the time window called "automatic labelling" was closed.

6. Depending on the participant each gesture was repeated 5 or 6 times.

7. While we did not assume any specific learning effects we were worried about the effects of tiring of the arm during the experiments. It could be that, if a gesture is always the last after a 30min or 1h session, it is performed differently than at the start. Therefore the process is repeated for each of the 31 gestures in random order.

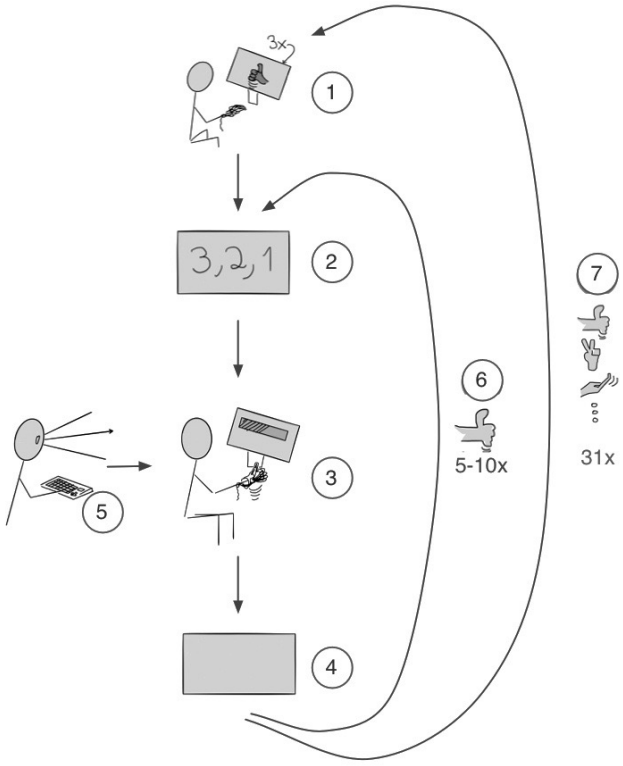


Figure 4.6.: Schematic of the data collection process.

4.3. Collected Data

During the experiment participants also did wear EMG sensors in the form of a Myo armband. In Luzhnica et al. [2016] and also in this thesis we do not cover this data. At data collection the glove records the data of each of its sensors. With the 52 *DOF* of direct movement, the 3 *DOF* of the magnetometer and the five pressure sensors the glove records 64 channels. The glove loops through the sensors every $12ms$ and reads the current values and sends this array to a computer, resulting in a sampling rate of $83,\bar{3}Hz$. This array is then written into a .csv file. In the experiment the data was saved in the following way automatically: For each participant, a folder with the participant code was created. In that folder a file in the format `CODE_TYPE_TIMESTAMP.csv` was created, with `CODE` being the participant code, `TYPE` being either gesture, myo or labels, timestamp being in the format `Year_Month_Day_Hour_Minute_Second`. An example would be `PS42_glove_2015_08_13_15_06_08.csv`.

The type glove means it has all the channels from the glove, myo has all the channels from the myo and labels have a start and end index for glove and myo for each time a label was recorded in a row. If you combine all the raw data (without the myo data) into one big python pickle we have around *3.4 GB* of raw data.

If an experiment was interrupted in between because there was a short break needed or a problem with a sensor, it could be picked up at a later time. In this case, several of the above files exist in the user's folder, with a different starting timestamp. A first step is to create one dataset suitable for exploration out of these user recording sessions and individual files.

4.3.1. Preprocessing - Labels

The raw .csv files from the data collection have no index in the time domain, and if the experiment is interrupted, the index starts at 0 for the new file. By multiplying the file index with a time delta of $0.012s$ and adding this to the starting time of recording, encoded in the file name, we recover the time domain of the labels.

The labels in the label file are start and end indices of rows in the data collection files. Columns for labels extend the data matrix. Then each row in the data matrix is annotated with the corresponding labels. We differentiate between four different types of labels. Depending on the way a label is created (automatically or manual) and if it annotated a dynamic or static part of the gesture, the label has a different type. The label types we have are the following:

- Automatic: the user is asked by the program of the experiment to perform the gesture after a countdown within three seconds. The start and the end index of this window is tagged automatically by the data collection software with the `label_automatic` type
- Manual: The manual label is the combination of the dynamic and the static label, and recorded as the `label_manual` type
- Dynamic: When the user performs a gesture an experimenter manually pressed the 1 key on the keyboard when the dynamic part of a gesture has taken place. The dynamic part is defined as the part where arm and/or fingers move. The start and the end index of this manual labelling is recorded in the `label_dynamic` type.
- Static: When the dynamic part ends, often a static part is added to a gesture. The static part is when the arm rests in a pose forming the symbolic shape the gesture represents (like thumbs up). If that part is present, the experimenter manually labels that part by pressing the 2 key on the keyboard. The start and end frame are recorded into the `label_static` type.

The label types for a structure are visible in figure 4.7. If the experimenter made an error or if any other reason lead a violation of this structure, we excluded the label from the data.

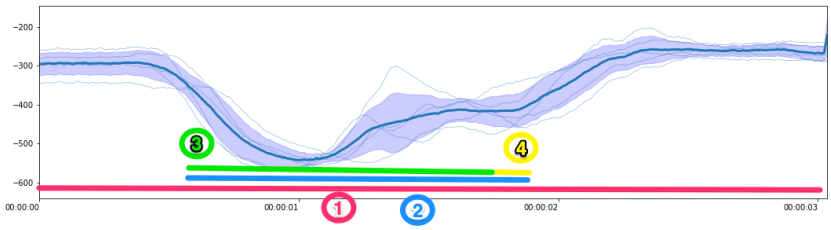


Figure 4.7.: An example of a `LabelGroup`. (1) is an automatic label (always 3s), (2) is the manual label shorter than the automatic label. A manual label consists of a dynamic (3) and a static (4) part. The static part is optional. The dynamic part is always before the static part and not optional.

We validate labels in a two-way process: We use the data already annotated with labels to find consecutive rows with label information. From these consecutive rows, the label's structure is recreated and each group of labels saved as a `LabelGroup` object. If that does not work, the label is discarded. 1.9616% of the labels in the dataset are thrown away this way. Additionally, the structure of the label is checked by recreating the structure directly from the labels file. This is also saved in a `LabelGroup` object. 1.1902% of the labels in the labels file do not meet that criterium, but they are mostly the same who are also removed the other way. At least the two lists of `LabelGroup` are compared if they annotate the same time sequences. If they do not agree the corresponding labels are thrown away. For most users, no correction is needed. For a few users up to 4% of the LabelGroups need to be removed. In total, another 0.9155% of LabelGroups need to be removed over the complete data.

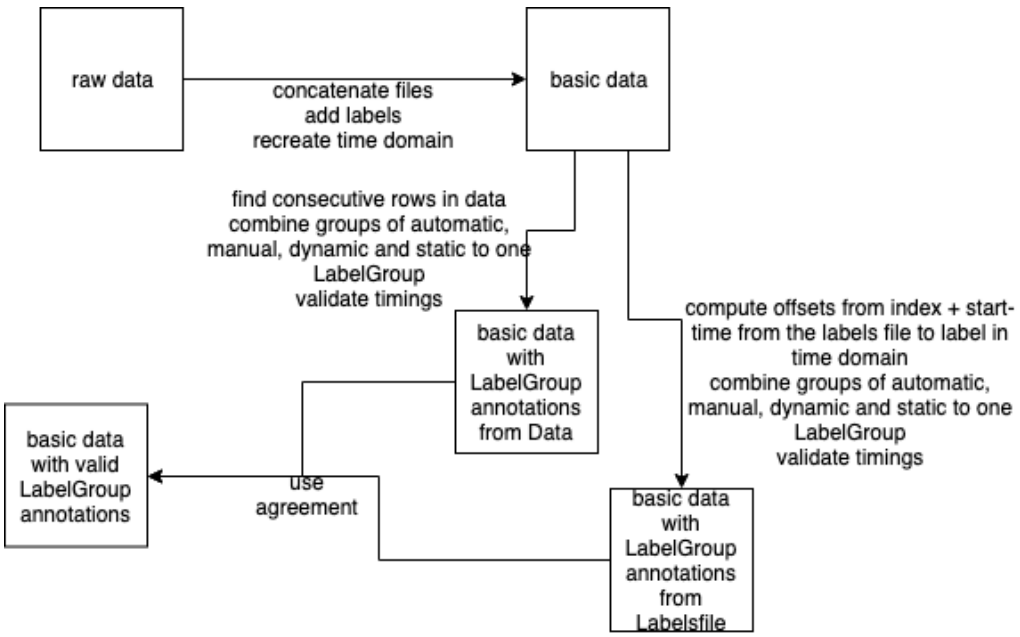


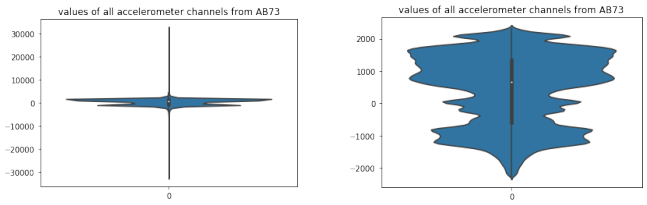
Figure 4.8.: Showing the process from raw data to preprocessed data with valid labels

4.3.2. Preprocessing - Outlier Removal

For later modelling, it makes sense to look at data quality. One possible way to look at that is to check if some values are outliers. We know from experiments that sometimes the sensors gave an error, which resulted in the return of a very large positive or negative number. For our dataset, we want to remove this

erroneous values. Additionally, inspecting the distribution of values for sensors and channels gives us an insight into how well a classifier can differentiate between classes, and also might hint at dangers of overfitting. F.e. if the distributions of values for a gesture do not overlap for a sensor, a simple thresholding algorithm can directly be used as a classifier.

Before visualising the sensors, we must check if they are normally distributed or not, to see if a boxplot visualisation makes sense or if we need something different. All sensors are not normally distributed. The test statistics for testing for normal distribution for the sensor values are reported in the Appendix, Table A.1. Because the value distribution is non-normal, scatterplots are used for the outliers visualisation and violin plots for visualising the distributions of the sensor values.



(a) with outliers (b) without outliers

Figure 4.9.: Valuerange of all accelerometer sensors used for user AB73 with (a) and without (b) outliers removed

When visualising the distribution of values for different sensors outliers are clearly visible (f.e. figure 4.9). For the removal, the whole dataset at once is used. This way we get between $10-60$ million sensor values depending on the sensor. Often used methods like Local Outlier Factor [Breunig et al., 2000] (LOF) or Isolation Forrests [Liu et al., 2008, 2012] cannot cope with that amount of data. Although the Isolation Forrest algorithm claims to have linear memory requirements through subsampling when building the trees, the algorithm did not run on a computer of 32 GB. Also when trying it on a 61 GB machine on Amazon web services (AWS) the algorithm did not do anything for 1h, so the experiment was discarded. The LOF paper does not say anything on memory consumption. While the Isolation Forrest threw a python memory exception on the 32 GB machine, the LOF did not terminate, and we ended the experiments after a few hours. Because of that, the strategy for outlier removal was modified in the following way:

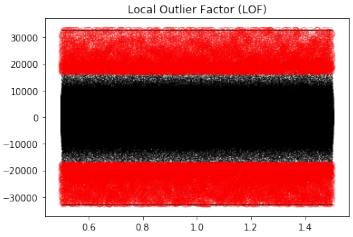
1. The data is permutated, and then divided into batches of 500.000 samples per batch.
2. On each batch LOF is performed.
3. We then count the number of outliers at the upper and lower $1,5\%$ quantile.
4. Then the fraction of outliers vs. inliers is computed. If there are more than 0.5% outliers we assume the sensor has outliers in the area of the upper and lower 1.5% quantile, else we declare the sensor outlier free.
5. We then manually tune the lower and upper quantile to reach a level where LOF says about 50% are outliers. We then treat all this data as outliers for the future.

An exception of the above is the flexion sensor. There exists a clear group of outliers at the end and the start of the spectrum (see figure 4.10e (b) for reference) which is most likely a form of clipping. However, they are grouped enough that the LOF does not assume them outliers. Therefore, for the flexion sensor, the last rule is ignored, and a threshold including these extreme values is set even if LOF does say they are not outliers. The results are individual thresholds per sensors expressed in lower and higher quantiles as a cutoff value. All chosen thresholds are lower than the $\pm 1.5\%$ quantiles. A visualisation of the outlier

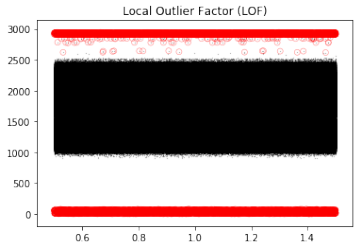
removal is shown in figure 4.10 for each sensor type. The resulting quantiles, the contamination and the amount of reduced data is shown in table 4.2. The effect of the outlier removal can easily be seen in figure 4.9, where the image with all the data is dominated by the outliers, while you see a distinct pattern after outlier removal. A selection of distributions of the accelerometer sensor from randomly selected users is in the appendix, figure A.1. A complete comparison of the distributions of the sensors for one user is in the appendix, figure A.2. There is no visualisation of the distributions of the sensors done over all users as this needed to long to compute.

stats	flexion	accelerometer	gyroscope	pressure	magnetometer
lower quantile	0.588948%	0.03%	0.05%	0.03%	0.03%
higher quantile	99.56215%	99.98%	99.95%	99.95%	00.05%
contamination	2.84388%	57.05%	67.1737%	43.33%	62.2364%
reduced data	1.0268%	0.0499914%	0.0999844%	0.0533914%	0.0786372%

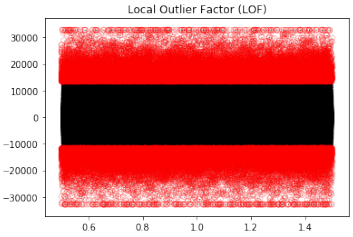
Table 4.2.: Lower and higher threshold expressed in quantiles for data assumed to be an outlier for the different sensors. Contamination expresses how many outliers LOF found within the threshold, reduced data tells how much of the original data is then seen as an outlier.



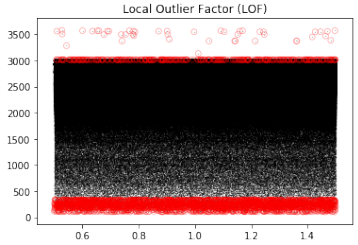
(a) Outliers detected for the accelerometer sensor



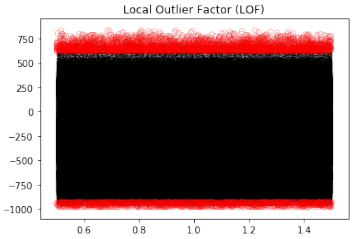
(b) Outliers detected for the flexion sensor



(c) Outliers detected for the gyroscope sensor



(d) Outliers detected for the pressure sensor



(e) Outliers detected for the magnetometer sensor

Figure 4.10.: Outliers detected for sensors with LOF outside the thresholds (Thresholds are defined in Table 4.2). A red circle is around an outlier. A black circle is a data point.

For visualisation, the outlier data is removed. In modelling the outlier data is filled by an average of the neighbouring values. Filling the outliers with the average makes sense as we have time series data and the individual points do depend on each other.

4.3.3. Description of Basic Data

After data cleaning, we have 22 users with clean data. A user has on average 126.742 samples of recording. This number of samples corresponds to an average recording time of 25min and 21s (this is the recording time, but not the duration of the experiment, which includes additional steps). Table 4.3 shows the number of time steps and the recording time of each participant. For the hardware unit, we combine the accelerometer and gyroscope to one IMU. Counted this way the glove has 25 sensors. The data consists of data of five different sensor types (accelerometer, gyroscope, magnetometer, flexion, pressure). All these sensors produce 62 channels of data per time point. The data is sampled with $83.\hat{3}Hz$, so each record from a consecutive recording is $0.0012s$ apart.

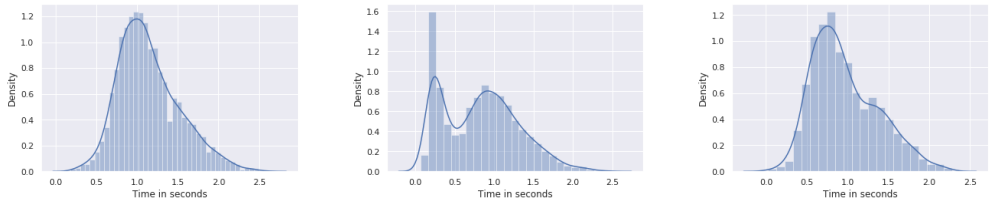
User Code	Samples	Recording Time
AB73	183008	0:36:36.096
AF82	105835	0:21:10.020
AL29	184276	0:36:51.312
AW18	103159	0:20:37.908
CB23	105331	0:21:03.972
CB24	106082	0:21:12.984
CF58	119296	0:23:51.552
DG12	104551	0:20:54.612
DH42	110435	0:22:05.220
DL24	181278	0:36:15.336
JL61	112566	0:22:30.792
JQ28	191596	0:38:19.152
JS52	187429	0:37:29.148
MF20	103021	0:20:36.252
MS55	108852	0:21:46.224
PC29	120138	0:24:01.656
PM32	180668	0:36:08.016
PS42	105795	0:21:09.540
RR45	105651	0:21:07.812
RW32	105821	0:21:09.852
SF1	103205	0:20:38.460
YW13	187082	0:37:24.984

Table 4.3.: Number of time steps and recording time for each participant.

4.3.4. Timing of Gestures

For deciding on the length of the sliding window, it makes sense to know how long a gesture can be in the dataset. Therefore we computed the statistics of the gesture types. Since the automatic label is fixed to 3s this statistic is not included.

The manual label type is the label type used for modelling. The separation between dynamic and static is too complex, but the manual label type is more exact than the automatic label type. The average gesture takes a bit more than a second for a manual gesture. Table 4.4 shows the statistics and figure 4.11 shows the according histograms. Another interesting comparison is the different timings for the manual label type for all user. Table 4.5 shows this comparison. In figure 4.12 we show all gestures sorted from the shortest



(a) Timing of manual labels. (b) Timing of dynamic labels. (c) Timing of static labels.

Figure 4.11.: The timing histograms for the different label types computed over all gestures and users.

Metric	Manual	Dynamic	Static
min	0.024s	0.072s	0.024s
max	2.616s	2.46s	2.28s
mean	1.147s6	0.831s	0.968s
var	0.144s	0.228s	0.162s

Table 4.4.: Comparison over the timings of a gesture over all users

on average to the longest for the manual label type. We can see that the variances are generally not so high. Also, the *min* and *max* values are different, but not too extreme. What is more, is that within the bounds some users tend for longer or less long gestures as we can see from the mean + kurtosis combination.

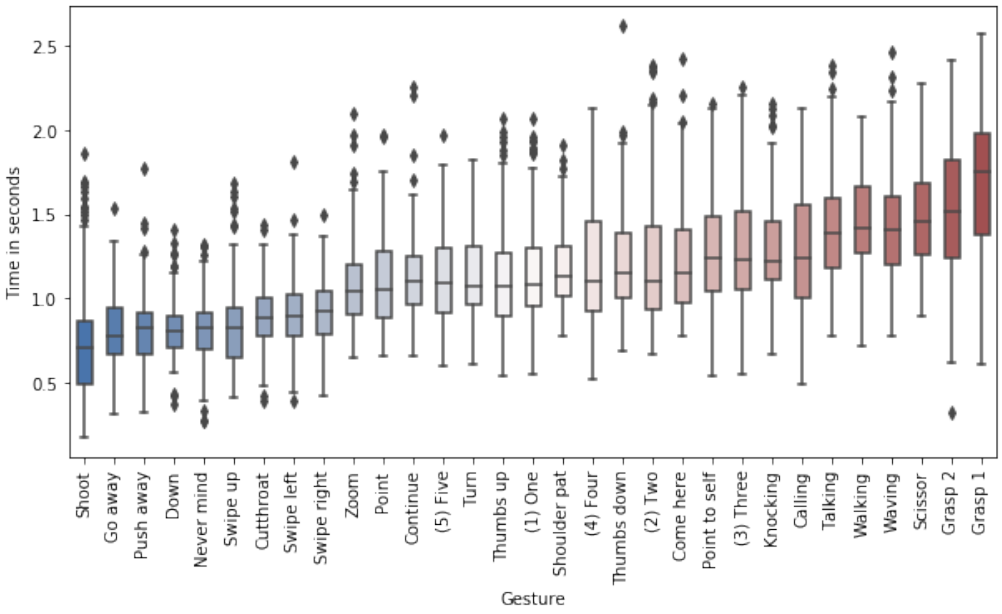
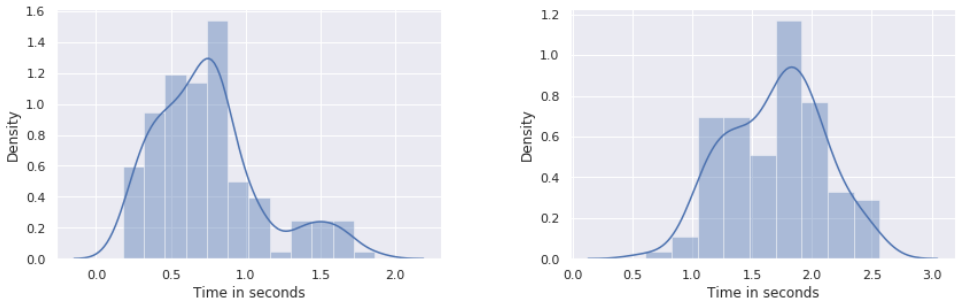


Figure 4.12.: Timing of static labels.

If we search the gestures for the one with the shortest and longest average manual gesture, we find that the Shoot gesture is the one performed fastest by the participants (0.75s on average) and the Grasp 1 is the one with the longest average timing (1.69s on average). If we search for the most uniformly performed gesture in terms of the gesture with the smallest variance in the timing it is the Down gesture which takes 0.82s on average but with a variance of only 0.03s. The most differently long performed gesture, on the other hand, is the Grasp 2 gesture with an average of 1.49s and a variance of 0.18s. Figure 4.13 shows the histograms of these gestures.

# lbls	min	max	mean	variance	skewness	kurtosis	user
307.0	0.492	2.016	1.090124	0.081020	0.977200	0.959664	AB73
153.0	0.564	2.160	1.071294	0.074505	0.933502	1.332054	AF82
308.0	0.276	2.088	1.080935	0.083344	0.464313	0.639929	AL29
152.0	0.612	2.244	1.237974	0.116471	0.607737	-0.313408	AW18
152.0	0.552	2.496	1.417184	0.101046	0.268666	0.712247	CB23
154.0	0.480	1.872	1.031922	0.067122	0.731036	0.679540	CB24
158.0	0.504	1.932	1.100127	0.104537	0.650358	-0.092005	CF58
154.0	0.180	2.028	0.932416	0.123974	0.856840	0.754349	DG12
153.0	0.288	2.460	1.007216	0.165798	1.022538	1.678622	DH42
301.0	0.204	2.256	1.289741	0.212006	0.003985	-0.952714	DL24
157.0	0.588	2.340	1.048968	0.072654	1.142722	2.578293	JL61
294.0	0.276	2.448	1.277224	0.192333	0.248830	-0.284948	JQ28
293.0	0.204	2.400	1.284491	0.130125	0.229425	0.123738	JS52
153.0	0.828	2.424	1.571843	0.142954	0.101736	-0.769278	MF20
149.0	0.492	1.788	0.938819	0.056836	0.785940	0.270476	MS55
156.0	0.324	2.616	1.456769	0.155767	0.211470	0.151588	PC29
308.0	0.396	1.608	1.048714	0.046416	0.097028	0.125705	PM32
152.0	0.372	2.436	0.907658	0.128420	1.903573	4.749115	PS42
153.0	0.276	2.148	0.988941	0.124843	0.499671	0.200059	RR45
146.0	0.228	2.220	1.184466	0.197616	0.464832	-0.507275	RW32
154.0	0.684	2.184	1.300597	0.094621	0.272054	-0.308279	SF1
307.0	0.300	2.148	0.973134	0.081594	1.000358	2.087748	YW13
4414.0	0.180	2.616	1.146843	0.143565	0.645843	0.276898	all users

Table 4.5.: Statistics of the timings of gestures when using the manual annotation type for each users, and all users in the last row.



(a) Timing of the Shoot gesture

(b) Timing of the Grasp 1 gesture

(c) Timing of the Down gesture

(d) Timing of the Grasp 2 gesture

Figure 4.13.: The timings of different gestures from the manual label type. Gesture (a) is the one with the shortest average in the dataset, (b) the one with the longes. (c) is the most uniform gesture, while (d) is the one with most variance.

4.3.5. Correlations in the Raw Data

It takes too much time to look at visualisations of all 63 channels' values distribution by hand and make the comparison in your head. Instead, we must go to statistical tests. We first test each channel for a normal distribution using the test from D'Agostino and Pearson [D'Agostino and Pearson, 1973; D'Agostino, 1971]. These results are too many to be shown in the thesis but can be checked in the notebook 'Distribution of values for channels and sensors'. With large data, even slight variations have that this test fail. So we define alpha to be $1e-7$ for rejecting the null hypothesis. No channel is normal-distributed.

Since we know the data is non-normally distributed, we need to take another test to compare the individual channels. We use the Kolmogorov-Smirnov (k-s) test which can compare two sample-based distributions. If we plot the heatmap of the average correlation of distributions of the different channels (average correlation in the form of the Kolmogorov-Smirnov test statistic) we find that the channels on average have little in common. However, there are patterns that, within one sensor group, the channels are indeed correlated. The correlation of the channels computed over all users is in figure 4.14. Additionally in figure 4.15 the user with the least correlation and the strongest correlation is shown. We can see that the differences between the users are there, but the pattern of what correlates with what does not change. If we look at the heatmap channel 10 and 11 (both flexion sensors) should be very similar, as should be 21 and 22 (both accelerometers) but the pairs themselves should be very different. Let's visualize the distribution for user MS55 of those four channels as histograms (see figure 4.16) to check if they are similar.
A common problem in any activity detection or gesture detection problem is the detection of the zero class vs the gestures. Usually, the zero class is far larger than the classes with actions in it. That also means that a good separation is key to a good performing algorithm. It is therefore interesting to see if there is a difference in the raw data between the zero class and the gestures. The zero class has a less large value range and less variety. An example for the magnetometer sensor can be seen in figure 4.17, the whole comparison is in appendix, figure A.4.

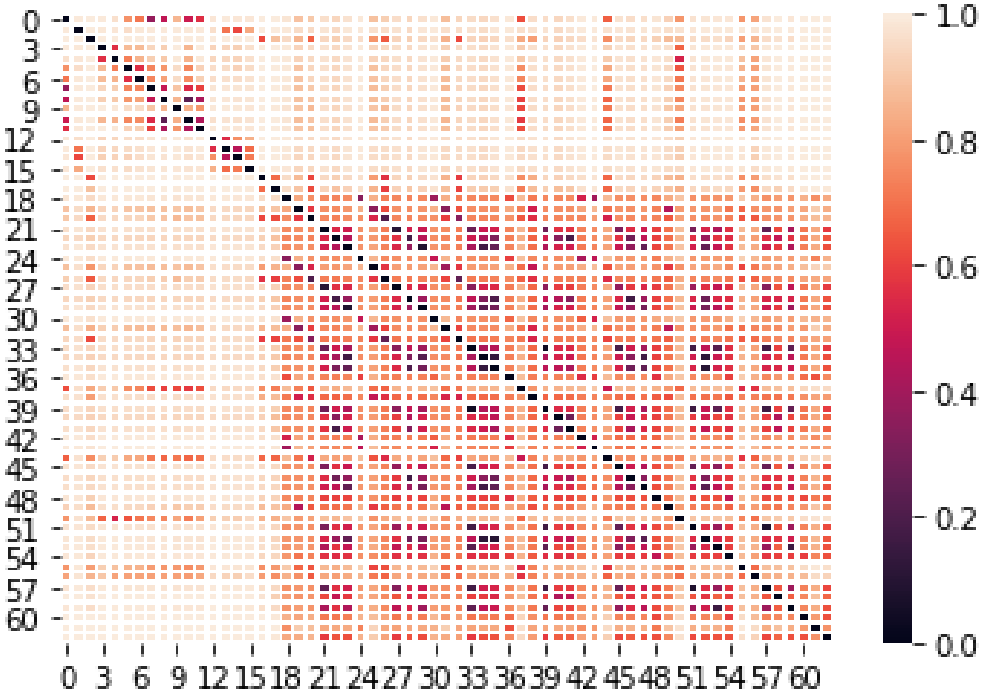
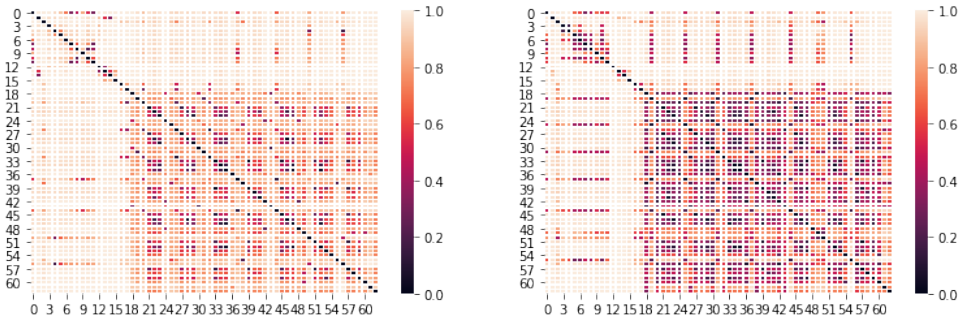


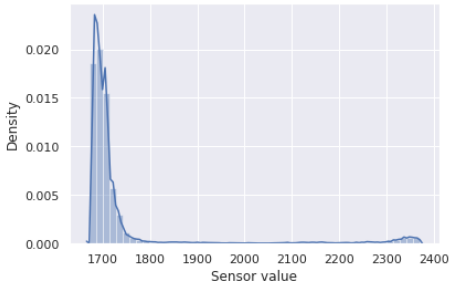
Figure 4.14.: A heatmap with the k-s test statistic of each channel in comparison with each other computed over all users. White means the sensor has nothing in common, black it is the same distribution.



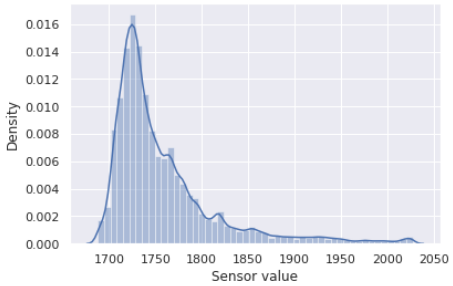
(a) k-s heatmap of user with least similar sensor value distributions (user MS55).

(b) k-s heatmap of user with most similar sensor value distributions (user CB23).

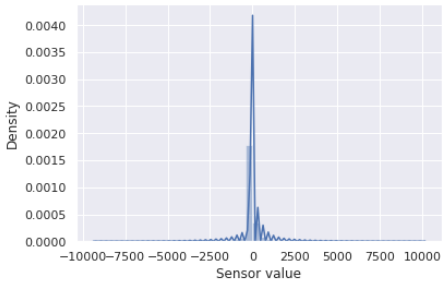
Figure 4.15.: Heatmap with the k-s test statistic of each channel in comparison with each other. White means the sensor has nothing in common, black it is the same distribution.



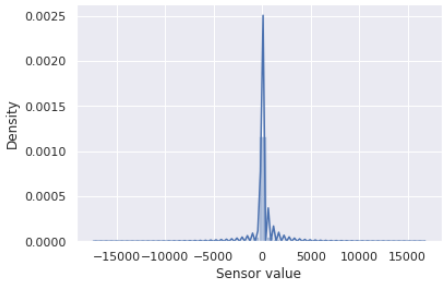
(a) Distribution of sensor 10 (flexion)



(b) Distribution of sensor 11 (flexion)

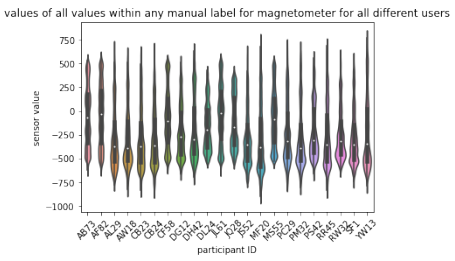


(c) Distribution of sensor 21 (accelerometer)



(d) Distribution of sensor 22 (accelerometer)

Figure 4.16.: Distributions of different sensor values of user MS55.



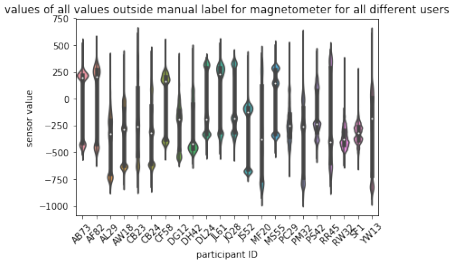


Figure 4.17.: Value distribution of the magnetometer sensor within labelled data (a) and within the zero class (b).

4.3.6. Raw Time Series

For feature engineering and modelling, we, of course, do not deal with independent values but with time series. It is interesting if there is a visible structure in the raw time series for a gesture. For inspecting that we create diagrams the following way:

1. For a specific gesture and a specific user let's find all repetitions of a gesture.
2. We compute an average signal plus a standard deviation signal out of these repetitions.
3. The average signal is plotted as a thick blue signal. The standard deviation is coloured as a transparent blue area around the average signal.
4. Then also all the original signals are plotted on top as thin lines.
5. If it is not over the whole time series we colour the time of the gesture within the plot as a red bar at the bottom.

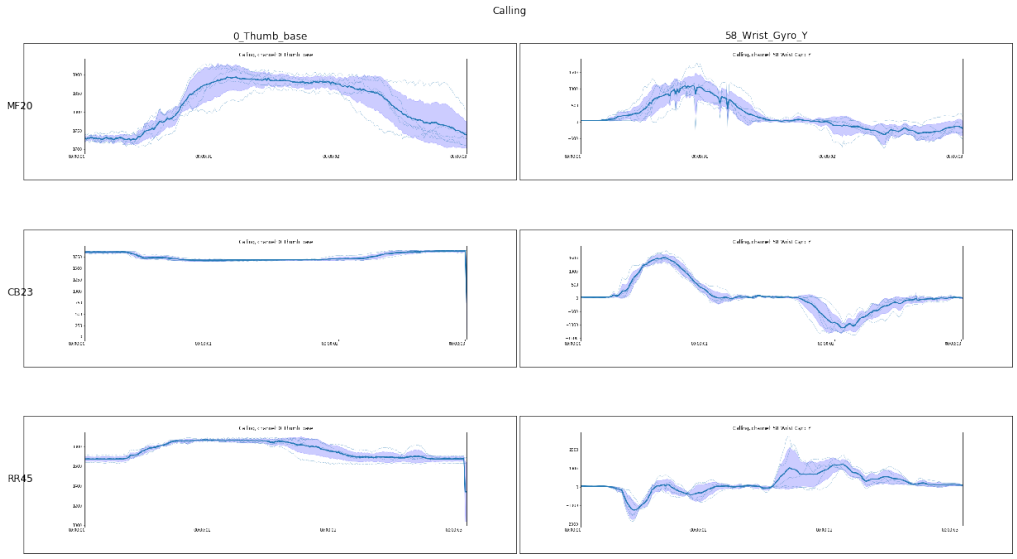


Figure 4.18.: Time series for the flexion channel at the thumb and the y-axis gyroscope at the wrist for the Calling gesture among 3 participants.

There are too many channels and users to look into all of them. Instead, we sample some user, channel and gesture combinations. We see the gesture Calling, performed by three different users (rows) for the first flexion sensor of the thumb and the gyroscope on the wrist on the yz-plane in figure 4.18. We can see structure in these plots. A larger grid of ten random channels and ten random users is shown in the appendix, figure A.3.

4.4. Feature Engineering

For modelling, redundant information can hurt the machine learning as it might put too much weight on that information. A goal of feature engineering and feature selection is to remove redundancy. Redundant features, if they originate from two different sensors, are useful in feature selection since it allows the algorithm to choose between the sensors. In contrast to the thesis, we did not do sensor selection in Luzhnica et al. [2016]. Also, in this thesis, the plan is to make a data-driven approach to find the right features instead of the expert-driven approach in Luzhnica et al. [2016]. Therefore the approaches to feature engineering are not the same.

In Luzhnica et al. [2016] we remove the gravity component from the acceleration and only keep the linear acceleration in the dataset. The value of the gravitation alone to detect the posture was not yet clear at that time. The gravity component was removed using a complementary filter [Higgins, 1975], which typically gives satisfying results and is computationally less expensive and less complex to implement than a Kalman Filter [Kalman, 1960]. In addition, to get axis independent values from the accelerometer and gyroscope sensors, we computed the absolute energy signal ($energy = \sqrt{x^2+y^2+z^2}$) for each of our IMUs. Magnetometer values were discarded as their values provide information related to the absolute location of the hand whereas gesture recognition should work regardless of the hand's location. Furthermore, all data dimensions are normalised with zero mean and a standard deviation of 1.

4.4.1. Window Length and Step-Size

As a basic unit for classification we use sliding windows, i.e. data windows of fixed sample size that constitute snapshots of the continuous data stream. Features are computed per window. Sliding windows are a well-established method of feature extraction used in many domains (speech to text [Sejnowski and Rosenberg, 1987], activity recognition [Koskimaki et al., 2009; Krishnan and Cook, 2014; Ortiz Laguna et al., 2011], etc..). Their advantage is that the extracted features can be used with almost any algorithm [Dietterich, 2002]. They typically have two hyperparameters: size and step. For parameter selection, we cross-validated the data with several window sizes (140, 160, 180, 200 samples, where 1 second contains 85-87 samples). We used steps of 20, 30, 40 and 50 samples and again used cross-validation to select a value for this parameter. The best configuration is the one with a window length of 200 frames, step size of 20 and 15 Fast Fourier Transformation (FFT) coefficients with an average cross-validation (across all compared algorithms) score of 95.6%. Using only five FFT components and changing the step size to 50 does change the accuracy only minimally to 95.3%. This configuration is computationally less expensive. For the thesis, we do not do this hyperparameter search for the sliding window and step size again, but take the window size of 200 and step size of 50. Expressed in time, that means the window is 2.6s long and makes a step for 0.6s.

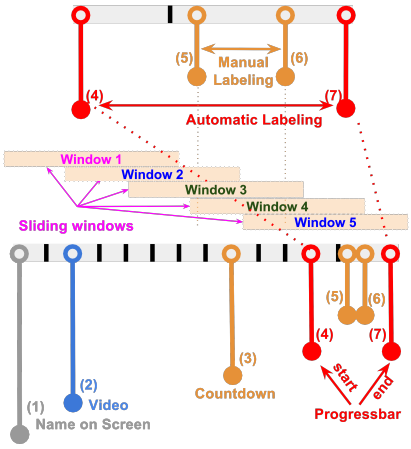


Figure 4.19.: Experiment timeline for a single repetition and sliding windows construction.

4.4.2. Annotation of the Sliding Windows

As for the labels, when training the model and for validation, in Luzhnica et al. [2016] we consider one window to have a gesture label only if the window contains the whole gesture. Otherwise, we label it as idle class.

For testing, we simulated a system where sensors stream continuously. When sliding windows are moved over the continuous sensor signals, then there are windows with no gesture in it (window 1 in figure 4.19), with partial gestures in it (windows 1 and 5 in figure 4.19), and windows with full gestures in it (windows 2 and 3 in figure 4.19). For model training and validation only windows with no or full gestures were used, while for evaluation of the selected algorithm in realistic settings, the algorithm was also evaluated on windows that contain a partial gesture. Moreover, there is no balancing (neither in the training nor the test data set) but class weighting is used when training to prevent bias towards the larger classes. This configuration of the test set corresponds to the data that would be available in a real-world continuous sensor data stream. For windows that contain a partial gesture, we assume the algorithm prediction is correct when the classification outcome is either the idle class or the correct gesture class that is partially in the window. We refer to this strategy as dual labelling in the test set.

4.4.3. Window Features from the Original Paper

As data dimensions we understand the following: (x, y, z, *energy*) values of gyroscope, (x, y, z, *energy*) values of accelerometer, values of pressure sensors, values of bending sensors. For each data dimension, we used the following descriptive statistic as features: minimum, maximum, range, average, standard deviation and signal energy from the sliding windows. Minimum and maximum values of the flexion sensors should contribute to capturing the static part (posture) of the gesture. On the other hand, the derived values from the norm value of the gyroscope and the accelerometer should capture orientation independent motions aspects, as the norm is just the intensity of the accelerometer or gyroscope in any direction.

For the gyroscope and accelerometer values, we also used the spectrum features, namely the amplitude of Fast Fourier Transform (FFT) coefficients for the signal in the given window. For FFT, one has to decide which and how many coefficients are used. Typically, either the n first or the n largest (by amplitude) coefficients are used [Mörchen, 2003]. Figure 4.20 shows that in our case, on average, the amplitudes of FFT coefficients decrease monotonically. This monotonous decrease means that the first coefficients are the largest ones, which in turn means that the lower frequencies are dominant. Therefore, using the amplitudes of the first n coefficients is a good way to proceed. For selecting a suitable number of first FFT coefficients, we use again cross-validation to choose amongst the following options: $n \in \{5, 6, 10, 15\}$.

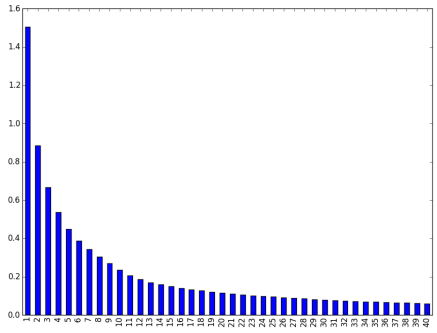


Figure 4.20.: Average amplitudes (over all signals) of the first FFT coefficients (excluding the zeroth coefficient) for all 200 frame windows.

In total, we extract 78 statistical features from flexion sensors, 30 statistical features from pressure sensors, 336 statistical features and $n \times 56$ FFT features for IMUs, where n is the number of first FFF components used for the window. It is worth pointing out that the majority of the features comes from motion sensors (IMUs). Altogether, this gives 724 features using 5 FFT components and 1284 when using 15 of them.

To avoid correlated features, we calculated correlations between features and automatically removed the features that highly correlate with each other (with the absolute Pearson correlation index more or equal to 0.99). Finally, extracted features are normalised with zero mean and a standard deviation of 1.

4.4.4. Extended Window Features

In this thesis, we kept the size of the windows and step size from our original work in Luzhnica et al. [2016]. Contrary to that the training set also contains windows with partial gestures.

We split linear force from the accelerometer using a complementary filter [Higgins, 1975], but keep the raw accelerometer signal which contains the gravitation aspect. We also compute the direction cosine from the gyroscope, which is a normalised representation of the current angle. Like we did in Luzhnica et al. [2016] we add the energy signal for accelerometer and gyroscope. By adding these channels, we generate four more channels (three for linear acceleration, one for energy) for each accelerometer sensor and three more channels for each gyroscope. We have seven IMUs sensors giving 49 additional channels, so 112 channels.

For each window, all channels are transformed into a frequency domain using the FFT and into a frequency/time domain using the wavelet transform (CTW). Applying these two transformations transforms 112 signals in the window for extraction into 273 signals as the basis for extraction. From each signal, the 12 statistical features described in section 3.4 are extracted. From the raw 112 signals we perform peak

detection using wavelets and extract four peak features (number of peaks, minimal peak, maximal peak, mean peak value). For the 112 signals in the frequency domain, we additionally extract the spectral centroid, the bandwidth, the power spectral entropy, the first five coefficients of the transform and the sum of those coefficients, giving nine further features. For the frequency/time domain signals, a sum for each bin of the CWT transform is returned additionally to the 12 statistics. We fix the transformation to 10 pins, so we add ten extra features from these signals. This gives $112 * (3 * 12 + 4 + 9 + 10)$, so 6.608 features from each signal.

We also add features for pairs of channels. The angle of a pair of channels, the Spearmen correlation and p-value and the difference in the mean signal are computed for the raw signal and after transforming both signals into the frequency domain. Additional the cross-correlation shift is computed as a feature, giving nine features for each pair. We compute these features only for the first and second row of pairs of the flexion sensors. As each row has five channels, we have $2 * \binom{5}{2}$ pairs of data (so 20 pairs). With nine features for each pair, this gives additional 180 features. In total we have 6.788 features.

The work of this thesis is a data-driven approach, and one might immediately see that not all features are generated. Especially one could return all 200 FFT coefficients and compute all channels' combinations. Adding all FFT coefficients would add 195 features to each channel so that it would add 21.840 features. Adding all combinations would mean to add $\binom{112}{2} * 9$, so 55.944 features. There are two problems with this:

First, our training dataset has about 70.000 entries. With really generating all features we would have 84.392 features, so more features than instances in the training data. Many algorithms assume that the number of features is less than the number of instances, and an own set of modelling would have to be chosen for this problem.

Second, a practical reason exists. With the 6.788 feature transformation it already takes 25h on a 31GB Ram, Intel i7 k7700 4x4.3 GHz machine, and with errors the first conversions took days. Also working with the initial models took hours. Therefore we limit this work to compute as many features as computationally feasible given the resources instead of all features. We drop the additional FFT features as they are already shown to not add too much to the resulting classifier in our work in Luzhnica et al. [2016] and in Mörchen [2003]. We drop the pairwise features too as they are a different type of features and we concentrate on the single channel features. We mainly add some of those to see the effect of sensor selection on the pairwise features.

4.5. Train and Test-Set Split

It is common praxis in the machine learning community to split the train, test and validation set by selecting a random subset of the data. According to the blog post by Rachel Thomas[4], this is however only a good praxis if the data is truly identically and independently distributed (i.i.d.). If data is time-series data, the data of each sample is dependent on the data of the past sample, and the samples are not i.i.d. anymore. Using sliding windows is a way to lessen that effect. However, since our windows overlap if we select randomly we have the danger that our training set contains a lot of data which is very similar to data in the validation and test set.

We use the random split into train and test set in our work in Luzhnica et al. [2016] with 80% train and 20% test set. The validation results are achieved with 5-fold cross validation.

In this thesis we do not use the random splitting provided by scikit-learn. Instead, we split the sets by hand by using the following strategy: For the test set, we choose one random user (CF58) and save it for the test set. For the validation set, we iterate all users and select one random time the user performs one gesture. Then we move all windows capturing that gesture into the validation set. Additionally, we delete all the windows before and after without a label, so there are no overlapping sequences. Since gestures are performed with a different speed for each user, we then compute the average amount of labels used to

capture the gestures from a user and move a section of the zero class of that user into the validation set. This selection of the zero class means that the validation set is balanced also among the zero class. We also delete that section of the zero class from the training data. With that method we have it guaranteed, that there is no window which has an overlapping window in any other set. The training set resembles the realistic case of a new participant using the glove.

4.6. Modelling and Feature Selection in the Original Paper

In Luzhnica et al. [2016] we look into several different models for the gesture recognition. As previously mentioned, similar to activity recognition solutions, we emphasise the motion sensors and follow a similar approach (sliding windows) as is frequently used in activity detection. Therefore, we chose classification algorithms that have proven to provide robust performance on activity recognition using wearable sensors [Koskimaki et al., 2009; Krishnan and Cook, 2014; Ward et al., 2006; Ward, 2006], namely:

- K Nearest Neighbours (KNN)
- Linear Discriminant Analysis Classifier (LDAC)
- Support Vector Machines (SVM) with a linear kernel
- Logistic Regression (LR)

According to the survey presented in Bulling et al. [2014], discriminative classification algorithms are very useful in identifying features that mostly contribute to discriminations between activities using wearable sensors. Therefore, our discriminative classification algorithms (in our case SVM and LR) should work very well in case extracted features of windows capture the gesture well. LDAC is suitable, when after transforming the data in a linear manner with LDA, there exist linear decision boundaries between the classes. On the other hand, KNN uses the notion of distance in feature space, and it can perform well even when linear separability is not possible.

Considering the number of features we have (724 when using 5 FFT components, 1284 when using 15 of them), we were concerned about overfitting. Therefore, we employed dimensionality reduction techniques prior to training. We used Principal Component Analysis (PCA) which applies an orthogonal linear transformation of the data, in an unsupervised manner, resulting in a maximised variance of data in the transformed space. On the other hand, we also used the supervised linear transformations, namely Linear Discriminant Analysis (LDA) and also its state-of-the-art alternative Spectral Regression Discriminant Analysis (SRDA) [Cai et al., 2008]. The latter methods utilise class labels for minimising the within-class variance and maximising between-class variance (in the transformed space). An extensive analysis on how the used algorithms and dimensionality reduction work, including the mathematics behind it, can be found in Bishop [2006]. In our cross-validation process, the classifiers have been trained in both ways: without any dimensionality reduction and with prior dimensionality reduction transformations.

For each window and step-size, we prepared the dataset as follows: We selected only those windows from the complete dataset that are "unambiguous windows", i.e. windows that contain either no gesture, or a full gesture (see figure 4.19, where window 1 contains no gesture, windows 2 and 5 contain a partial gesture and are not part of the training data set, and windows 3 and 4 contain a full gesture). The rationale was that the classifier should only learn the full gestures, not parts of gestures.

Moreover, the classes in our data are unbalanced as the majority of the windows are labelled as "idle class" (no gesture). The following procedure achieved balancing: First, we calculate the average number of windows per gesture which we will denote by k. Then we removed, before splitting to train and test set, all idle class windows except k random number of idle class windows from the data set.
The corresponding training data set was then 80% randomly chosen windows, and the test data set the

[4] https://www.fast.ai/2017/11/13/validation-sets/

remaining 20%.

We used the following procedure to select the winner combination of configuration (parameters) and algorithm:

First, we compute the performance of each combination "configuration/algorithm" by 5-fold cross validation over the training data set. Then, for each configuration, we compute the average performance over all algorithms to select the winner configuration. The winner algorithm would then be the best-performing algorithm for this configuration. The rationale for this procedure was that we wanted to have the configuration to be as robust as possible concerning an algorithm to avoid overfitting, i.e. we did not want to select a configuration that only works for a single algorithm.

The best configuration is the one with a window length of 200 frames, step-size of 20 and 15 FFT coefficients with an average cross-validation (across all compared algorithms) score of 95.6%. Another configuration with less computationally intensive parameters, namely window length of 200 frames, window steps for the sliding windows of 50 and only 5 FFT coefficients had an average cross-validation score of 95.3%. Considering that the score difference is minimal whereas the computation efficiency is higher, we selected the latter configuration. For this configuration, the best performing algorithm was LDA+LR with a cross-validation f_1 score of 99.8%. Here, initially, LDA was used to perform dimensionality reduction to 32 components and then a logistic regression algorithm was trained and tested on the dimensionally reduced data.

Removing FFT calculations during the gesture extraction can speed up processing the data stream. Removing the FFT components from all accelerometer and gyroscope dimensions results in a recognition f_1 score (when considering dual labelling of the ambiguous windows in the test set) of 98.2%.

4.6.1. Performance on Continous Sensor Data

As our results reveal, we achieve a high classification accuracy in general. As presented in figure 4.21, the prediction confidence is also high. It is important to stress that our results were achieved by including in the test set the windows that contain partial gestures. In a live gesture recognition system, there is no way of excluding them. More specifically, in a live scenario, we need to get a sliding window over a stream of data, as visualised in figure 4.19, and since we don't know when a gesture starts and when it ends, we can't know beforehand whether a partial gesture is in a window.

As realistic algorithm performance, we consider its performance on the following dataset: All windows are used in the test data set, which includes those with a partial gesture window. A partial gesture window is the one that contains only a portion of a gesture (see window 2 and 3 in figure 4.19). Moreover, there is no balancing (neither in the training nor test data set) but class-weighting is used when training to prevent bias towards the larger classes. This corresponds to the data that would be available in a real-world continuous sensor data stream. For windows that contain a partial gesture, we assume the algorithm prediction is correct when the classification outcome is either the idle class or the correct gesture class that is partially in the window. We refer to this strategy as dual labelling in the test set.

On this test set, the LDA+LR algorithm with a window size of 200, a step size of 50, and with only the first 5 FFT coefficients in the feature set, performs with a 98.5% f_1 score. The confusion matrix is given in table 4.6 and the receiver operating characteristic (ROC) curve is visualised in figure 4.21. Here, from 9581 windows, 9440 were classified correctly. From the correctly classified windows, 1618 contained full gestures, 2802 partial gestures and 5020 contained no gesture at all (belonged to the idle class). On the other hand, 141 windows were misclassified from which 6 contained full gestures, 106 partial gestures and 29 came from the idle class.

	Nothing	*(1) One*	*(2) Two*	*(3) Three*	*(4) Four*	*(5) Five*	*Thumbs up*	*Thumbs down*	*Point to self*	*Shoot*	*Scissor*	*Cutthroat*	*Continue*	*Knocking*	*Waving*	*Come here*	*Go away*	*Push away*	*Never mind*	*Talking*	*Calling*	*Walking*	*Shoulder pat*	*Point*	*Swipe left*	*Swipe right*	*Swipe up*	*Down*	*Turn*	*Zoom*	*Grasp 1*	*Grasp 2*
Nothing	6024	2	1	5		4	3	1	7	1	1	4	2		1		6	12	2		2		3	5	15	3	4	28	2	7	6	8
(1) One		107																														
(2) Two			113																													
(3) Three				110																								1				
(4) Four					116																											
(5) Five						97																										
Thumbs up							108																									
Thumbs down								116																								
Point to self									117																							
Shoot										89																						
Scissor											126																					
Cutthroat												110																				
Continue													96											1								
Knocking														123																		
Waving															133																	
Come here																111																
Go away																	93															
Push away																		111														
Never mind																			80													
Talking																				107												
Calling												1									128											
Walking																						110										
Shoulder pat																							113									
Point																								112								
Swipe left																									103							
Swipe right																										95						
Swipe up																											108					
Down	1																								1			107				
Turn																													127			
Zoom																														95		
Grasp 1																															138	1
Grasp 2																																117

Table 4.6.: Confusion matrix of classification using dual labelling in test set. Note that zero values (no misclassification) have been removed from the table for better readability.

In the test set, we used the dual labelling strategy which delivers an accuracy of 98.5% (see table4.6 and figure 4.21). We argue that dual labelling is acceptable for a live system: In the end, it is just a matter of how fast the recognition system realises that the gesture is being performed. In case it predicts the correct gesture class (the one that is partially contained in the window), then we can recognise the gesture even before it is completed. Otherwise, if the classifier predicts it to be an idle class window, then the next window (or previous one) will be a window with the full gesture in it and will be classified correctly.

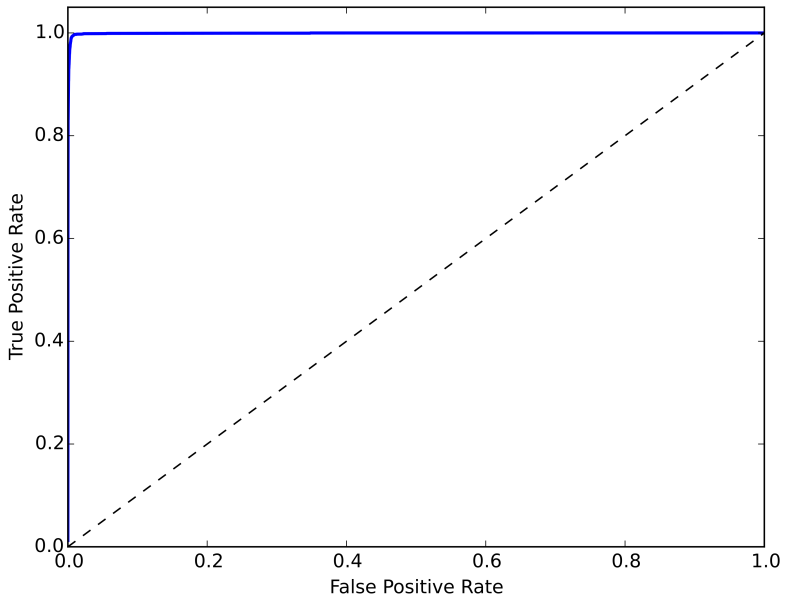


Figure 4.21.: ROC curve for weighted average of all classes.

4.7. Data-Driven Modelling and Feature Selection

In comparison to our work in Luzhnica et al. [2016], we use a data-driven approach here. We reuse the results from window size and step size, but instead of a maximum of 1.284 features, we start with 6.788 and use algorithmic approaches (filters and wrappers in feature selection, see Russell and Norvig [2010]) to reduce the number of features and later sensors. Feature selection must not be confused with dimensionality reduction like PCA which we used in Luzhnica et al. [2016] to reduce the data to 32 dimensions. You still need all the features and dimensions to compute this reduction, i.e. compute a data-point for prediction. With the feature and sensor selection (similar to our results for the FFT coefficients) the features do not even need to be computed.

Another difference is the training, test and validation set. We use partial windows already in training and validation. Also, the partial windows are annotated with their class. If that works, the classification system can recognise gestures as soon as a window hits the gesture. Given a step size of 50 instances, which corresponds to $0.6s$, this means that a gesture can be recognised in less than $0.6s$. We did not balance the training dataset but used class weights. We did, however, balance the validation set. We did not use random sampling for cross-validation or testing but use the strategy described in section 4.5 for our split.

Like in Luzhnica et al. [2016] we scale the data to a unit of 1 and centre it. However, here we use the `RobustScaler` from scikit-learn [Pedregosa et al., 2011] to do the scaling. The robust scaler uses the interquartile range to do the scaling, which is more robust on non-uniformly distributed data.

4.7.1. Extended Features - Initial Feature Selection and Modelling with Filters

A first step in the data-driven approach is to get the number of features down to a number suitable for modelling. The question arises what a good number of features is. If all features are uncorrelated, a good number of features is to have just one less than data points, so $N-1$ features. That all features are uncorrelated is rarely the case, and in our case already the data channels themselves have some correlational structure. If the features are correlated a good number of features is $\sqrt{N}$ [Hua et al., 2004]. We have a bit more than 70.000 instances in our dataset so a good number would be around 300 features ($\sqrt{70000} \approx 264$).

We start with filter methods. Filters are, as the name suggests, methods to select only those features which adhere to certain criteria and filter out the rest. After each filter step, we try to fit the most common scikit-learn classifiers to the data and report the validation sets f_1 score. In this step, we still have many features, and only a few models can deal with this well. We report all models we tried and which models worked with the whole dataset.

We initially remove all constant features using the `VarianceThreshold`. This reduces the dimensionality from 6.788 to 6.368. After that we reduce the size to 60% of the 6.368 using a `SelectPercentile` filter. This filter is parameterised by a selection criterium which is either χ^2, f-ANOVA or the mutual information $MI(X;Y)$. We choose the f-ANOVA and mutual information criterium for the selection.

Even here, `SelectPercentile` can not work with the complete data. We, therefore, subsample the data in a stratified manner. Stratified manner means that we subsample but keep the relative class size to each other constant. With 2.500 samples (so a 2.500×6.368 data matrix) `SelectPercentile` works well, and we generate two candidate datasets with 3.821 features.

We try to fit all the classifier models from scikit-learn [Pedregosa et al., 2011]. These are: Three different SVMs which mainly have a different kernel and contamination parameter (L-SVM, SVC+RBF, NuSVM+RBF), Gaussian Processes with and without restarts (GP and R-GP), Logistic Regression (LR) and LR with ℓ_1 regularization (L1-LR), Passive-Aggressive (PA), Stochastic Gradient Descent with a hinge loss (SGD), LDA, QDA, Nearest Centroid (NC), k-Nearest Neighbours (kNN), Radius Neighbours (RN), Naive Bayes with a Gaussian kernel (NB(G)), Decision Trees (DT), Random Forests (RF), Extra Trees (ET), Bagging with kNN models (B-kNN), Bagging with SVC models (B-SVC), Ada Boost (AB) and

Gradient Boost (GBo). The details of all these models are presented in section 3.7.

As discussed in section 3.7 bagging is a method to reduce variance and overfitting, and extra trees are especially well suited for dealing with high dimensionality and redundant and noisy features. In line with these results on all of these high dimensional datasets (6.368 and two times 3.821 dimensions aka. features) the extra tree is the best performing algorithm, with a validation error of 96.32% on the 6.368 feature set, 96.17% on the f-ANOVA feature set, and 96.72% on the MI feature set.

Many of the algorithms did not work on these datasets. Also, on average, the algorithms perform better with the two reduced (f-ANOVA and mutual information) datasets than with the full dataset (52,16% vs 65,63% with the f-ANOVA and 55,01% with the MI dataset). In the f-ANOVA case, the classifiers scored better on average, while the best classifier was trained with the mutual information metric. It is again an Extra Tree with 96.72% f_1 score on the validation set. Given the long running-times, we did not do hyperparameter tuning for the individual algorithms. The results are summarised in table 4.7.

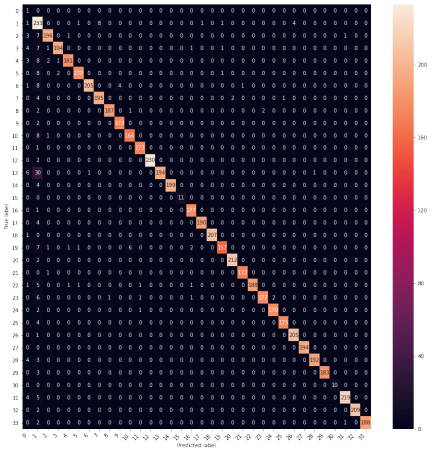


Figure 4.22.: Confusion matrix from the extra tree classifier on the validation set.

algo.	70kx6k f_1 validation	f-Anova 70kx3k f_1 validation	MI 70kx3k f_1 validation
PA	30.50	80.99	46.91
SGD	7.66	55.95	7.03
LDA	94.94	93.99	94.05
QDA	62.29	60.88	55.35
NC	8.12	11.59	5.95
GNB	21.47	56.90	38.86
DT	–	92.34	92.80
RF	96.01	95.68	96.43
ET	96.32	96.17	96.72
AB	–	11.85	16.04
mean	52,16	65,63	55,01
L-SVM, SVC+RBF, NuSVM+RBF, kNN, B-kNN, B-SVC, GBo, Interupt after 1h GP, R-GP, Memory Error LR, L1-LR, Kernel died			

Table 4.7.: f_1 scores in percent of algorithms with different feature set sizes.

4.7.2. Extended Features - Model Based Feature Elimination

With about three thousand features it is possible to use Recursive Feature Elimination (RFE) and Recursive Feature Elimination Cross Validated (RFE-CV). These methods continuously fit any algorithm which can give the importance of features after fitting and use this importance to remove features which do not contribute enough to the final score. It then repeats the process until the score of the model reaches some threshold. Common choices for such algorithms are SVMs [Ma et al., 2017; Park et al., 2018; Yan and

Zhang, 2015], Random Forests [Degenhardt et al., 2017; Fernandez-Lozano et al., 2015; Gregorutti et al., 2017; Ma et al., 2017] or Lasso [Fernandez-Lozano et al., 2015; Spencer et al., 2017]. We use the Lasso (ℓ_1) regularisation for this. Depending on the usage of the concrete algorithm (RFE vs RFE-CV) and the strength of the ℓ_1 regularisation for the Lasso (we vary α with 0.01, 0.1, 0.5, 0.9) this gives different results in the range of 192 to 1910 features. After these reductions, we fit the models again. The accuracy of moderately sized feature sets (765 and 383) are better on average (with $70,68\%$ and $69,43\%$ respectively) than the extreme versions with 1910 or the two 192 sets. This different accuracy for different feature set sizes means we are a bit above the theoretic amount of correlated features (which is around 300 in our case), but still in the range of this number. It is interesting however, that for algorithms that mainly reduce variance, like random forests and extra trees, the number of features has little impact (all accuracies are above 95%) until it reaches the 192 feature case where they still perform well with 94% and 93.21% f_1 score on the validation set.

Type	RFE	RFE-CV			
α	0.1	0.01	0.1	0.05	0.9
# Feat.	1910	765	383	192	192
Alg.	f_1 val	f_1 val	f_1 val	f_1 val	f_1 val
PA	39.04%	92.13%	85.08%	47.43%	19.43%
SGD	6.97%	92.32%	86.44%	38.70%	07.03%
LDA	93.36%	91.85%	86.05%	76.20%	67.69%
QDA	60.59%	91.59%	95.47%	87.20%	75.86%
NC	5.84%	24.26%	31.04%	11.02%	06.81%
kNN	–	93.99%	86.42%	59.18%	30.28%
G.NB	37.71%	67.54%	66.65%	61.38%	39.93%
DT	92.14%	93.44%	92.84%	91.25%	89.61%
RF	96.12%	95.51%	95.51%	94.57%	93.36%
ET	96.14%	95.60%	95.09%	94.00%	93.21%
AB	16.36%	06.42%	08.10%	10.01%	11.99%
GB	–	03.51%	04.46%	03.33%	–%
mean	$54,43\%$	$70,68\%$	$69,43\%$	$59,94\%$	$48,65\%$

Table 4.8.: Algorithms on differently reduced datasets.

4.8. Sensor Selection

It is interesting to see how well the glove would do if you leave out some sensors. If this is easily done, it enables a better use of prototypes for hardware developement. You first build a prototype with too many sensors and record data for all envisioned use cases for this hardware. Then you find a minimal configuration which yields the wanted performance for a chosen use case. Given a cost for each sensor and costs of integration for each sensor, it is possible to optimise for the expenses directly. Then you can build a next prototype iteration with the correct hardware. Much work raises the issue of expensive sensor setups like in home monitoring [Spencer et al., 2017], home surveillance of elder people [Fiorini et al., 2017], wearable sensors for ergonomics [Maman et al., 2017] or sensor configurations for surgical robots [Gómez-de Gabriel and Harwin, 2015].

So why is this hard? Can we not just try all configurations? It takes days to evaluate all models with this dataset. Even with a Hadoop cluster, it takes minutes. Depending on how you count we have 25 sensors. To assess all combinations 2^{25} trials are needed, which is about 33 million trails. If we assume we have a well-performing cluster and evaluating the models for a trail takes two minutes, we would wait 126 years for the result. If there are very few sensors and evaluation is easy one can search for solutions per hand. Lee et al. [2017] evaluate eight IMUs to asses if a construction worker has a good posture. They

compare a calculated angle of the IMU to how straight a worker stands and analyse the correlation of this measure to the target. Instead of a combination of sensors, only the one best performing sensor is found by their approach. Gómez-de Gabriel and Harwin [2015] evaluate sensor configurations for surgical robots. They consider the costs of the sensors and compare the sensor configurations manually by performing usability studies done with surgeons in a virtual environment. Another strategy is to make an extensive meta-study on all sensor configurations tried so far like Chambers et al. [2015] did for physical movement sensors like IMUs (in the paper it is called microsensors) for sport specific movements. This approach gives expert knowledge and is helpful, but it can not be used to evaluate the design of a new system.

In wireless sensor network research, sensor selection is defined as the selection of sensors to spread information or energy through the network of sensor nodes, but this is only a temporal selection [Chang and Tassiulas, 2000; Pradhan et al., 2002; Woodcock and Strahler, 1987]. In this work, sensor selection is finding a subset of sensors which still allow a system to perform at a specified accuracy for a given target. The given target is the set of gestures to recognise. While it is, of course, possible to use sensor selection to rebuild the glove still allowing the 31 gestures, it is also possible to select a subset of the gestures, often with an application in mind, and run the sensor selection with that.

An option to make sensor selection is by using a greedy approach coupled with some way to rank the sensors. A possibility to rank the sensors is using the ℓ_1 regularisation, which for linear models is usually called Lasso. Using the Lasso for feature selection is very common. It is used to improve the accuracy and using fewer features in many tasks like for glucose monitoring devices [Zanon et al., 2013] or to find features for physical fatigue at the workspace using wearable sensors in Maman et al. [2017]. This work also explains which sensors are selected with the feature selection. It does not explicitly model a sensor selection model but uses feature selection until only a few sensors are left. This is possible due to the low number of sensors (5 shimmer IMUs and a heart rate belt).

Like us, Spencer et al. [2017] uses Lasso to perform feature selection. The system in Spencer et al. [2017] is built by first starting with an empty set of sensors and performing the best first search algorithm. It tries if the system gets better if any of the five sampled sensors improve the accuracy of the system. The best set is then saved as a solution, and the process is repeated until a threshold is reached or the system does not improve. Based on this collected sets of sensors the system then ranks the sensors by comparing all sets and removing the sensor which did least improve the accuracy.

Our system does directly start with a full configuration of sensors and recursively removes the one which has the lowest impact or lowest costs. The target of Spencer et al. [2017] is a regression task. Our system is a classification system, which also allows us to easily try out subsets of the classification and perform sensor selection there. Additionally, our approach allows us to build in the costs of the sensors into the algorithm, and directly optimise for the total costs of the system.

4.8.1. The Recursive Sensor Elimination Algorithm

We present a greedy algorithm for sensor selection, based on the recursive feature algorithm. We call the basic algorithm recursive sensor elimination (*RSE*) and also present its extensions.

First the algorithm needs a mapping function *fts* from features to sensors. Let $fts(D') \mapsto \vec{s}, f_{\text{map}}$ be a mapping function which maps the columns in the feature matrix D' onto the sensors the feature is generated from, $\vec{s}$ be a vector tracking how many features are active for a sensor, and f_{map} be a map of sensor indices to feature indices. If the column vector $\vec{f_a} \in D'$ is a vector of a feature created from sensor s_b than the function $fts(D') \mapsto \vec{s}, f_{\text{map}}$ increments the count of $\vec{s}$ at b and adds the feature number a to f_{map} as position b. In case a feature is generated from two sensors, the feature is added twice at the appropriate positions.

Given $\vec{s}, f_{\text{map}}$ we need a heuristic function which tells which sensor is the most promising to remove. We call this heuristic function $select(\vec{s}) \mapsto s_b$ which selects a sensor to remove. This heuristic function has

a huge impact on the algorithm. In the basis version of the algorithm $select(\vec{s}) \mapsto s_b$ finds the sensor with the lowest amount of features which is larger than zero, as we mark removed sensors with a 0 in the array $\vec{s}$. $select(\vec{s}) \mapsto s_b \quad s_b = \min(\vec{s} > 0)$. We use the function $eliminate(s_b, f_{\text{map}}, D') \mapsto D''$ to remove the features of sensor s_b from the data matrix D'.

Also let C be a set of classifiers c suitable for the task, $findbest(C, D', y, D''_v, y_v) \mapsto c, a_c$ be the function to find the best classifier given the current configuration and a_c be the corresponding accuracy at the validation set. Let $rfe(D', y, \text{model}) \mapsto D''$ be a recursive feature elimination using a model which removes not important features for the classification of the variable y from D' and produces a new feature matrix D''. The model can be anything which gives a ranking of features. We choose the Lasso regression with its ℓ_1 regularisation as it directly tries to remove the features as part of the optimisation algorithm by optimising the cost function $\sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right)\right)\right) + \alpha * \sum_{i=1}^{m} |\mathbf{w}_i|$ (where $\sum_{i=1}^{m} |\mathbf{w}_i|$ is the ℓ_1 penalty) where y is a dependent variable of binary value, and several one vs. rest classifiers need to be trained.

As a result, we want a list of sensor configurations together with their accuracies. One element of this list is the tuple $\vec{s}_{\text{b}}, \text{acc}, c$ with $\vec{s}_{\text{b}}$ being a binary vector indicating if a sensor is used or not, acc being an accuracy measure (i.e. f_1 in our case) and c being the classifier which achieves the accuracy for that configuration. We call this list L_c. The basis RSE now works the following way:

Algorithm 1 Calculate L_c with $RSE(D', y, D'_v, y_v, C, \emptyset)$

if num of features in $D' = 0$ **then**
  exit
end if
$\vec{s}, f_{\text{map}} = fts(D')$ {map features to sensors}
$s_b = select(\vec{s})$ {select sensor to remove}
$D'' = eliminate(s_b, f_{\text{map}}, D')$ {remove features of sensor s_b from feature matrix}
$D'' = eliminate(s_b, f_{\text{map}}, D')$ {remove features of sensor s_b from feature matrix}
$c = findbest(C, D'', y, D'_v, y_v)$ {remove features of sensor s_b from feature matrix}
$\vec{s}_{\text{b}} = \vec{s} \neq 0$ {binarieze $\vec{s}$}
put $\vec{s}_{\text{b}}, a_c, c$ into L_c
$D''' = rfe(D'', y, \ell_1)$
if num of sensors not removed in $\vec{s} = 0$ **then**
  exit
end if
$RSE(D''', y, D'_v, y_v, C, L_c)$ {recursively call RSE}

For our function $fts(D') \mapsto \vec{s}, f_{\text{map}}$ we traced which feature belongs to which sensor and part of the hand by encoding it in the feature header. During feature extraction, whenever a feature is generated, a header is also generated in the form $x_{o.c}\_x_f\_o\_s$ with x_{oc} being the original channel number, f_f being a feature index in the 59 features we provide, o the original header and s a suffix describing the new feature. An example of such a generated feature header would be `16_14_Wrist_extension_fft_min`, which is the feature from channel 16, which is the wrist extension flexion sensor. The feature is transformation 14, which is the minimum amplitude of the FFT coefficients. Because of this coding scheme, after features are removed, it is possible to map back the remaining features to the original channels and sensors.

In this thesis we compare three possible choices for a heuristic function $select(\vec{s})$. Choice (a) is selecting the sensor from $\vec{s}$ with the lowest amount of features. Some sensors create more features than others. An IMU with 6 *DOF* generates six times as many features as the flexion sensor. Therefore a choice (b) for the function $select(\vec{s})$ is to normalise each number of features by the original number a sensor can spawn and then remove the sensor with the lowest relative amount of features. We call these two heuristics "abs" for absolute and "rel" for relative removal.

4.8.2. Recursive Sensor Elimination Results for all Gestures

We perform sensor selection with some of the datasets produced by the recursive feature elimination (1910, 765, 192). On each step of the sensor selection, we save away the resulting configuration of sensors, the number of features and the best performing classifier with the accuracy on the validation set. Note that for two of these configurations there is already some sensor removed by feature selection alone.

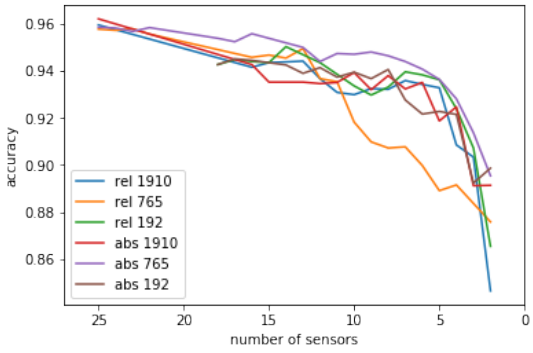


Figure 4.23.: Performance of the *RSE* algorithm on different datasets and $select(\vec{s})$ heursitics. The accuracy is the f_1 score on the validation set.

The 765 dataset has both, the best and worst evolution of accuracy, depending on the strategy. The 1910 and 192 both behave very similarly. The trends in accuracy for the different datasets give the interesting observation that the number of features to start with does not have a high impact on the performance of the *RSE* algorithm.

When comparing "abs" vs "rel" on the datasets we can see that the absolute strategy works better from an accuracy point of view, especially in the middle segment with 15 to 4 sensors. They do however remove different sensors. In all cases, the magnetometer and pressure sensors are removed first. Then "abs" favours IMUs over flexion sensors, while "rel" favours the reverse. For an engineer, both might make sense.

When the algorithm runs a list L_c is generated at each iteration, containing a configuration for the model. For an engineer, it makes sense not only to see a resulting accuracy and number of sensors but the particular configurations. We created an example visualisation for this in table 4.9 showing the configurations in consecutive steps of five iterations.

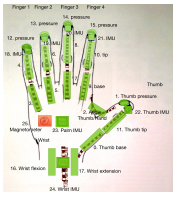
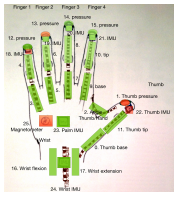
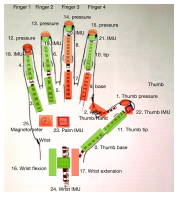
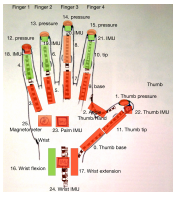
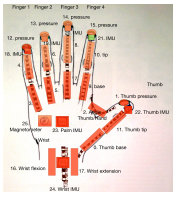
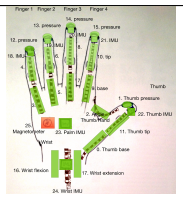
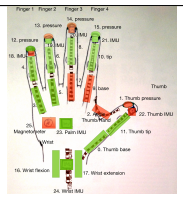
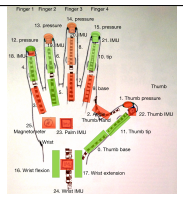
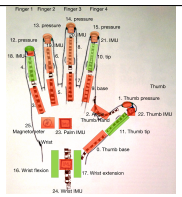
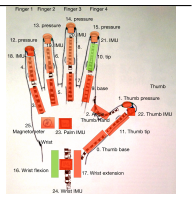
A765															
s	#f	#S		#f	#S		#f	#S		#f	#S		#f	#S	
1	764	25	5	300	25	10	75	12	15	50	6	20	17	1		
M	$f1_v$		M	$f1_v$		M	$f1_v$		M	$f1_v$		M	$f1_v$			
ET	.96		QDA	.96		RF	.96	.99	ET	.94		RF	.90			
R765															
s	#f	#S		#f	#S		#f	#S		#f	#S		#f	#S	
1	764	25	5	69	17	10	44	11	15	18	6	20	7	2		
M	$f1_v$		M	$f1_v$		M	$f1_v$		M	$f1_v$		M	$f1_v$			
RF	.96		RF	.95		ET	.93		RF	.90		RF	.87			

Table 4.9.: Evolution of the sensor selection starting with the 765 RFE-CV dataset. A765 corresponds to the *absolute* algorithm and R765 to the *relative*. *s*: step number, #f: number of features, #S: number of sensors, M:model,$f1_v$: F1 in the validation set.

We can make the following observations: The index finger is the most important finger for predicting gestures. If the algorithm retains flexion, it keeps the whole index finger (in the relative case). Since many gestures need a moving part and the flexion of the index finger can not capture that, the algorithm keeps one of the wrist flexion sensors to have information about that. If it keeps the IMUs (absolute case) it also keeps the IMU of the index finger. The magnetometer is always the first sensor removed. Then most of the pressure sensors are removed. Pressure sensors are never within the last remaining sensors. If flexion or IMUs are removed depends on the configuration of the algorithm. While it seems logical that you do a relative weighting of the features compared to the number of features per sensor, with all features the results were better with the absolute feature removal. In every step, the relative feature removal removes more or the same amount of features as the absolute.

4.8.3. Usecase: Development of a Task-Specific Gesture Glove

We started the glove experiment with an alphabet of natural, interaction-oriented gestures in mind. For a specific use case often only a subset of these gestures is needed. In Luzhnica and Veas [2018], three gestures from the gesture glove are examined for controlling an interface to read text over the skin of a hand (skin reading). The gestures are swipe left, swipe right and swipe up to navigate between words and allow repetition. An own glove for skin reading is developed, and we can give an answer on how to augment the glove with hardware best to allow this three gesture control.

Having three of the four usual swipe gestures, it makes sense to examine the configurations for a glove detecting all four swipe gestures. An example would be to control any music player to start (up) or stop (down) the music playback and to advance to the next or previous song.

Another interesting use case is controlling a slide set with gestures. The requirements are to navigate the next and previous slide, pause the presentation and resume, and to start and stop interactive elements within the slide like videos. We can use the swipe left and right for navigation, the down and up gesture to pause and resume the presentation, and then two more gestures to start and stop interactive elements. We did not find any immediate natural gesture for this interaction, but some seem logical. Thumbs up and thumbs down, and go away and come here both seem like good choices. We have the hypothesis that the swipe gestures are better recognised with IMUs than with flexion sensors. Thumbs up and down is also a gesture which seems more defined by IMUs in comparison to go away and come here, so a decision between these gestures could lead to different costs for a given accuracy.

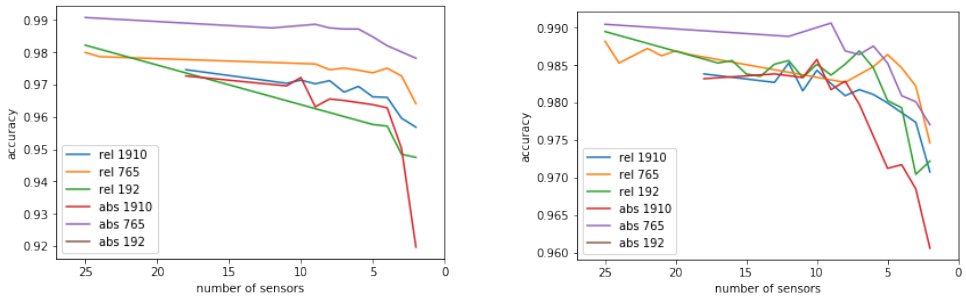
In total, we have four sets of gestures for use cases. The first (skin reading) and second (audio player control) are both subsets of the third and fourth set, which are two alternatives to compare among. We denote the sets as s_1, s_2, s_3 and s_4.

4.8.4. Results on Task Specific Gesture Sets

The results of *RSE* on our gesture sets for the use cases s_1, s_2, s_3 and s_4 are shown in figure 4.27. We can see that the f_1 scores for the subsets of gestures are all higher than for all 31 gestures. This not surprising, as the task to differentiate 3, 4 or 6 relative distinct gestures is easier. In all cases, accuracies are around $97-99\%$ as long as there are more than 10 sensors. Also in all cases, as also in the case of all 31 gestures, the accuracy only decreased slightly until the tipping point of around $7-5$ sensors. This tipping point means that an optimal configuration seems to have more than $7-5$ sensors.

4.9. Results on the Test Set

Our final test set consists of one unseen user. We compute the results for this user for the winning combinations of all the variations of feature size and selected sensors of the data-driven analysis. For all experiments with all 31 gestures, there exist a systematic error of confusing the "One (1)" gesture with the zero class and vice versa. An example of this systematic error is seen in the confusion matrix in figure 4.25. If one



(a) Results from set s_1

(b) Results from set s_2

(c) Results from set s_3

(d) Results from set s_4

Figure 4.24.: Different results of *RSE* for subsets of gestures.

ignores this particular error, the accuracies on the test set are in a similar range to the validation set since the zero class is not confused with other gestures than the "One (1)" gesture. This effect can be seen if we use the macro average metric for multi-class classification. In macro average, every class has the same importance, and so one systematic error does not have a significant impact. In the micro average case the relative amount of the class is used. Since the zero class and the "One (1)" gesture are switched, this completely dominates the result, and the system degenerates. A summary of all the results with different feature sizes can be seen in table 4.10.

	macro avarage			micro avarage		
dataset	precision	recall	f1-score	precision	recall	f1-score
6368 full	0.93	0.94	0.91	0.37	0.37	0.37
3.821 f-ANOVA	0.93	0.94	0.92	0.37	0.37	0.37
3.821 MI	0.93	0.94	0.92	0.37	0.37	0.37
RFE/1910 $\alpha = 0.1$	0.93	0.94	0.92	0.37	0.37	0.37
RFE-CV/765 $\alpha = 0.01$	0.93	0.94	0.92	0.37	0.37	0.37
RFE-CV/383 $\alpha = 0.1$	0.92	0.93	0.91	0.37	0.37	0.37
RFE-CV/192 $\alpha = 0.5$	0.92	0.93	0.90	0.37	0.37	0.37
RFE-CV/192 $\alpha = 0.9$	0.91	0.91	0.89	0.36	0.36	0.36

Table 4.10.: Overview of the accuracies on the test set of the best classifiers on different feature sizes.

The same behaviour can be observed with the sensor selection over all 31 gestures. When using the macro average, the results are in line with the validation set result. But the same systematic error results in a worse micro average result. The curve of the two heuristics over the different sensor configurations is in figure 4.26 (a).

A more problematic result is the evaluation on the test set of the sensor selection with only a subset of gestures. We can see the results in figure 4.26 (b). It exhibits similar behaviour than the other models on

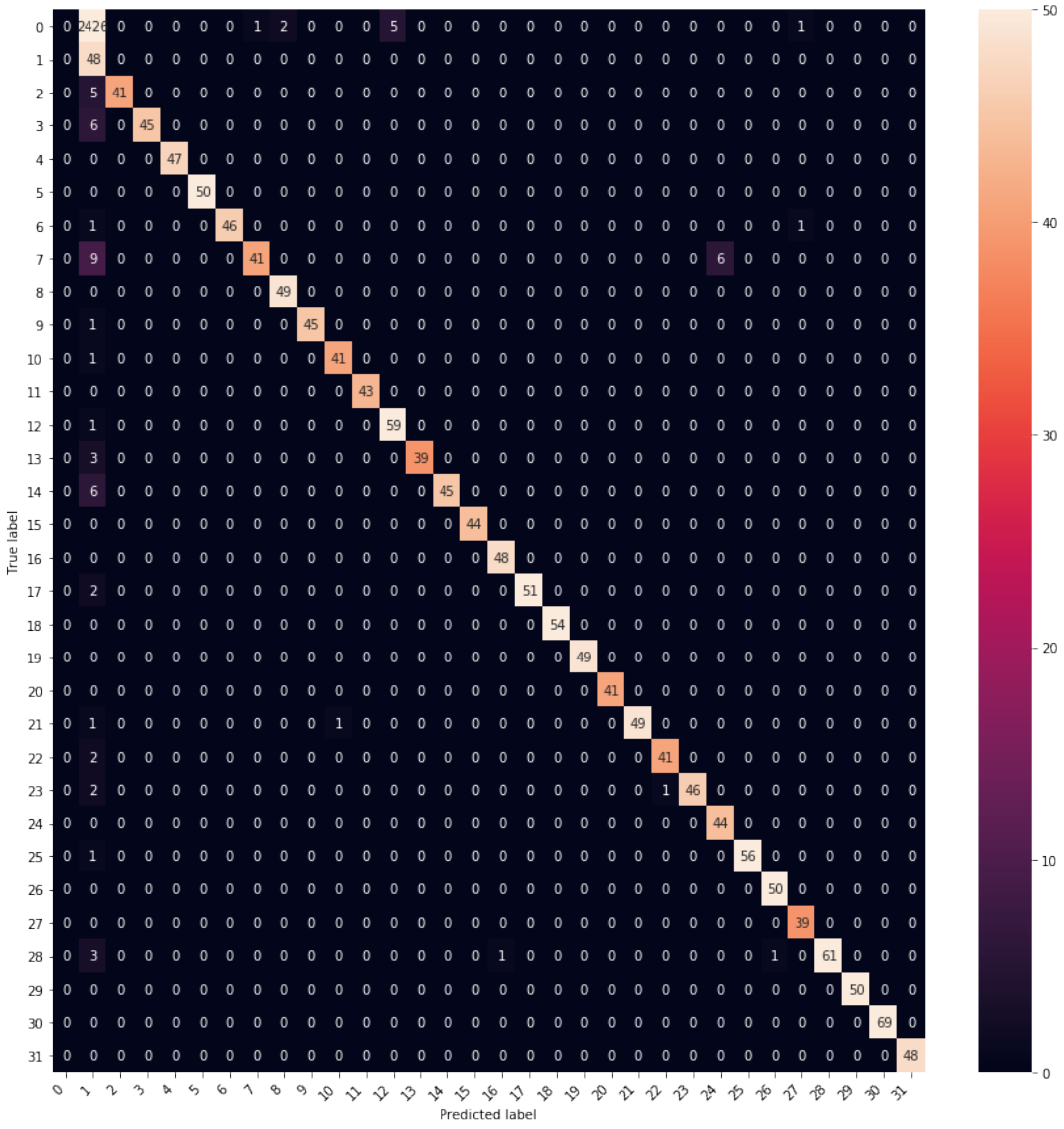
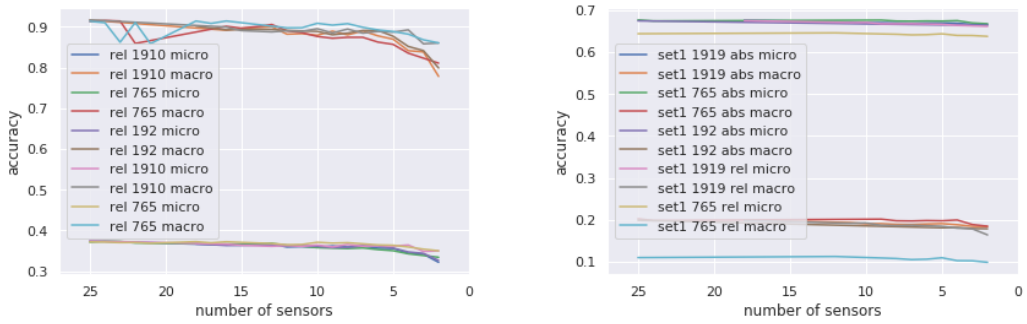


Figure 4.25.: Confusion matrix of the extra tree for the full dataset. We can see a systematic error of misclassification of the gesture “One (1)” and the zero class.

the test set. However, this is not the reason of one confusion/one systematic error anymore. In this case, the zero class is even more dominant, and the system confuses all classes with the zero class. The macro and micro accuracy drop, with macro only being in the range of $60-70\%$ anymore compared with over 95% on validation. This does not invalidate the general algorithm of the *RSE* but shows that we need a different strategy in modelling in general for these subsets which are working well against the zero class.



(a) Chart of the results of the sensor selection algorithm for all 31 gestures on the test set for both, macro and micro averaging the f_1 score.

(b) Chart of the results of the sensor selection algorithm for the gesture set s_1 on the test set for both, macro and micro averaging the f_1 score.

Figure 4.26.

4.10. Conclusion

In this thesis, the hardware setup and data-collection software of the smartglove is described in depth to provide a future reference. Also, the loading of the dataset and explanation of the raw data, the effect of data cleaning and outlier removal is explained in detail. All of these aspects help, if a person wants to explore this work further.

We split the data into a different training, validation and test set which is more theoretically sound than in our previous work in Luzhnica et al. [2016]. We can show that on the validation set similar accuracies and results are achieved as in the randomly subsampled validation set in the previous work, and with that validate the old strategy.

From theory, we found that about 300 features ($\sqrt{70.000}$) would be a good number for our dataset. We explored with different algorithmic methods the space of 6.368 to 192 and showed that on average classifiers perform best in the region of $765-383$ which is a bit higher than the theoretical number of features but in range.

On the other hand, we show that mainly variance reducing methods like random forests, extra trees and bagging, in general, perform well on large datasets and are not negatively impacted by feature sets over 765 features. Both ensemble methods, extra trees and random forests are our best performing classifiers in all experiments.

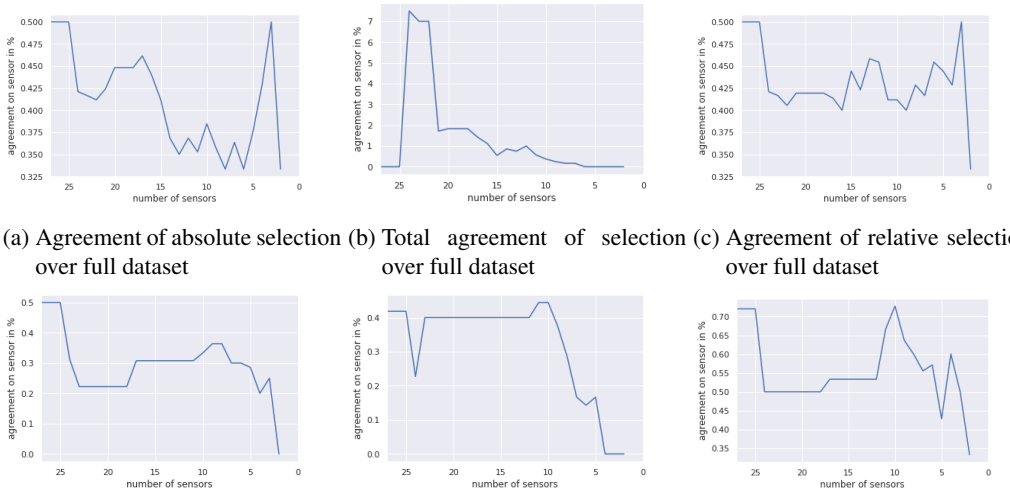
In future work it would be interesting to do a more in-depth analysis of the resulting features. We do this to some extent for the smartwatch experiment. However, visualising the features after PCA, LDA or especially t-SNE after each transformation might shed some additional light on the experiments presented in this work.

The results for the classifiers on the test set with an own user show a strange error. For 30 classes the performance in the system is splendid. However, the classifiers systematically confuse the "One (1)" gesture with the zero class, bringing the micro-average, which incorporates class weights, to only 37%. The

reason for the confusion can be a not yet found error in the scripts or the training strategy and missing dual labelling. Andrew Ng states in his great machine learning course that the distributions of validation set and test set should be the same. In our case the validation set is balanced, and the test set not. Even more the applied strategy on modelling gets problem if the zero class get's very dominant in case of the use case specific gestures subsets s_1, s_2, s_3 and s_4, where the accuracy on the test set drops dramatically. Training the sensor selection without partial windows would be a first next step to investigate into this problem.

Instead of training only with windows with the full gesture or with all partial gestures it would be interesting to see the effect of thresholding the partial windows (f.e. include into training all with more than 50% of a gesture) and dual labelling.

In our work in Luzhnica et al. [2016] additional error analysis is done, especially on systematic confusions. It would be an exciting future work to do this analysis for the different configurations of feature sets. While with feature and sensor selection we brought the computational cost in some cases into a lower amount than in the paper, we did not discuss computational costs in detail in this thesis.



(a) Agreement of absolute selection over full dataset

(b) Total agreement of selection over full dataset

(c) Agreement of relative selection over full dataset

(d) Agreement of absolute selection over gestures s_1

(e) Total agreement of selection over gestures s_1

(f) Agreement of relative selection over gestures s_1

Figure 4.27.: Different agreements of the *RSE* algorithm with the absolute selection heuristic on the left, the relative in the right, and the total agreement in the middle.

With *RSE* we present a novel algorithm for sensor selection. While the basic algorithm is shown in this thesis several dimensions to explore the idea further exist.

Incorporating costs into the $select(\vec{s})$ heuristic allows to better model the intent to optimise the system towards a specific use case. Costs can be hardware costs, costs for building the system, assumed maintenance costs or even energy costs. In some case having more sensors increases the costs by additional control electronics, in others single sensors might be more expensive. This means that costs can become highly non-convex but more interesting to model.

An important question is the quality of the *RSE* algorithm. One proposal is to use the agreement of one particular version of the algorithm over different datasets as a quality criterion. Agreement is defined by how often the algorithm selects the same sensor along with the slightly different feature sets. First experiments of this effect are shown in figure 4.27 and might hint that the relative algorithm is of higher quality in that sense than the absolute algorithm. Measuring not the sensor, but the agreement on where to place the sensor (f.e. on the index finger) might help shed light on the importance of the placement of sensors vs the sensors type. Further exploration is needed in this area.

The *RSE* uses repetitive application of the ℓ_1 regulariser in the form of Lasso. This choice means we assume a linear dependency between the features. In our work, however, we show that bagging trees are the best methods, which does not imply a linear dependency. RFE has already been extended to not only work with Lasso or SVMs but also with random forests or LDA. It would be interesting, if this affects our algorithm.

Additionally to that, *RSE* is very aggressive at the moment. If in one iteration RFE eliminates one sensor by feature removal alone, *RSE* additionally removes the weakest sensor. Also, the number of features is reduced by both methods, RFE and sensor removal, in each step. This potential double removal might mean we go too fast towards the assumed minimum. Some extensions might make the algorithm more stable:

Only removing a sensor when the $select(\vec{s})$ of RFE did not yet select one, or the other way around, always stopping RFE before all features of a sensor would be selected, and then use $select(\vec{s})$.

Also, it would make sense to make sensor removal the only driver for changing the feature dimension at first and always start with the full feature set without the features of the removed sensor in each iteration.

Instead of looking at the number of remaining absolute or relative features (weighted by some cost), it might be interesting to sum the remaining importance of the features from the model as a heuristic for removal.

For linear dependencies, one could envision that all features spawned by a single sensor are additionally multiplied by a sensor weight. If this weight is used with the ℓ_1 regularisation, this would lead to sensor selection while optimising the cost function. Performing the selection while training it would allow models where training is a heavy burden like DeepLearning to benefit from the sensor selection.

Given the positive results with two sensors and six gestures for controlling a slide set, it is interesting if such a system is possible on a smartwatch. This is explored in the following experiment.

Chapter 5

The SmartWatch Experiment

The next experiment in this thesis is one with a smartwatch. The aim is to reuse the results from the feature engineering and modelling with the smartglove for a smartwatch. One might ask the question, what can you do with the sensor of a more common wearable, namely the one 6 *DOF* IMU[1] of a smartwatch?

The idea was to create a simple system with four gestures used to steer f.e. a slide show. We choose to use the gestures "slide left" and "slide right" to map to the next and previous slide action. "Thumbs down" pauses a presentation, and "thumbs up" resumes a paused presentation.

5.1. The Hardware and Data Collection Software

The Apple Watch Series 3 has a 6 *DOF* accelerometer. The very thin MEMS sensor is produced by STMicroelectronics and later by Bosh. The details are not disclosed to the public, and as a developer you only interface with the `CoreMotion` API. On the positive side, this API already does Sensor Fusion for you and gives values in the right SI system to the developer.

For experimenting, we did write a labelling software. An Apple Watch Extension starts and stops the recording of the sensor. For debugging it also shows the accelerometer and heading values. The interface is shown in Figure 5.1.

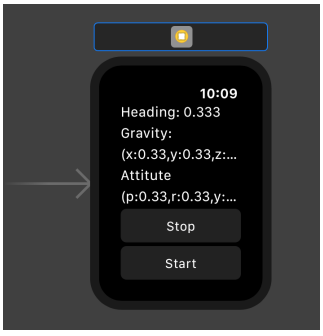


Figure 5.1.: Apple Watch Extension for sending sensor data.

The problem here is that Apple does not allow continuous sensing per default. Only workout apps are allowed to do that. So the App starts a workout whenever you hit record, and stops it in the `RecordingManager` class. In the background, it collects `CoreMotion` data from the watch. In the `MotionManager` class we manually fix the position of the watch to be on the left arm. We then start a recording of the IMU data with basic sensor fusion. While the API has an option to fix the reference frame into world coordinates, it can only do this if a magnetometer sensor is available. Because of this, this option is only available on the iPhone. We set the recording to 50 Hz.

The Apple Watch restricts the communication of the watch to the iPhone to the `WatchKit` (`WC`) API. While

[1] actually most smartwatches have no magnetometer inside, including the Apple watch, so only 6 *DOF* exist.

not explicitly mentioned, this API seems to only stream constantly if the Extension is active, which means the screen of the watch is facing up and is lighted. Else, from our experiments, it does only occasionally send a package. Because of that, we collect the `CoreMotion` data in a queue and whenever we can we send the data to the iPhone. For analysis this does not create timing problems as the timestamp of `CoreMotion` objects is generated by a hardware clock when the sensor is read. However, the timestamp is in nanoseconds from the last restart of the Apple Watch. To synchronise with the iPhone, we therefore send the iPhone's time on the first roundtrip and compute an offset. The `CoreMotion` objects themselves are not serialisable, so we extract a row out of each of them and send a batch of data in the form of a matrix. By sending quantities of data as a matrix, we also space when transmitting the data.

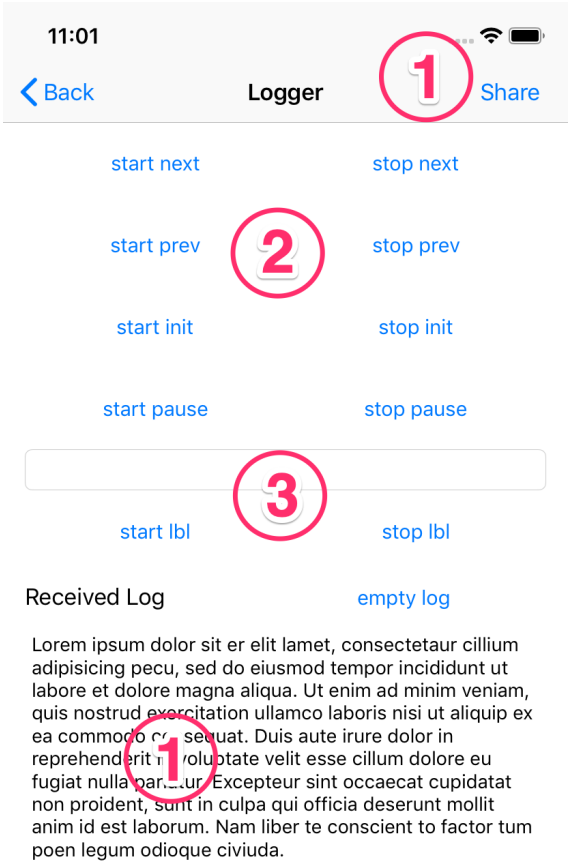


Figure 5.2.: Screen of the labelling function of the application.

On the iPhone, an iOs application acts as a server for receiving the data. It can do two things with the data: First, it can collect it and add labels to it. Second, it can run inference and send the detections to a

server. We will now explain the collection part of the application.

The interface of the labelling app is quite simple and can be seen in figure 5.2. An area ((1) in the figure) displays the received data directly as text/matrix. This area acts as a way to debug if you are correctly receiving data. In the upper area of the application ((2) in the figure) buttons for annotations are present. Starting and stopping an annotation is done with these buttons. If an annotation is started the timestamp of the button press is recorded with the annotation at device time. Because we also record the offset between the two devices we can later recover the temporal relationship. It is possible to annotate anything with a text entry ((3) in the figure). When enough data is collected, one can use the functionality of a `UIActivityViewController` to share data in the form of a CSV file. We found this a practical and easy way, as it works with large files over the wifi. F.e. it finds your laptop over AirDrop, and you can from there use the files for further modelling.

We receive the collected data on the laptop and use scikit-learn and python to do the modelling. We then convert the models in `CoreML` models which we deploy back into the application. We have a debug mode where the application makes inference on the received app data and sends the data to a server. We go into these parts in the later sections of this chapter.

5.2. Experiment / Data Collection

The dataset is collected from one person only. While the results from the smartglove hint that such systems work among many people, nothing can be said about how well the system would generalise to different persons.

We recorded sessions with me performing the four gestures “slide left”, “slide right”, “thumbs up” and “thumbs down”. In contrast to the experiment we did with the glove I was standing most of the time and gesticulating while making a fictional presentation, and performing the gestures. I did hold the labelling app in my other hand so this might bias the results a bit. I also had one session where I was sitting, and one where I was going around. These are all influences we did not have in the gesture glove study but are present in the real world use. In total, I recorded 45 minutes 15 seconds of data.

5.3. Data

Since there are fewer channels, it is also easier to visualise the data. The idea to look for templates in the raw time series was developed in this experiment and just then applied for the smartglove. We also needed that to find and fix potential timing errors, since the clock of the Apple watch is relative to its last reboot and because of the unstable scheduling of the transmission.

Because of initial problems with data collection (f.e. the nanoseconds timestamp format of the Apple watch) a lot of data was invalid. The visualisation techniques used here were used to remove invalid data from later feature engineering and modelling. In the end, we had 135.475 datapoints with 988 labels with valid data.

Figure 5.3 shows data from one session of recording where I repeatedly performed all four gestures. Since there is no magnetometer on the watch, the magnetometer channel is always just a constant value. Also since the watch cannot fix the coordinate reference space, there is no heading. The small lines at the bottom of a plot are always a labelled section.

It seems that, if we stop the recording app first and then the watch extension (sometimes even if we do stop the watch extension first), there are leftovers on the next recording session. It seems that the watch has its own store for sensor data which is sometimes not flushed. A picture of that effect in the appendix in figure B.1.

Figure 5.3.: Data from one recording session.

If we look into detail of a gesture like f.e. the “swipe right” gesture (see figure 5.4 and figure 5.5) we can see patterns that seem relative distinct. Still, similarish patterns exist while gesticulation in speaking as we see in figure 5.3. It is interesting to see if the feature engineering can pick this information up. For tuning the window size, we again looked at how much time a gesture takes. I need on average 1.999705s for the “thumbs up”, 2.116329s for the “thumbs down”, 1.980159 for the “swipe right” and 2.052116s for the “swipe left”. Given a sampling rate of 50Hz, this is about 100 records as a minimum sliding window length to capture the whole gesture. We decided to use a sliding window of 128 frames per window and a step size of 20 frames (so a step every 0.4 seconds or in a live system a decision every 0.4 seconds). The decision of 128 is also somewhat a technical one. The implementation of the fast FFT on iOS can only handle power of two sized data arrays, so it is a decision between using a window size of 128 or one of 256 timesteps.

5.4. Data Augmentation

I only recorded 45min of data. Transforming that to windows every 0.4 seconds only yields about 6000 data points, which is not a lot, if you also make at least a validation set out of it (we do not make a test set as this is meant to be a live system, so the test is meant to be a real test).

A common praxis from DeepLearning is to use data augmentation. Data augmentation is any transformation of the data which produces different data but does not alter the label. In images these are transformations like flipping the image, changing the brightness or similar. The transformations which are allowed always have to do with the domain. F.e. if you flip the image of a cat horizontally it is still a natural appearing cat, if you flip it vertically, while it is still a cat, it is not a natural appearing shape (unless the cat falls). An example of data augmentation for a cat can be seen in figure 5.6.

Camps et al. [2018] discusses the problem that this is not necessarily easy with sequence data. Their data augmentation consists of a random shift and 3D rotation in space. They associate a probability of how dirty that signal is and learn that together with the labels. So what are transformations of IMU data that do not hurt the labels? Our observation is that making a gesture faster or slower within some limits it is still the same gesture (a fast “thumbs up” and a slow “thumbs up” is still all a “thumbs up”). Another transformation is to add a tiny bit of additional Gaussian noise to the signal. All sensors are noisy and adding a bit might help the algorithm to not overfit on that.

If we change the speed of a gesture, the easiest augmentation is to divide the values among the new pins evenly. For good data augmentation, you want arbitrary speed changes, so you should allow making something f.e. 1.124 times as fast. To allow that we made an algorithm based on the following idea: We want to change the speed, but not the path travelled, so having an original signal x and an augmented signal $\hat{x}$ the constraint holds that $x, \hat{x}$ subject to $\int\int x = \int\int \hat{x}$.

Augmenting the gesture by slowing it down: For making it f.e. 1.3 times slower you need 1.3 times more values rounded up. Then each value must be divided by 1.3 and then distributed over 1.3 fields. That is, taking the full value divided by 1.3 for one field, and 0.3 of the value divided by 1.3 for the next field. That field is then filled with 0.7 of the next value divided by 1.3 and so on. There is a reduction factor for each value and a distribution factor which is used to distribute the value over the array. A pseudocode implementation can be seen in algorithm 2.

Augmenting the gesture by speeding it up: For speeding up, fewer pins in the signal need to be used. Into these pins, you sum up a fraction of the next pins. If you would like to make the signal 1.3x faster, you would sum into the first pin the full value of the first pin of the source signal and 0.3 of the second value. Then you would take 0.7x the second value of the source signal for the second pin and sum 0.3x the third value of the source to it, and go on analogous. Algorithm 3 show a pseudocode implementation of that.
In figure 5.7 we show a test of transforming some artificial signal with the suggested algorithm. A ran-

Figure 5.4.: Some zoomed in data of the next gesture.

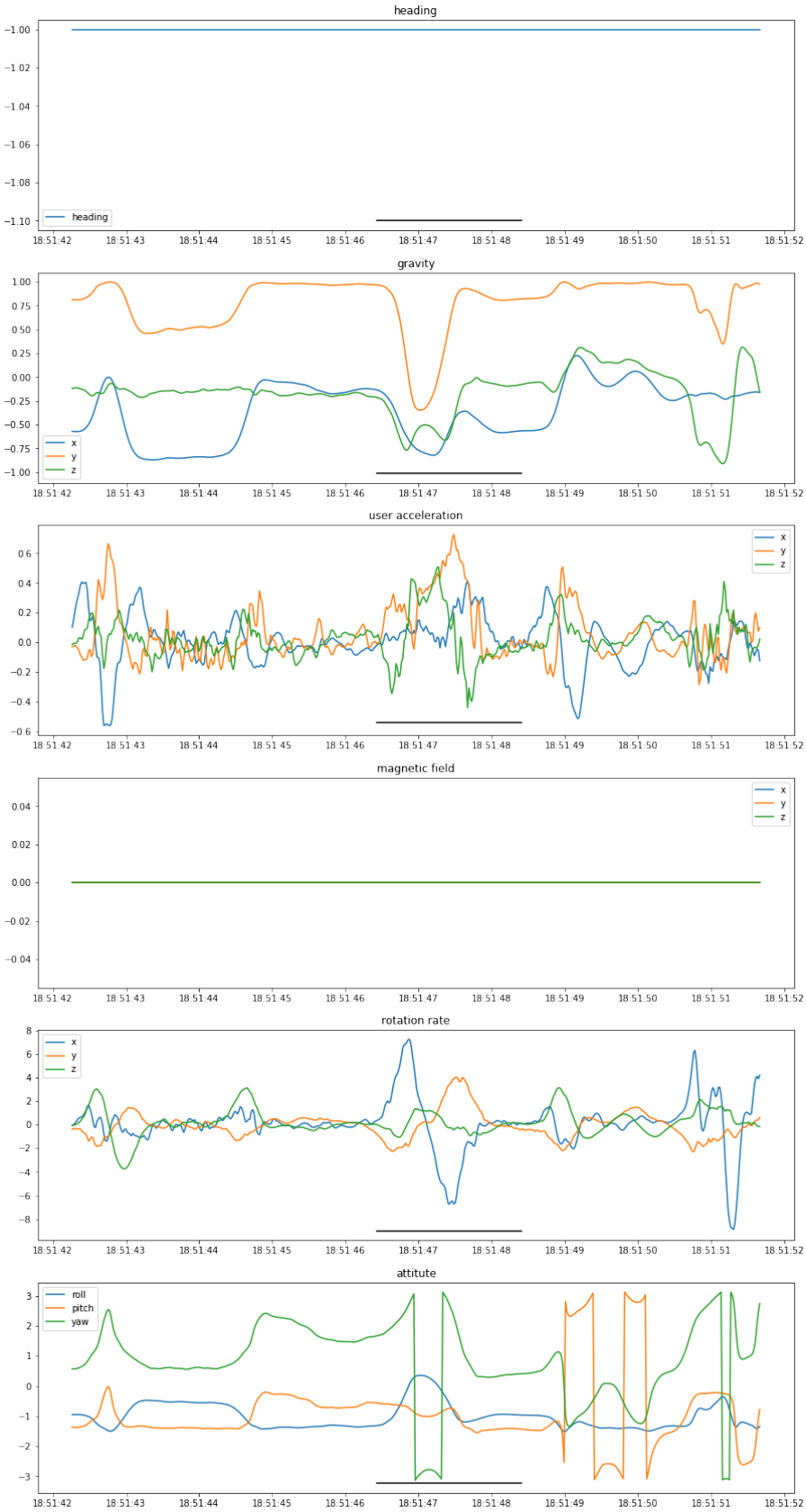


Figure 5.5.: One next gesture in detail.

Figure 5.6.: Data augmentation of a kitten. These random transformations still allow the image to be clearly identified as an image of a kitten. Image taken from the fast.ai documentation.

Algorithm 2 Calculate $\hat{x} = \text{slowdown}(x)$

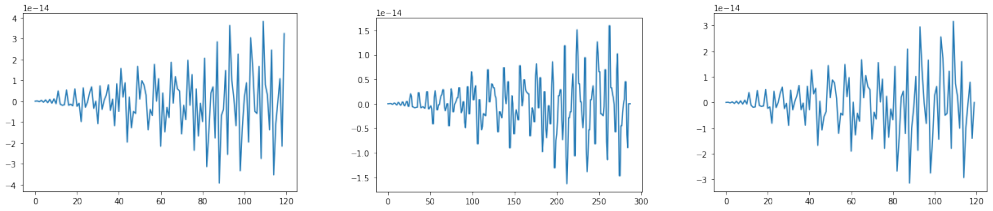
Require: $nf =$ new frequency in herz (f.e. 50), $of =$ old frequency in herz, $s =$ current signal
  $rf \leftarrow nf/of$ {reduction factor}
  $df \leftarrow nf/of$ {init distribution factor}
  $ns \leftarrow$ np.zeros(len$(s) * \lceil nf/of \rceil$)
  $si, di \leftarrow 0$
  while $si <$ len$(s) - 1$ **do**
    if $df > 0$ **then**
      $df \leftarrow df - 1$
      $ns(di) = ns(di) + s(si)/rf$
      $di \leftarrow di + 1$
    else
      $ns(di) = ns(di) + (s(si)/rf) * df$
      $ns(di) = ns(di) + (s(si+1)/rf) * (1 - df)$
      $di \leftarrow di + 1$
      $si \leftarrow si + 1$
      $df \leftarrow rf - (1 - df)$
    end if
  end while

Algorithm 3 Calculate $\hat{x} = \text{speedup}(x)$

Require: nf = new frequency in herz (f.e. 50), of = old frequency in herz, s = current signal
  $sf \leftarrow of/nf$ {sum fraction}
  $ns \leftarrow$ np.zeros(len$(s) * \lceil nf/of \rceil$)
  $si, di \leftarrow 0$
  while $di <$ len$(ns) - 1$ **do**
    if $sf > 0$ **then**
      $sf \leftarrow sf - 1$
      $ns(di) = ns(di) + s(si)$
      $si \leftarrow si + 1$
    else
      $ns(di) = ns(di) + s(si) * sf$
      $ns(di+1) = ns(di+1) + s(si) * (1 - sf)$
      $di \leftarrow di + 1$
      $si \leftarrow si + 1$
      $sf \leftarrow (of/nf) - (1 - sf)$
    end if
  end while

domish signal (signal $= \sin \pi * [1...n]$ is stretched by 2.1 times and transformed back.

Adding noise to a signal is relatively easy. The only art is to estimate the σ parameter of the noise. From the visualisation and the value ranges of the data we can assume that a σ of 0.1 will not hurt the signals.



(a) Original signal over 120 time points (b) Signal streched to 288 time points (c) Signal transformed back

Figure 5.7.: Example of the timing augmentation algorithm.

5.5. Feature Engineering

We used a similar feature engineering to the smartglove. Since there is no linear algebra library like numpy, all the math for the transformations are written by hand in swift. Currently, the swift team works hard on math optimisations so this small library might be obsolete soon.

Since we needed to implement many things by hand, we skipped the features which we found too much effort to implement and test. These features to drop are the wavelet transform and the wavelet-based peak detection. Also, the FFT implementation of the accelerate framework of iOS is used. The accelerate framework can only handle certain window sizes for the FFT transformation (windows with a size which is the power of 2). We kept all the statistical features and the FFT features (statistics, spectral centroid, spectral entropy, first five coefficients and the sum of them and the bandwidth of the frequency) and dropped the CWT and the peak features.

We visualised the transformed features by stacking the windows after each other in a time series and

colouring the line according to the label of the window. Since we generated over 300 images, this would be too much even for the appendix. We must ask the reader to check out the github repository for that and he might need to re-run the experiments. However two examples of a visualisation of the transformation are shown in figure 5.8.

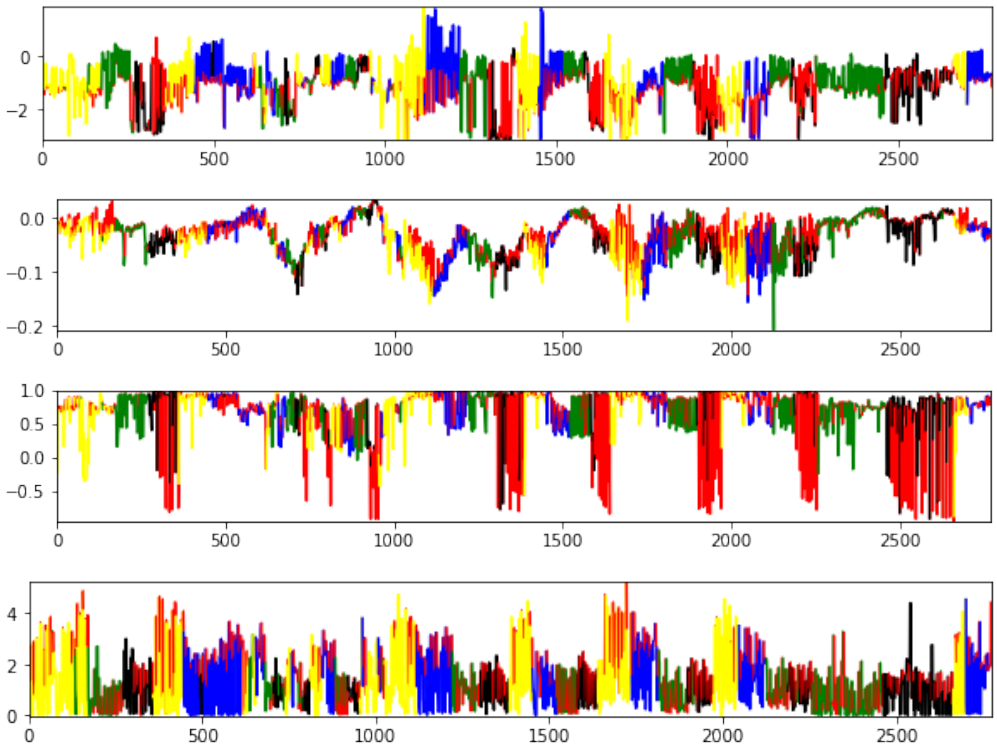


Figure 5.8.: Visualisation of the feature transformations per feature. X axis is the sliding windows which are in temporal relationship. The colour of the line corresponds to the label at that window point. Red is the zero class.

In total 357 features are generated from the IMU. While the IMU is a 6 *DOF* sensor, bear in mind that Apple already did a bit of the feature engineering in the form of digital signal processing for us: Besides the six raw channels we also get the linear acceleration and the roll pitch and yaw in a reference frame relative to the first reading of the watch. All these channels are used as input to generate the features.

As a basis we have 135.475 instances with 988 instances being labelled. Using a sliding window of 128 frames (2.46s) and a stride of 48 frames (a bit less than a second) this results in 2774 windows to analyse. However, since the lowest class has 198 windows, we subsample the other classes to the same amount with cluster centroids subsampling [Zhang and Mani, 2003]. So the final dataset has a dimension of $(198 * 5) \times 357$.

5.6. Models

The models for the smartwatch are validated with cross-validation over random subsampling of the windows. One reason is that the dataset is very small. There is no test set, as the performance of the system is tested by running it in a demo mode.

For training the models, another new aspect comes into play: Not all models are possible to be transferred to `CoreML`. F.e. the best classifier on the smartglove was extra trees, the winner in Luzhnica et al.

[2016] was Logistic Regression. Both are not available in `CoreML`. You would need to re-implement the algorithm for iOS to use them (at least the inference part) and transfer the weights by hand. Re-implementing the inference part is too much effort, so we did not do that, but constrain us to algorithms which can be transferred to `CoreML`. On the other hand, since the data set is way smaller we do more parameter tuning, especially for the SVM which can achieve good performances with the correct set of parameters.

We grid search the different multiclass decision boundaries (one vs one, one vs rest) the different kernels (RBF, polynomial, linear sigmoid) and its hyperparameters ($\gamma \in [0.001, 100], poly \in [2,4], c \in [0.001, 100]$ or $nu \in [1e{-}100, 1]$). In total, we searched 239 different configurations for the SVM. SVC worked better than NuSVM or LinearSVM. The most important parameter for SVC is to allow high contamination with the parameter c. While all different kernels and decision functions for SVC appear in the top 20 best performing SVMs, all had a c parameter over 50. A table with the parameters of the best 20 SVM configurations sorted by validation accuracy is in the appendix, table B.1.

While the default parameters most of the time performed badly, a tuned SVM can achieve accuracies with over 95% on the validation set. In the end we choose an SVC with the following parameters: C=50.004999999999995, class_weight='balanced', decision_function_shape='ovo', gamma='auto', kernel='rbf', shrinking=True. It gives 99% accuracy when used on all the data. The details are in table 5.1.

	precision	recall	f1-score	support
1	1.00	0.96	0.98	207
2	0.98	0.99	0.99	197
3	0.98	1.00	0.99	189
4	1.00	1.00	1.00	207
total	0.99	0.99	0.99	800

Table 5.1.: The winning classifier which can also be transferred into `CoreML`.

5.7. Full Stack ML, a Working System

The iPhone application contains a debug/demo model which uses the exported SVM and the handcrafted feature engineering. The idea is to have a full working system as a demo. While we could transfer the model itself to iOS by the python tool `coremltools.converters.sklearn`, this is not possible for the feature engineering. Most models transferred by `CoreML` seem to be DeepLearning models which do the feature engineering in the algorithm themselves. In the case of classical ML, feature engineering is a vital part. Not being able to transfer the feature engineering part automatically therefor is a major weakness of the `CoreML` tools. Swift lacks a good linear algebra library. There is the Apple accelerate framework which is built on Objective-C and C but can accelerate the linear algebra computation on the GPU or the Neural Engine of the iPhone if it has one. We build a wrapper for the methods we need and re-implemented the feature engineering. To test it, we created some synthetic signal and ran the feature engineering in python and in swift in an own swift notebook. We corrected the implementations until they gave the same output. The module `Math.swift` has the important math operations needed for feature engineering. The module `TransformToFeature` implements the feature transformation for one window. The class `Classifier` collects the data from the `WC` session. It then waits until there are at least 128 frames of data. Then it transforms the current window into features every 0.4 seconds and makes a classification. The system is performant enough to do that on an iPhone 7 (also tested on an iPhone Xs).

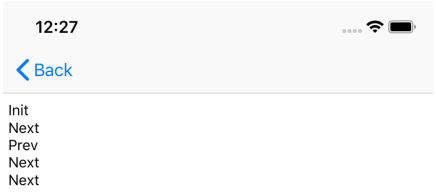


Figure 5.9.: Debug output of the recognition

We only report a label if the label is different from the last classification or if 2s passed, to avoid outputting too many labels in the debug mode.

An example can be seen in figure 5.9. If configured we can send this change in detection to a server. We are working on a server application running on a laptop to control a slide set when such a request is sent. This server is work in progress.

5.8. Conclusion

From the experience of using the debug/demo mode of the application, the recognition is by far not as good as the 0.99% of the SVM would suggest. We still need to figure out why that is that way. Without keeping a rigorous statistic it is more like 70% percent.

Training the system with different strategies for the partial windows is also worth an investigation for the smartwatch experiment. Also collecting more data, and try the same train, validation and test split would be interesting.

We wanted to use a full stack system which does the recognition of the gesture on the device, not on a server, so you can f.e. control you slideshow anywhere where you can open any network connection to you laptop. This is possible by transferring the models using the `CoreML` toolchain. However, not all models which are state of the art are supported. Also in classical ML the feature engineering is a vital part, and there is no easy method to transfer the feature engineering. Different to the smartglove is the data collection, as it relies on other technologies. We build a labeling app for that. We also showed a way to augment data from an IMU to get more data, which helped in training the models. With this method your are able to triple the data size.

Still there is a set of open issues with the system: First of all while the performance on the validation set is very good, the system does not perform as well while using it live. It is to be investigated why this is the case. Especially, when you do such a simple set, that you can handcraft very simple thresholds and perform very well, if the system does not perform almost perfectly there is no sense in doing it with machine learning.

It would be interesting to try the data augmentation on the smartglove, and extend it to sensors which report absolut values like the pressure and flexion sensor. This should be easier, as the stretching and shrinking could be done with interpolation.

Another interesting thing would be to find a way to transfer the data set from the smart glove's IMU on the wrist (and maybe the other data as a prior) so this can be used for the training of the watch. Two main changes would be needed: Firstly, the value ranges must be adopted to the different recordings. Secondly, the smartglove's data is sampled with $83.3Hz$ and the smartwatch is sampled with $50Hz$. Resampling the method could f.e. be done by first approximating the function with cubic splines and then sampling from this representation again.

Even more interesting would be to encoporate other datasets and use DeepLearning to train an IMU natural movement model and then see if this model can be transferred to help with inference.

Overall this experiment showed that you can use the techniques from the smartglove also on a more commonly used wearable.

Chapter 6

Discussion of the work

This work gives an introduction into the theory of gesture recognition, its technology, its methods and its use cases. It sheds light on what a basic gesture is and on the anatomy of the human arm and its kinematics and needed degrees of freedom.

The work assumes that the dynamics of the hand can be captured by sliding windows and feature extraction from human activity recognition (HAR) research. It gives an extensive overview of the features used in HAR and applies them to this problem. This work also gives an overview of the state of the art classification algorithms in machine learning.

It presents two experiments, one with a custom built smartglove based on earlier work done in Luzhnica et al. [2016] and another one as a demo system with a consumer wearable, a smartwatch.

The thesis gives more detailed documentation of the electronics, data collection software and collected data than is possible in the page limited paper of Luzhnica et al. [2016]. It demonstrates the effect on outlier detection and data cleaning and gives more detailed statistics on the dataset.

We explore a data-driven approach towards gesture recognition in this work. It shows that by generating all kind of known features and using automatic tools to select the needed ones, it is possible to train very competitive models. It explains a theoretical optimal size of features and explores algorithmically different feature sizes and the impact on recognition.

Splitting training, validation and test data is done differently in this work. It shows that with a different validation split similar performances are reached than presented in Luzhnica et al. [2016]. It also shows with the unbalanced zero class in the test set that the special training with partial windows in the training set performs worse than in the original work with only full windows in training and dual labelling.

It shows the constraints of porting a system to a consumer device by creating a demo application for the smartwatch and smartphone, and which design decision (like limiting the choice of classification algorithm) have to be made. For this system it presents a data augmentation method for IMUs where labels are not corrupted by changing the timing of the time series and adding small amount of noise.

We present for the first time the *RSE* algorithm, an algorithm to perform sensor selection for a target system. We describe use cases and possible heuristics for the selection and show first insights on how the algorithm performs.

Appendix A

Data of the Glove

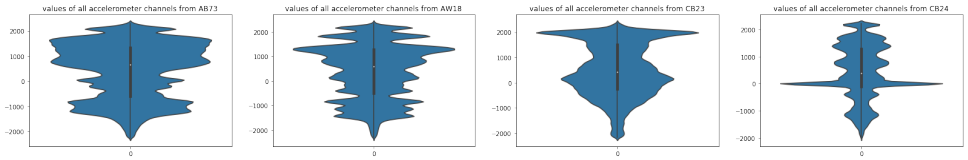


Figure A.1.: Valuerange of all accelerometer sensors for different users

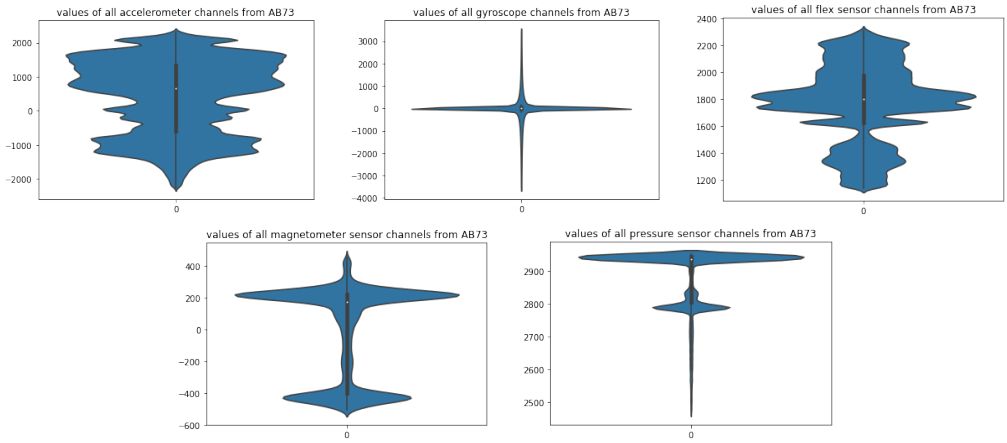


Figure A.2.: Value Distribution of different Sensors for user AB73

Table A.1.: All k2 test statistics and p-values of all user/sensor type combination when performing a normal test.

First user	Second sensor	N	min	max	k2	p
AB73	acceleromter	3843168	-2636	2518	8.444566e+05	0.0
AF82	acceleromter	2222535	-2439	2336	2.197433e+06	0.0
AL29	acceleromter	3869796	-2372	2497	6.599598e+06	0.0
AW18	acceleromter	2166339	-2787	2601	3.664112e+05	0.0

Continued on next page

Table A.1 – *Continued from previous page*

First user	Second sensor	N	min	max	k2	p
CB23	acceleromter	2211951	-2313	2566	1.668757e+05	0.0
CB24	acceleromter	2227722	-2367	2426	1.270681e+05	0.0
CF58	acceleromter	2505216	-2783	2513	8.868839e+05	0.0
DG12	acceleromter	2195571	-3074	2545	3.843390e+05	0.0
DH42	acceleromter	2319135	-2654	2451	1.001795e+06	0.0
DL24	acceleromter	3806838	-2614	2707	8.133829e+05	0.0
JL61	acceleromter	2363886	-2711	2562	9.164507e+05	0.0
JQ28	acceleromter	4023516	-2617	2541	1.253615e+06	0.0
JS52	acceleromter	3936009	-2839	2729	5.544492e+05	0.0
MF20	acceleromter	2163441	-2461	2384	5.990880e+05	0.0
MS55	acceleromter	2285892	-2850	2582	1.140166e+06	0.0
PC29	acceleromter	2522898	-14286	10800	2.358820e+05	0.0
PM32	acceleromter	3794028	-3364	2772	4.439832e+05	0.0
PS42	acceleromter	2221695	-2241	2373	2.555001e+06	0.0
RR45	acceleromter	2218671	-3019	2689	1.065922e+06	0.0
RW32	acceleromter	2222241	-3198	2647	1.944557e+05	0.0
SF1	acceleromter	2167305	-8986	9240	2.183389e+05	0.0
YW13	acceleromter	3928722	-2922	2551	5.617225e+05	0.0
AB73	flexation	2379104	1114	2391	1.072472e+05	0.0
AF82	flexation	1375855	1136	2354	3.100936e+04	0.0
AL29	flexation	2395588	1142	2347	7.334131e+04	0.0
AW18	flexation	1341067	1075	2387	4.440907e+04	0.0
CB23	flexation	1369303	1026	2342	1.062614e+05	0.0
CB24	flexation	1379066	1101	2391	3.545334e+04	0.0
CF58	flexation	1550848	1097	2399	5.034919e+04	0.0
DG12	flexation	1359163	1082	2352	3.654121e+04	0.0
DH42	flexation	1435655	1113	2365	2.052000e+04	0.0
DL24	flexation	2356614	1071	2315	1.555223e+05	0.0
JL61	flexation	1463358	1144	2385	3.242145e+04	0.0
JQ28	flexation	2490748	1053	2932	5.733102e+05	0.0
JS52	flexation	2436577	1081	2388	9.361195e+04	0.0
MF20	flexation	1339273	1082	2372	3.808368e+04	0.0
MS55	flexation	1415076	1145	2379	1.815430e+04	0.0
PC29	flexation	1561794	23	2350	5.143577e+05	0.0
PM32	flexation	2348684	1129	2355	5.122904e+04	0.0
PS42	flexation	1375335	1096	2344	4.353182e+04	0.0
RR45	flexation	1373463	1115	2404	6.889147e+03	0.0
RW32	flexation	1375673	1098	2385	4.279792e+04	0.0
SF1	flexation	1341665	27	2365	4.481073e+05	0.0
YW13	flexation	2432066	1085	2301	1.186358e+05	0.0
AB73	gyroscope	3843168	-11103	11352	1.096027e+06	0.0
AF82	gyroscope	2222535	-11075	12317	7.397091e+05	0.0
AL29	gyroscope	3869796	-10113	11555	1.309652e+06	0.0
AW18	gyroscope	2166339	-11926	14597	8.752041e+05	0.0
CB23	gyroscope	2211951	-9273	10433	6.245442e+05	0.0
CB24	gyroscope	2227722	-11223	13308	9.891769e+05	0.0
CF58	gyroscope	2505216	-11673	14507	1.189913e+06	0.0
DG12	gyroscope	2195571	-12672	14081	7.289906e+05	0.0
DH42	gyroscope	2319135	-10943	11868	7.902876e+05	0.0
DL24	gyroscope	3806838	-11918	12613	9.612428e+05	0.0
JL61	gyroscope	2363886	-12260	13750	8.667529e+05	0.0

Continued on next page

Table A.1 – *Continued from previous page*

First user	Second sensor	N	min	max	k2	p
JQ28	gyroscope	4023516	-12645	14789	1.441606e+06	0.0
JS52	gyroscope	3936009	-10089	12097	1.125299e+06	0.0
MF20	gyroscope	2163441	-9134	10614	8.029807e+05	0.0
MS55	gyroscope	2285892	-11756	13370	7.655694e+05	0.0
PC29	gyroscope	2522898	-14031	18776	1.264157e+06	0.0
PM32	gyroscope	3794028	-13077	14657	1.237140e+06	0.0
PS42	gyroscope	2221695	-8394	11860	1.175412e+06	0.0
RR45	gyroscope	2218671	-12636	14814	9.845657e+05	0.0
RW32	gyroscope	2222241	-11333	14239	8.636745e+05	0.0
SF1	gyroscope	2167305	-9593	11149	9.150341e+05	0.0
YW13	gyroscope	3928722	-12754	13966	1.288589e+06	0.0
AB73	pressure	915040	453	2953	1.179727e+06	0.0
AF82	pressure	529175	1267	2952	1.388097e+05	0.0
AL29	pressure	921380	706	2955	2.320638e+05	0.0
AW18	pressure	515795	620	2948	4.545777e+05	0.0
CB23	pressure	526655	113	2947	7.638754e+05	0.0
CB24	pressure	530410	703	3013	5.656773e+05	0.0
CF58	pressure	596480	413	2952	7.013277e+05	0.0
DG12	pressure	522755	537	2944	6.541262e+04	0.0
DH42	pressure	552175	278	2954	2.031838e+05	0.0
DL24	pressure	906390	446	2941	1.325338e+06	0.0
JL61	pressure	562830	909	2953	5.145988e+05	0.0
JQ28	pressure	957980	498	2947	7.386344e+05	0.0
JS52	pressure	937145	804	2956	9.001867e+05	0.0
MF20	pressure	515105	501	2947	5.066696e+05	0.0
MS55	pressure	544260	615	2952	3.323270e+05	0.0
PC29	pressure	600690	194	2948	6.733048e+05	0.0
PM32	pressure	903340	573	2945	9.271505e+05	0.0
PS42	pressure	528975	907	2945	1.932210e+05	0.0
RR45	pressure	528255	218	2944	6.310491e+04	0.0
RW32	pressure	529105	471	2954	3.719549e+05	0.0
SF1	pressure	516025	383	3014	7.990293e+05	0.0
YW13	pressure	935410	439	2942	9.932568e+04	0.0

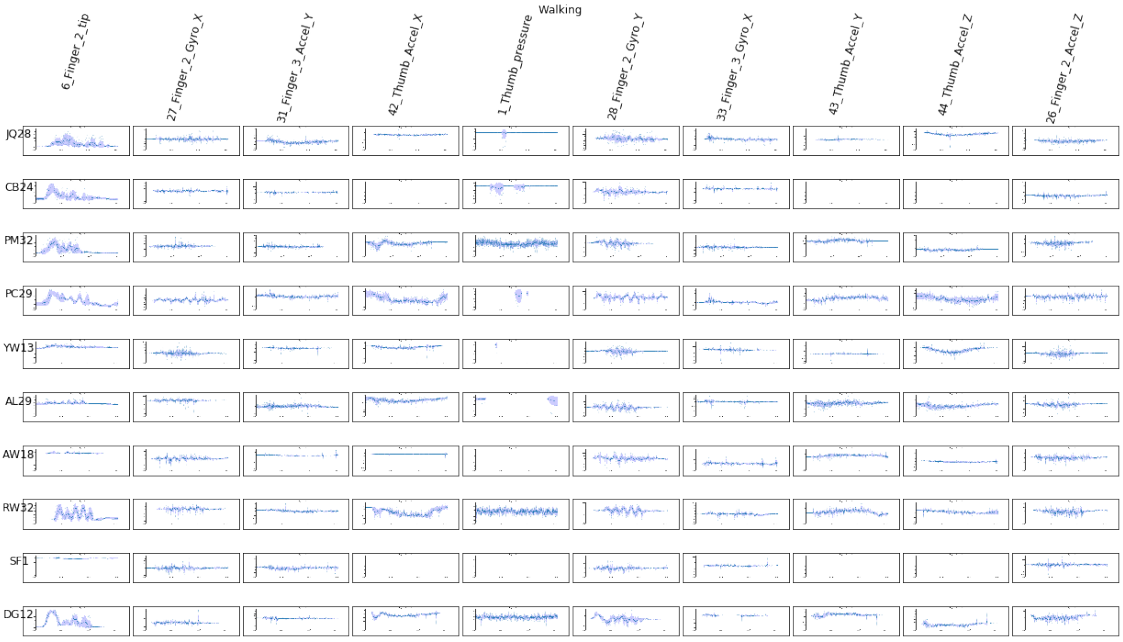


Figure A.3.: Time Series of 10 users among 10 channels for the gesture walking

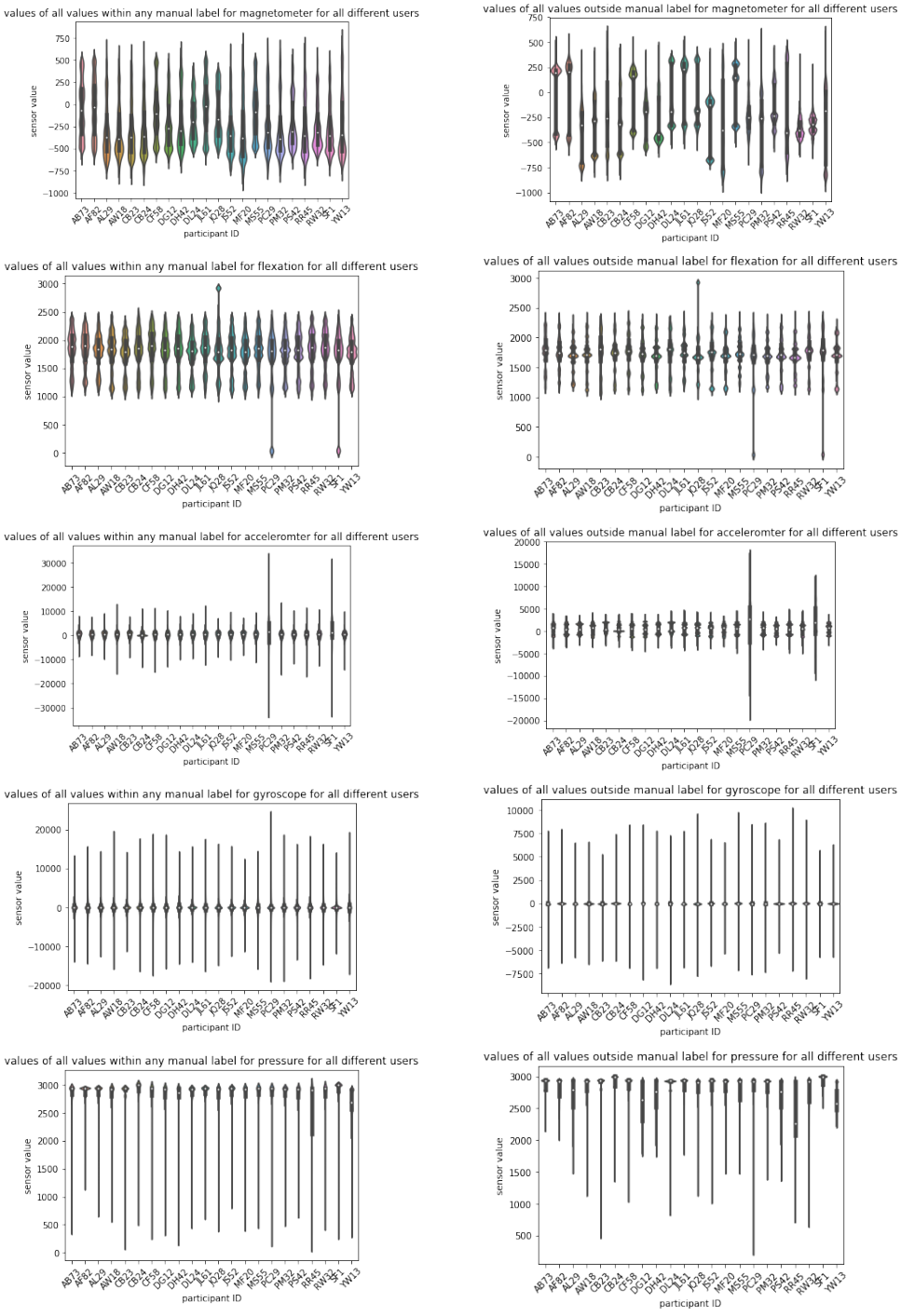


Figure A.4.: Comparison of Value Distributions within labels or within the zero class

Appendix B

Data of the Smart Watch

	algorithm	c	decision function	g	kernel	p	total-valid
238	SVC	100.0	ovr	auto	sigmoid	-	0.973654
237	SVC	50.0	ovr	auto	sigmoid	-	0.973654
236	SVC	50.0	ovr	auto	rbf	-	0.973654
235	SVC	100.0	ovo	auto	sigmoid	-	0.973654
234	SVC	50.0	ovo	auto	sigmoid	-	0.973654
233	SVC	50.0	ovo	auto	rbf	-	0.973654
232	SVC	50.0	ovr	0.01	sigmoid	-	0.972324
231	SVC	50.0	ovo	0.01	sigmoid	-	0.972324
230	SVC	100.0	ovr	auto	poly	2.0	0.970994
229	SVC	100.0	ovo	auto	poly	2.0	0.970994
226	SVC	50.0	ovo	0.01	poly	4.0	0.970966
225	SVC	50.0	ovo	0.01	poly	3.0	0.970966
228	SVC	50.0	ovr	0.01	poly	4.0	0.970966
227	SVC	50.0	ovr	0.01	poly	3.0	0.970966
224	SVC	100.0	ovr	auto	rbf	-	0.969664
223	SVC	50.0	ovr	auto	poly	2.0	0.969664
222	SVC	100.0	ovo	auto	rbf	-	0.969664
221	SVC	50.0	ovo	auto	poly	2.0	0.969664
216	SVC	100.0	ovo	0.01	poly	4.0	0.969637
213	SVC	50.0	ovo	0.01	poly	2.0	0.969637

Table B.1.: Best configurations of SVMs for the smartwatch demo

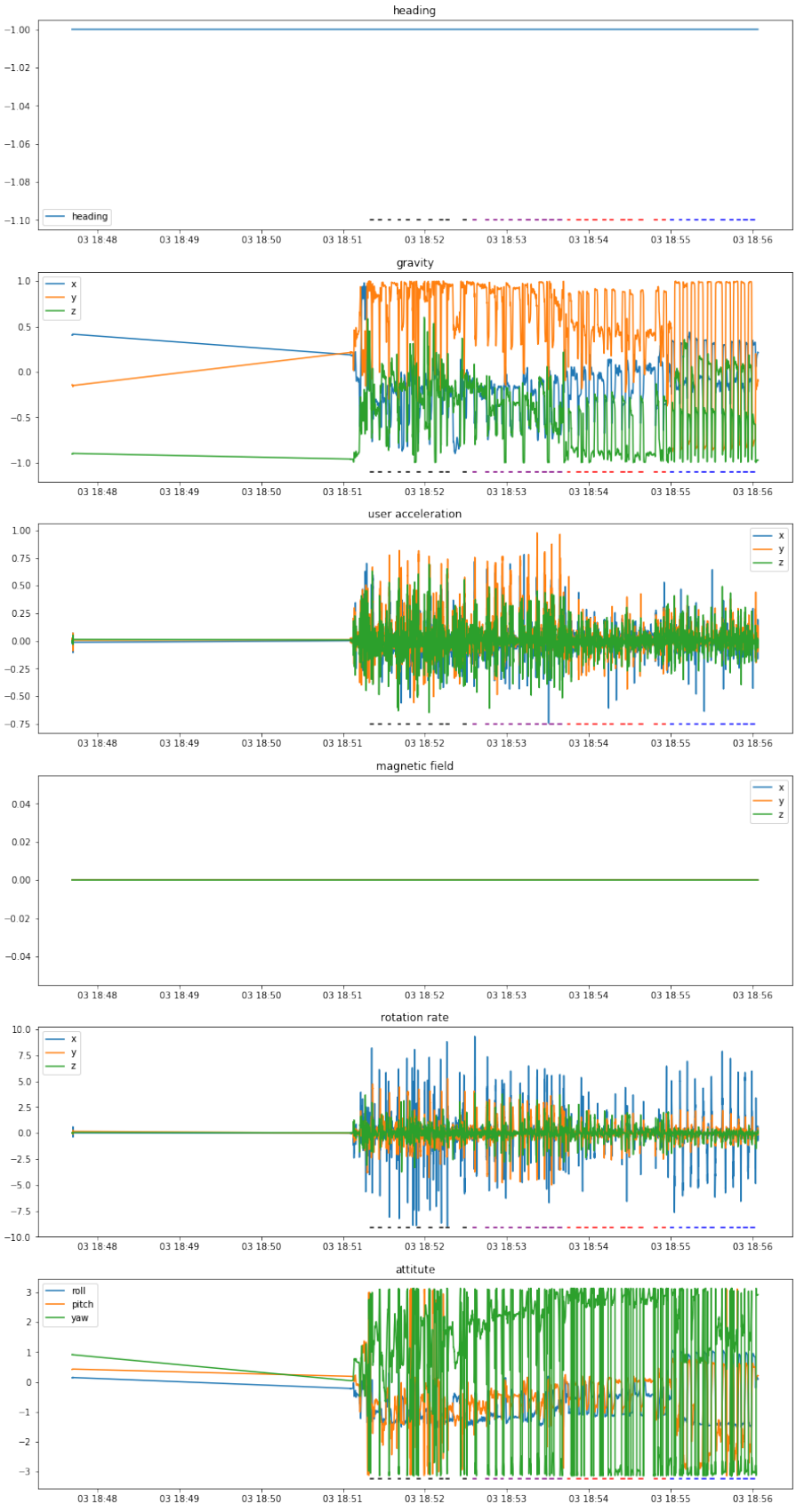


Figure B.1.: Data from a session where the watch stored data from a previous recording. In this case you have some left over recordings on the first connection

Bibliography

Irman Abdic, Lex Fridman, Daniel McDuff, Erik Marchi, Bryan Reimer, and Bj""orn Schuller. 2016. Driver frustration detection from audio and video in the wild. In *KI 2016: Advances in Artificial Intelligence: 39th Annual German Conference on AI, Klagenfurt, Austria, September 26-30, 2016, Proceedings*, Vol. 9904. Springer, 237. (Cited on page 15.)

Sreejan Alapati and Shivraj Yeole. 2017. A Review on Applications of Flex Sensors. *International Journal of Emerging Technology and Advanced Engineering* 7 (07 2017), 97–100. (Cited on page 28.)

Mohamed Aly. 2005. Survey on Multiclass Classification Methods. (2005). (Cited on page 18.)

A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha. 2017. A Low Power, Fully Event-Based Gesture Recognition System. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7388–7397. DOI:http://dx.doi.org/10.1109/CVPR.2017.781 (Cited on page 8.)

Ling Bao and StephenS. Intille. 2004. Activity Recognition from User-Annotated Acceleration Data. In *Pervasive Computing*, Alois Ferscha and Friedemann Mattern (Eds.). Lecture Notes in Computer Science, Vol. 3001. Springer Berlin Heidelberg, 1–17. DOI:http://dx.doi.org/10.1007/978-3-540-24646-6_1 (Cited on page 16.)

R. G. Baraniuk. 2007. Compressive Sensing [Lecture Notes]. *IEEE Signal Processing Magazine* 24, 4 (July 2007), 118–121. DOI:http://dx.doi.org/10.1109/MSP.2007.4286571 (Cited on page 17.)

John Barnard and Xiao-Li Meng. 1999. Applications of multiple imputation in medical studies: from AIDS to NHANES. *Statistical Methods in Medical Research* 8, 1 (1999), 17–36. DOI:http://dx.doi.org/10.1177/096228029900800103 PMID: 10347858. (Cited on page 11.)

G. Bernieri, L. Faramondi, and F. Pascucci. 2015. A low cost smart glove for visually impaired people mobility. In *Control and Automation (MED), 2015 23th Mediterranean Conference on*. 130–135. DOI:http://dx.doi.org/10.1109/MED.2015.7158740 (Cited on page 7.)

Gerald Bieber, Thomas Kirste, and Michael Gaede. 2014. Low Sampling Rate for Physical Activity Recognition. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '14)*. ACM, New York, NY, USA, Article 15, 8 pages. DOI:http://dx.doi.org/10.1145/2674396.2674446 (Cited on page 17.)

Henrik Birk, Thomas B. Moeslund, and Claus B. Madsen. 1997. Real-Time Recognition of Hand Alphabet Gestures Using Principal Component Analysis. In *In 10th Scandinavian Conference on Image Analysis*. (Cited on pages 5, 8, 9, and 23.)

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg. (Cited on pages 11, 12, 18, 20, 21, 23, 24, and 48.)

K. Bohm, K. Hubner, and W. Vaanaen. 1992. GIVEN: Gesture driven Interactions in Virtual Environments. A Toolkit Approach to 3D Interactions. In *Proceedings Interfaces to Real and Virtual Worlds*. 243–254. (Cited on page 5.)

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT '92)*. ACM, New York, NY, USA, 144–152. DOI:http://dx.doi.org/10.1145/130385.130401 (Cited on pages 11, 12, and 21.)

Leo Breiman. 1996. Bagging Predictors. *Machine Learning* 24, 2 (01 Aug 1996), 123–140. DOI:http://dx.doi.org/10.1023/A:1018054314350 (Cited on page 24.)

Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32. DOI:http://dx.doi.org/10.1023/A:1010933404324 (Cited on page 24.)

Markus Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-Based Local Outliers. In *PROCEEDINGS OF THE 2000 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 93–104. (Cited on pages 10 and 36.)

Andreas Bulling, Ulf Blanke, and Bernt Schiele. 2014. A Tutorial on Human Activity Recognition Using Body-worn Inertial Sensors. *ACM Comput. Surv.* 46, 3, Article 33 (Jan. 2014), 33 pages. DOI:http://dx.doi.org/10.1145/2499621 (Cited on page 48.)

Grigore C. Burdea. 1996. *Force and Touch Feedback for Virtual Reality*. John Wiley & Sons, Inc., New York, NY, USA. (Cited on page 26.)

Deng Cai, Student Member, Xiaofei He, Jiawei Han, and Senior Member. 2008. SRDA: An Efficient Algorithm for Large-Scale Discriminant Analysis. *IEEE Transactions on Knowledge and Data Engineering* (2008), 1–12. (Cited on page 48.)

Y. Cai. 2014. *Ambient Diagnostics*. CRC Press. https://books.google.at/books?id=xzvSBQAAQBAJ (Cited on page 25.)

Juli Camps, Albert Sam, Mario Martn, Daniel Rodrguez-Martn, Carlos Prez-Lpez, Joan M. Moreno Arostegui, Joan Cabestany, Andreu Catal, Sheila Alcaine, Berta Mestre, Anna Prats, Maria C. Crespo-Maraver, Timothy J. Counihan, Patrick Browne, Leo R. Quinlan, Gearid Laighin, Dean Sweeney, Hadas Lewy, Gabriel Vainstein, Alberto Costa, Roberta Annicchiarico, ngels Bays, and Alejandro Rodrguez-Molinero. 2018. Deep Learning for Freezing of Gait Detection in Parkinsons Disease Patients in Their Homes Using a Waist-worn Inertial Measurement Unit. *Know.-Based Syst.* 139, C (Jan. 2018), 119–131. DOI:http://dx.doi.org/10.1016/j.knosys.2017.10.017 (Cited on pages 15, 22, 31, and 67.)

N. Cardoso, J. Madureira, and N. Pereira. 2016. Smartphone-based transport mode detection for elderly care. In *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*. 1–6. (Cited on page 11.)

Ryan Chambers, Tim J. Gabbett, Michael H. Cole, and Adam Beard. 2015. The Use of Wearable Microsensors to Quantify Sport-Specific Movements. *Sports Medicine* 45, 7 (01 Jul 2015), 1065–1081. DOI:http://dx.doi.org/10.1007/s40279-015-0332-9 (Cited on page 54.)

Mahalanobis Prasanta Chandra and others. 1936. On the generalised distance in statistics. In *Proceedings of the National Institute of Sciences of India*, Vol. 2. 49–55. (Cited on page 23.)

Jae-Hwan Chang and L. Tassiulas. 2000. Energy conserving routing in wireless ad-hoc networks. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, Vol. 1. 22–31 vol.1. DOI:http://dx.doi.org/10.1109/INFCOM.2000.832170 (Cited on page 54.)

Yin-Wen Chang and Chih-Jen Lin. 2008. Feature Ranking Using Linear SVM. In *Proceedings of the Workshop on the Causation and Prediction Challenge at WCCI 2008 (Proceedings of Machine Learning Research)*, Isabelle Guyon, Constantin Aliferis, Greg Cooper, André Elisseeff, Jean-Philippe Pellet, Peter Spirtes, and Alexander Statnikov (Eds.), Vol. 3. PMLR, Hong Kong, 53–64. http://proceedings.mlr.press/v3/chang08a.html (Cited on page 17.)

Qing Chen, Nicolas D. Georganas, and E.M. Petriu. 2007. Real-time Vision-based Hand Gesture Recognition Using Haar-like Features. In *Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007. IEEE*. 1–6. DOI:http://dx.doi.org/10.1109/IMTC.2007.379068 (Cited on pages 8 and 24.)

K Chokkanathan and S Koteeswaran. 2018. A Study on Machine Learning: Elements, Characteristics and Algorithms. *International Journal of Engineering and Technology(UAE)* 7 (04 2018), 31–35. DOI: http://dx.doi.org/10.14419/ijet.v7i2.19.13793 (Cited on page 19.)

Salvador Cobos, Manuel Ferre, M. Ángel Sánchez-Urán, Javier Ortego, and Rafael Aracil. 2010. Human hand descriptions and gesture recognition for object manipulation. *Computer Methods in Biomechanics and Biomedical Engineering* 13, 3 (2010), 305–317. DOI:http://dx.doi.org/10.1080/10255840903208171 PMID: 20146129. (Cited on pages xi, 3, and 6.)

James W. Cooley and John W. Tukey. 1965. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comput.* 19 (1965), 297–301. DOI:http://dx.doi.org/10.1090/S0025-5718-1965-0178586-1 (Cited on page 14.)

D. Coomans and D.L. Massart. 1982. Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-Nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta* 136 (1982), 15 – 27. DOI:http://dx.doi.org/https://doi.org/10.1016/S0003-2670(01)95359-0 (Cited on page 22.)

Eva Coupeté, Fabien Moutarde, and Sotiris Manitsaris. 2015. Gesture Recognition Using a Depth Camera for Human Robot Collaboration on Assembly Line. *Procedia Manufacturing* 3 (2015), 518 – 525. DOI: http://dx.doi.org/10.1016/j.promfg.2015.07.216 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, {AHFE} 2015. (Cited on page 7.)

T. Cover and P. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13, 1 (January 1967), 21–27. DOI:http://dx.doi.org/10.1109/TIT.1967.1053964 (Cited on page 22.)

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online Passive-Aggressive Algorithms. *J. Mach. Learn. Res.* 7 (Dec. 2006), 551–585. http://dl.acm.org/citation.cfm?id=1248547.1248566 (Cited on pages 11, 12, and 20.)

Ralph D'Agostino and E. S. Pearson. 1973. Tests for Departure from Normality. Empirical Results for the Distributions of b_2 and $\sqrt{b_1}$. *Biometrika* 60, 3 (1973), 613–622. http://www.jstor.org/stable/2335012 (Cited on page 41.)

Ralph B. D'Agostino. 1971. An Omnibus Test of Normality for Moderate and Large Size Samples. *Biometrika* 58, 2 (1971), 341–348. http://www.jstor.org/stable/2334522 (Cited on page 41.)

N.H. Dardas and Nicolas D. Georganas. 2011. Real-Time Hand Gesture Detection and Recognition Using Bag-of-Features and Support Vector Machine Techniques. *Instrumentation and Measurement, IEEE Transactions on* 60, 11 (Nov 2011), 3592–3607. DOI:http://dx.doi.org/10.1109/TIM.2011.2161140 (Cited on pages 8, 9, and 22.)

Frauke Degenhardt, Stephan Seifert, and Silke Szymczak. 2017. Evaluation of variable selection methods for random forests and omics data sets. *Briefings in Bioinformatics* 20, 2 (10 2017), 492–503. DOI: http://dx.doi.org/10.1093/bib/bbx124 (Cited on page 53.)

ThomasG. Dietterich. 2002. Machine Learning for Sequential Data: A Review. In *Structural, Syntactic, and Statistical Pattern Recognition*, Terry Caelli, Adnan Amin, RobertP.W. Duin, Dick de Ridder, and Mohamed Kamel (Eds.). Lecture Notes in Computer Science, Vol. 2396. Springer Berlin Heidelberg, 15–30. DOI:http://dx.doi.org/10.1007/3-540-70659-3_2 (Cited on page 45.)

Pedro M Domingos. 2012. A few useful things to know about machine learning. *Commun. ACM* 55, 10 (2012), 78–87. (Cited on pages 19 and 20.)

Pan Du, Simon M. Lin, and Warren A. Kibbe. 2006. Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics* 22, 17 (07 2006), 2059–2065. DOI:http://dx.doi.org/10.1093/bioinformatics/btl355 (Cited on page 16.)

L. E. Dunne, B. Smyth, and B. Caulfield. 2007. A Comparative Evaluation of Bend Sensors for Wearable Applications. In *2007 11th IEEE International Symposium on Wearable Computers*. 121–122. DOI: http://dx.doi.org/10.1109/ISWC.2007.4373797 (Cited on page 26.)

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. Knowledge Discovery and Data Mining: Towards a Unifying Framework. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 82–88. http://dl.acm.org/citation.cfm?id=3001460.3001477 (Cited on page 10.)

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research* 15 (2014), 3133–3181. http://jmlr.org/papers/v15/delgado14a.html (Cited on page 19.)

Carlos Fernandez-Lozano, Jose A. Seoane, Marcos Gestal, Tom R. Gaunt, Julian Dorado, and Colin Campbell. 2015. Texture classification using feature selection and kernel-based techniques. *Soft Computing* 19, 9 (01 Sep 2015), 2469–2480. DOI:http://dx.doi.org/10.1007/s00500-014-1573-5 (Cited on page 53.)

Laura Fiorini, Filippo Cavallo, Paolo Dario, Alexandra Eavis, and Praminda Caleb-Solly. 2017. Unsupervised Machine Learning for Developing Personalised Behaviour Models Using Activity Data. *Sensors* 17, 5 (2017). DOI:http://dx.doi.org/10.3390/s17051034 (Cited on page 53.)

R. A. Fisher. 1936. THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS. *Annals of Eugenics* 7, 2 (1936), 179–188. DOI:http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x (Cited on page 23.)

Giancarlo Fortino, Stefano Galzarano, Raffaele Gravina, and Wenfeng Li. 2015. A framework for collaborative computing and multi-sensor data fusion in body sensor networks. *Information Fusion* 22 (2015), 50 – 70. DOI:http://dx.doi.org/https://doi.org/10.1016/j.inffus.2014.03.005 (Cited on pages xi, 14, 15, 16, 24, and 25.)

Yoav Freund, Robert Schapire, and Naoki Abe. 1999. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence* 14, 771-780 (1999), 1612. (Cited on page 24.)

Yoav Freund and Robert E Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (1997), 119 – 139. DOI:http://dx.doi.org/https://doi.org/10.1006/jcss.1997.1504 (Cited on page 24.)

Jerome Friedman. 1989. Regularized Discriminant Analysis. *Journal of The American Statistical Association - J AMER STATIST ASSN* 84 (03 1989), 165–175. DOI:http://dx.doi.org/10.1080/01621459.1989.10478752 (Cited on page 20.)

Jerome Friedman. 2002. Stochastic gradient boosting. *Computational statistics & data analysis* 38, 4 (2002), 367–378. (Cited on page 24.)

Jerome Friedman, Trevor Hastie, and Rob Tibshirani. 2010. Regularized Paths for Generalized Linear Models Via Coordinate Descent. *Journal of Statistical Software* 33 (02 2010). DOI:http://dx.doi.org/10.1163/ej.9789004178922.i-328.7 (Cited on pages 20 and 21.)

Wenjiang J. Fu. 1998. Penalized Regressions: The Bridge versus the Lasso. *Journal of Computational and Graphical Statistics* 7, 3 (1998), 397–416. DOI:http://dx.doi.org/10.1080/10618600.1998.10474784 (Cited on pages 20 and 21.)

Pragati Garg, Naveen Aggarwal, and Sanjeev Sofat. 2009. Vision based hand gesture recognition. *World Academy of Science, Engineering and Technology* 49, 1 (2009), 972–977. (Cited on pages 3 and 8.)

Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely Randomized Trees. *Mach. Learn.* 63, 1 (April 2006), 3–42. DOI:http://dx.doi.org/10.1007/s10994-006-6226-1 (Cited on page 24.)

Przemys Glomb, Michal Romaszewski, Sebastian Opozda, and Arkadiusz Sochan. 2012. Choosing and Modeling the Hand Gesture Database for a Natural User Interface. In *Proceedings of the 9th International Conference on Gesture and Sign Language in Human-Computer Interaction and Embodied Communication (GW'11)*. Springer-Verlag, Berlin, Heidelberg, 24–35. DOI:http://dx.doi.org/10.1007/978-3-642-34182-3_3 (Cited on pages 5, 9, 23, and 27.)

Baptiste Gregorutti, Bertrand Michel, and Philippe Saint-Pierre. 2017. Correlation and variable importance in random forests. *Statistics and Computing* 27, 3 (01 May 2017), 659–678. DOI:http://dx.doi.org/10.1007/s11222-016-9646-1 (Cited on page 53.)

J. Grifka and M. Kuster. 2011. *Orthopädie und Unfallchirurgie*. Springer Berlin Heidelberg. https://books.google.at/books?id=2kofUfrqiB4C (Cited on page 3.)

Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, Mar (2003), 1157–1182. (Cited on page 17.)

Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. 2002. Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning* 46, 1 (01 Jan 2002), 389–422. DOI:http://dx.doi.org/10.1023/A:1012487302797 (Cited on page 17.)

Jesús M. Gómez-de Gabriel and William Harwin. 2015. Evaluation of Sensor Configurations for Robotic Surgical Instruments. *Sensors* 15, 10 (2015), 27341–27358. DOI:http://dx.doi.org/10.3390/s151027341 (Cited on pages 53 and 54.)

Jaehyun Han, Sunggeun Ahn, and Geehyuk Lee. 2015. Transture: Continuing a Touch Gesture on a Small Screen into the Air. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '15)*. ACM, New York, NY, USA, 1295–1300. DOI:http://dx.doi.org/10.1145/2702613.2732849 (Cited on pages 8 and 9.)

Mokhtar M Hasan and Pramod K Mishra. 2012. Hand gesture modeling and recognition using geometric features: a review. *Canadian Journal on Image Processing and Computer Vision* 3, 1 (2012), 12–26. (Cited on page 8.)

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. Springer. http://www.worldcat.org/oclc/300478243 (Cited on pages 23 and 24.)

W.T. Higgins. 1975. A Comparison of Complementary and Kalman Filtering. *Aerospace and Electronic Systems, IEEE Transactions on* AES-11, 3 (May 1975), 321–325. DOI:http://dx.doi.org/10.1109/TAES.1975.308081 (Cited on pages 13, 45, and 46.)

Tin Kam Ho. 1995. Random Decision Forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1 (ICDAR '95)*. IEEE Computer Society, Washington, DC, USA, 278–. http://dl.acm.org/citation.cfm?id=844379.844681 (Cited on page 24.)

Jianping Hua, Zixiang Xiong, James Lowey, Edward Suh, and Edward R. Dougherty. 2004. Optimal number of features as a function of sample size for various classification rules. *Bioinformatics* 21, 8 (11 2004), 1509–1515. DOI:http://dx.doi.org/10.1093/bioinformatics/bti171 (Cited on page 51.)

Chung-Lin Huang and Wen-Yi Huang. 1998. Sign Language Recognition Using Model-based Tracking and a 3D Hopfield Neural Network. *Mach. Vision Appl.* 10, 5-6 (April 1998), 292–307. DOI:http://dx.doi.org/10.1007/s001380050080 (Cited on page 5.)

Jie Huang, Wengang Zhou, Houqiang Li, and Weiping Li. 2015. Sign Language Recognition using 3D convolutional neural networks. In *2015 IEEE International Conference on Multimedia and Expo (ICME)*. 1–6. DOI:http://dx.doi.org/10.1109/ICME.2015.7177428 (Cited on page 8.)

Seung jean Kim, Kwangmoo Koh, Michael Lustig, Stephen Boyd, and Dimitry Gorinevsky. 2007. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine Learning Research* 2007 (2007). (Cited on page 21.)

J.-F. Jego, A. Paljic, and P. Fuchs. 2013. User-defined gestural interaction: A study on gesture memorization. In *3D User Interfaces (3DUI), 2013 IEEE Symposium on*. 7–10. DOI:http://dx.doi.org/10.1109/3DUI.2013.6550189 (Cited on pages 5 and 27.)

David Jurman, Marko Jankovec, Roman Kamnik, and Marko Topič. 2007. Calibration and data fusion solution for the miniature attitude and heading reference system. *Sensors and Actuators A: Physical* 138, 2 (2007), 411 – 420. DOI:http://dx.doi.org/https://doi.org/10.1016/j.sna.2007.05.008 (Cited on page 25.)

Rudolph Emil Kalman. 1960. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering* 82, 1 (1960), 35–45. (Cited on pages 13 and 45.)

Manolya Kavakli, Meredith Taylor, and Anatoly Trapeznikov. 2007. Designing in Virtual Reality (DesIRe): A Gesture-based Interface. In *Proceedings of the 2Nd International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA '07)*. ACM, New York, NY, USA, 131–136. DOI:http://dx.doi.org/10.1145/1306813.1306842 (Cited on page 7.)

S. Kazi, A. As'Arry, M. Z. Md. Zain, M. Mailah, and M. Hussein. 2010. Experimental Implementation of Smart Glove Incorporating Piezoelectric Actuator for Hand Tremor Control. *WSEAS Trans. Sys. Ctrl.* 5, 6 (June 2010), 443–453. http://dl.acm.org/citation.cfm?id=1853907.1853914 (Cited on page 7.)

Holger Kenn, Friedrich Van Megen, and Robert Sugar. 2007. A glove-based gesture interface for wearable computing applications. In *Applied Wearable Computing (IFAWC), 2007 4th International Forum on*. 1–10. (Cited on page 7.)

Engineering Coimbatore Kiruthika, Engineering Coimbatore, and Navin Kumar. 2014. Survey on Hand Gesture Recognition. *International Journal of Engineering Research and Technology* 3, 2 (2014), 943–946. (Cited on page 9.)

David B. Koons, Carlton J. Sparrell, and Kristinn R. Thorisson. 1993. Intelligent Multimedia Interfaces. American Association for Artificial Intelligence, Menlo Park, CA, USA, Chapter Integrating Simultaneous Input from Speech, Gaze, and Hand Gestures, 257–276. http://dl.acm.org/citation.cfm?id=162477.162508 (Cited on page 5.)

H. Koskimaki, V. Huikari, P. Siirtola, P. Laurinen, and J. Roning. 2009. Activity recognition using a wrist-worn inertial measurement unit: A case study for industrial assembly lines. In *2009 17th Mediterranean Conference on Control and Automation*. 401–405. (Cited on pages 10, 11, 12, 14, 15, 16, 45, and 48.)

Heli Koskim"aki, Ville Huikari, Pekka Siirtola, and Juha R"oning. 2013. Behavior modeling in industrial assembly lines using a wrist-worn inertial measurement unit. *Journal of Ambient Intelligence and Humanized Computing* 4, 2 (2013), 187–194. DOI:http://dx.doi.org/10.1007/s12652-011-0061-3 (Cited on page 7.)

L. F. Kozachenko and N. N. Leonenko. 1987. A statistical estimate for the entropy of a random vector. *Problemy Peredachi Informatsii* 23, 2 (1987), 9–16. (Cited on page 17.)

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. 2004. Estimating mutual information. *Phys. Rev. E* 69 (Jun 2004), 066138. Issue 6. DOI:http://dx.doi.org/10.1103/PhysRevE.69.066138 (Cited on page 17.)

Narayanan C. Krishnan and Diane J. Cook. 2014. Activity Recognition on Streaming Sensor Data. *Pervasive Mob. Comput.* 10 (Feb. 2014), 138–154. DOI:http://dx.doi.org/10.1016/j.pmcj.2012.07.003 (Cited on pages 10, 11, 45, and 48.)

Steve Krug. 2005. *Don'T Make Me Think: A Common Sense Approach to the Web (2Nd Edition)*. New Riders Publishing, Thousand Oaks, CA, USA. (Cited on page 1.)

Arun Kulshreshth and Joseph J. LaViola, Jr. 2015. Exploring 3D User Interface Technologies for Improving the Gaming Experience. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 125–134. DOI:http://dx.doi.org/10.1145/2702123.2702138 (Cited on page 7.)

Piyush Kumar, Jyoti Verma, and Shitala Prasad. 2012. Hand data glove: A wearable real-time device for human-computer interaction. *International Journal of Advanced Science and Technology* 43 (2012). (Cited on page 7.)

Marc Erich Latoschik. 2005. A User Interface Framework for Multimodal VR Interactions. In *Proceedings of the 7th International Conference on Multimodal Interfaces (ICMI '05)*. ACM, New York, NY, USA, 76–83. DOI:http://dx.doi.org/10.1145/1088463.1088479 (Cited on page 1.)

Ed Lawson, Denson Smith, Donald Sofge, Paul Elmore, and Frederick Petry. 2017. Decision forests for machine learning classification of large, noisy seafloor feature sets. *Computers & Geosciences* 99 (2017), 116 – 124. DOI:http://dx.doi.org/https://doi.org/10.1016/j.cageo.2016.10.013 (Cited on page 24.)

Boon-Giin Lee, Boon-Leng Lee, and Wan-Young Chung. 2015. Wristband-Type Driver Vigilance Monitoring System Using Smartwatch. *Sensors Journal, IEEE* 15, 10 (Oct 2015), 5624–5633. DOI:http://dx.doi.org/10.1109/JSEN.2015.2447012 (Cited on page 7.)

Wonil Lee, Edmund Seto, Ken-Yu Lin, and Giovanni C. Migliaccio. 2017. An evaluation of wearable sensors and their placements for analyzing construction worker's trunk posture in laboratory conditions. *Applied Ergonomics* 65 (2017), 424 – 436. DOI:http://dx.doi.org/https://doi.org/10.1016/j.apergo.2017.03.016 (Cited on page 53.)

Heike Leutheuser, Dominik Schuldhaus, and Bjoern M. Eskofier. 2013. Hierarchical, Multi-Sensor Based Classification of Daily Life Activities: Comparison with State-of-the-Art Algorithms Using a Benchmark Dataset. *PLoS ONE* 8, 10 (10 2013), e75196. DOI:http://dx.doi.org/10.1371/journal.pone.0075196 (Cited on pages 14 and 16.)

H. Lin, M. Hsu, and W. Chen. 2014. Human hand gesture recognition using a convolution neural network. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. 1038–1043. DOI:http://dx.doi.org/10.1109/CoASE.2014.6899454 (Cited on page 8.)

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. IEEE Computer Society, Washington, DC, USA, 413–422. DOI:http://dx.doi.org/10.1109/ICDM.2008.17 (Cited on page 36.)

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. *ACM Trans. Knowl. Discov. Data* 6, 1, Article 3 (March 2012), 39 pages. DOI:http://dx.doi.org/10.1145/2133360.2133363 (Cited on pages 10 and 36.)

G. Luzhnica, J. Simon, E. Lex, and V. Pammer. 2016. A sliding window approach to natural hand gesture recognition using a custom data glove. In *2016 IEEE Symposium on 3D User Interfaces (3DUI)*. 81–90. DOI:http://dx.doi.org/10.1109/3DUI.2016.7460035 (Cited on pages 1, 2, 3, 4, 7, 9, 27, 32, 34, 44, 45, 46, 47, 48, 51, 60, 61, 72, and 75.)

Granit Luzhnica and Eduardo Veas. 2018. Investigating Interactions for Text Recognition Using a Vibrotactile Wearable Display. In *23rd International Conference on Intelligent User Interfaces (IUI '18)*. ACM, New York, NY, USA, 453–465. DOI:http://dx.doi.org/10.1145/3172944.3172951 (Cited on page 57.)

S.E. Lyshevski. 2002. *MEMS and NEMS: Systems, Devices, and Structures*. CRC Press. https://books.google.at/books?id=2L876Po9vOoC (Cited on page 25.)

Lei Ma, Tengyu Fu, Thomas Blaschke, Manchun Li, Dirk Tiede, Zhenjin Zhou, Xiaoxue Ma, and Deliang Chen. 2017. Evaluation of Feature Selection Methods for Object-Based Land Cover Mapping of Unmanned Aerial Vehicle Imagery Using Random Forest and Support Vector Machine Classifiers. *ISPRS International Journal of Geo-Information* 6, 2 (2017). DOI:http://dx.doi.org/10.3390/ijgi6020051 (Cited on pages 52 and 53.)

Zahra Sedighi Maman, Mohammad Ali Alamdar Yazdi, Lora A. Cavuoto, and Fadel M. Megahed. 2017. A data-driven approach to modeling physical fatigue in the workplace using wearable sensors. *Applied Ergonomics* 65 (2017), 515 – 529. DOI:http://dx.doi.org/https://doi.org/10.1016/j.apergo.2017.02.001 (Cited on pages 53 and 54.)

A. M. Martinez and A. C. Kak. 2001. PCA versus LDA. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23, 2 (Feb 2001), 228–233. DOI:http://dx.doi.org/10.1109/34.908974 (Cited on page 23.)

Tiago Martins, Christa Sommerer, Laurent Mignonneau, and Nuno Correia. 2008. Gauntlet: a wearable interface for ubiquitous gaming. In *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*. ACM Request Permissions, New York, New York, USA, 367. (Cited on page 7.)

G Mclachlan. 1999. Mahalanobis Distance. *Resonance* 4 (06 1999), 20–26. DOI:http://dx.doi.org/10.1007/BF02834632 (Cited on page 23.)

A. Mehrabian. 1972. *Nonverbal Communication*. Aldine Publishing Company. https://books.google.at/books?id=Xt-YALu9CGwC (Cited on page 3.)

Tom M. Mitchell. 1997. *Machine learning*. McGraw-Hill. http://www.worldcat.org/oclc/61321007 (Cited on page 18.)

Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Kari Pulli. 2015. Multi-sensor system for driver's hand-gesture recognition. In *IEEE Conference on Automatic Face and Gesture Recognition*. 1–8. (Cited on page 7.)

Fabian Mörchen. 2003. *Time series feature extraction for data mining using DWT and DFT*. Technical Report. Math and Computer Science Department, Philipps University, Marburg, Germany. (Cited on pages 16, 46, and 47.)

Kouichi Murakami and Hitomi Taguchi. 1991. Gesture Recognition Using Recurrent Neural Networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*. ACM, New York, NY, USA, 237–242. DOI:http://dx.doi.org/10.1145/108844.108900 (Cited on page 9.)

Samir Mustapha, Ali Braytee, and Lin Ye. 2018. Multisource Data Fusion for Classification of Surface Cracks in Steel Pipes. *Journal of Nondestructive Evaluation, Diagnostics and Prognostics of Engineering Systems* 1, 2 (24 Jan 2018), 021007–021007–11. DOI:http://dx.doi.org/10.1115/1.4038862 (Cited on page 17.)

V. X. Navas, J. Destefano, B. J. Koo, E. Doty, and D. Westerfeld. 2012. Smart glove. In *Systems, Applications and Technology Conference (LISAT), 2012 IEEE Long Island*. 1–4. DOI:http://dx.doi.org/10.1109/LISAT.2012.6223202 (Cited on page 7.)

P. Neto, D. Pereira, J. Norberto Pires, and A.P. Moreira. 2013. Real-time and continuous hand gesture spotting: An approach based on artificial neural networks. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. 178–183. DOI:http://dx.doi.org/10.1109/ICRA.2013.6630573 (Cited on page 9.)

Juan C. Núñez, Raúl Cabido, Juan J. Pantrigo, Antonio S. Montemayor, and José F. Vélez. 2018. Convolutional Neural Networks and Long Short-Term Memory for skeleton-based human activity and hand gesture recognition. *Pattern Recognition* 76 (2018), 80 – 94. DOI:http://dx.doi.org/https://doi.org/10.1016/j.patcog.2017.10.033 (Cited on page 8.)

B. O'Flynn, J. T. Sanchez, P. Angove, J. Connolly, J. Condell, K. Curran, and P. Gardiner. 2013. Novel smart sensor glove for arthritis rehabiliation. In *2013 IEEE International Conference on Body Sensor Networks*. 1–6. DOI:http://dx.doi.org/10.1109/BSN.2013.6575482 (Cited on page 7.)

Javier Ortiz Laguna, AngelGarcía Olaya, and Daniel Borrajo. 2011. A Dynamic Sliding Window Approach for Activity Recognition. In *User Modeling, Adaption and Personalization*, JosephA. Konstan, Ricardo Conejo, JoséL. Marzo, and Nuria Oliver (Eds.). Lecture Notes in Computer Science, Vol. 6787. Springer

Berlin Heidelberg, 219–230. DOI:http://dx.doi.org/10.1007/978-3-642-22362-4_19 (Cited on pages 10, 11, and 45.)

Jiazhi Ou, Yanxin Shi, Jeffrey Wong, Susan R. Fussell, and Jie Yang. 2006. Combining Audio and Video to Predict Helpers' Focus of Attention in Multiparty Remote Collaboration on Physical Tasks. In *Proceedings of the 8th International Conference on Multimodal Interfaces (ICMI '06)*. ACM, New York, NY, USA, 217–224. DOI:http://dx.doi.org/10.1145/1180995.1181040 (Cited on page 7.)

Cemil Oz and Ming C. Leu. 2011. American Sign Language Word Recognition with a Sensory Glove Using Artificial Neural Networks. *Eng. Appl. Artif. Intell.* 24, 7 (Oct. 2011), 1204–1213. DOI:http://dx.doi.org/10.1016/j.engappai.2011.06.015 (Cited on page 5.)

Dajeong Park, Miran Lee, Sunghee E. Park, Joon-Kyung Seong, and Inchan Youn. 2018. Determination of Optimal Heart Rate Variability Features Based on SVM-Recursive Feature Elimination for Cumulative Stress Monitoring Using ECG Sensor. *Sensors* 18, 7 (2018). DOI:http://dx.doi.org/10.3390/s18072387 (Cited on pages 17 and 52.)

V. I. Pavlovic, R. Sharma, and T. S. Huang. 1997. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19, 7 (July 1997), 677–695. DOI:http://dx.doi.org/10.1109/34.598226 (Cited on page 5.)

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830. (Cited on pages 19, 20, 21, and 51.)

Asanka G. Perera, Yee Wei Law, and Javaan S. Chahl. 2019. UAV-GESTURE: A Dataset for UAV Control and Gesture Recognition. *CoRR* abs/1901.02602 (2019). http://arxiv.org/abs/1901.02602 (Cited on page 8.)

S. S. Pradhan, J. Kusuma, and K. Ramchandran. 2002. Distributed compression in a dense microsensor network. *IEEE Signal Processing Magazine* 19, 2 (March 2002), 51–60. DOI:http://dx.doi.org/10.1109/79.985684 (Cited on page 54.)

N. Praveen, N. Karanth, and M. S. Megha. 2014. Sign language interpreter using a smart glove. In *Advances in Electronics, Computers and Communications (ICAECC), 2014 International Conference on*. 1–5. DOI:http://dx.doi.org/10.1109/ICAECC.2014.7002401 (Cited on page 8.)

Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press. (Cited on pages 11, 12, and 22.)

Gang Ren and Eamonn O'Neill. 2013. Freehand Gestural Text Entry for Interactive TV. In *Proceedings of the 11th European Conference on Interactive TV and Video (EuroITV '13)*. ACM, New York, NY, USA, 121–130. DOI:http://dx.doi.org/10.1145/2465958.2465966 (Cited on page 7.)

Ryan Rifkin and Ross Lippert. 2007. Notes on Regularized Least Squares. (05 2007). (Cited on page 20.)

C. J. Van Rijsbergen. 1979. *Information Retrieval* (2nd ed.). Butterworth-Heinemann, Newton, MA, USA. (Cited on page 18.)

Alina Roitberg, Nikhil Somani, Alexander Perzylo, Markus Rickert, and Alois Knoll. 2015. Multimodal Human Activity Recognition for Industrial Manufacturing Processes in Robotic Workcells. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction (ICMI '15)*. ACM, New York, NY, USA, 259–266. DOI:http://dx.doi.org/10.1145/2818346.2820738 (Cited on page 7.)

Lior Rokach. 2005. *Ensemble Methods for Classifiers*. Springer US, Boston, MA, 957–980. DOI:http://dx.doi.org/10.1007/0-387-25465-X_45 (Cited on pages 23 and 24.)

M. Romaszewski, P. Glomb, and P. Gawron. 2014. Natural hand gestures for human identification in a Human-Computer Interface. In *Image Processing Theory, Tools and Applications (IPTA), 2014 4th International Conference on*. 1–6. DOI:http://dx.doi.org/10.1109/IPTA.2014.7001997 (Cited on pages 9 and 23.)

Stuart J. Russell and Peter Norvig. 2010. *Artificial intelligence : a modern approach*. Prentice Hall, Upper Saddle River, N.J. (Cited on pages 13, 22, 23, and 51.)

Albert Samà, Daniel Rodríguez-Martín, Carlos Pérez-López, Andreu Català, Sheila Alcaine, Berta Mestre, Anna Prats, M. Cruz Crespo, and Àngels Bayés. 2018. Determining the optimal features in freezing of gait detection through a single waist accelerometer in home environments. *Pattern Recognition Letters* 105 (2018), 135 – 143. DOI:http://dx.doi.org/https://doi.org/10.1016/j.patrec.2017.05.009 Machine Learning and Applications in Artificial Intelligence. (Cited on page 15.)

Robert E Schapire. 2003. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*. Springer, 149–171. (Cited on page 24.)

Terrence J. Sejnowski and Charles R. Rosenberg. 1987. Parallel Networks that Learn to Pronounce English Text. (1987). (Cited on page 45.)

Sougata Sen, Vigneshwaran Subbaraju, Archan Misra, Rajesh Krishna Balan, and Youngki Lee. 2015. The case for smartwatch-based diet monitoring. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*. 585–590. DOI:http://dx.doi.org/10.1109/PERCOMW.2015.7134103 (Cited on page 7.)

Muhammad Shoaib, Stephan Bosch, Hans Scholten, Paul J.M. Havinga, and Ozlem Durmaz Incel. 2015. Towards detection of bad habits by fusing smartphone and smartwatch sensors. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*. 591–596. DOI:http://dx.doi.org/10.1109/PERCOMW.2015.7134104 (Cited on page 7.)

Steven W. Smith. 1997. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA, USA. (Cited on page 26.)

Bruce Spencer, Feras Al-Obeidat, and Omar Alfandi. 2017. Accurately forecasting temperatures in smart buildings using fewer sensors. *Personal and Ubiquitous Computing* (16 Dec 2017). DOI:http://dx.doi.org/10.1007/s00779-017-1103-4 (Cited on pages 53 and 54.)

T. Stiefmeier, D. Roggen, G. Ogris, P. Lukowicz, and P. Lukowicz. 2008. Wearable Activity Tracking in Car Manufacturing. *Pervasive Computing, IEEE* 7, 2 (April 2008), 42–50. DOI:http://dx.doi.org/10.1109/MPRV.2008.40 (Cited on page 7.)

Jae Hong Suh, Soundar R.T. Kumara, and Shreesh P. Mysore. 1999. Machinery Fault Diagnosis and Prognosis: Application of Advanced Signal Processing Techniques. *CIRP Annals - Manufacturing Technology* 48, 1 (1999), 317 – 320. DOI:http://dx.doi.org/https://doi.org/10.1016/S0007-8506(07)63192-8 (Cited on pages 14 and 17.)

Tomoichi Takahashi and Fumio Kishino. 1991. Hand Gesture Coding Based on Experiments Using a Hand Gesture Interface Device. *SIGCHI Bull.* 23, 2 (March 1991), 67–74. DOI:http://dx.doi.org/10.1145/122488.122499 (Cited on page 5.)

Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu. 2002. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences* 99, 10 (2002), 6567–6572. DOI:http://dx.doi.org/10.1073/pnas.082099299 (Cited on page 22.)

Tsung-Han Tsai, Chih-Chi Huang, and Kung-Long Zhang. 2015. Embedded virtual mouse system by using hand gesture recognition. In *Consumer Electronics - Taiwan (ICCE-TW), 2015 IEEE International Conference on*. 352–353. DOI:http://dx.doi.org/10.1109/ICCE-TW.2015.7216939 (Cited on page 7.)

Wallace Ugulino and Hugo Fuks. 2015. Landmark Identification with Wearables for Supporting Spatial Awareness by Blind Persons. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 63–74. DOI:http://dx.doi.org/10.1145/2750858.2807541 (Cited on page 7.)

Jason Van Hulse, Taghi M. Khoshgoftaar, Amri Napolitano, and Randall Wald. 2012. Threshold-based feature selection techniques for high-dimensional bioinformatics data. *Network Modeling Analysis in Health Informatics and Bioinformatics* 1, 1 (01 Jun 2012), 47–61. DOI:http://dx.doi.org/10.1007/s13721-012-0006-6 (Cited on page 17.)

Wouter Van Vlaenderen, Jens Brulmans, Jo Vermeulen, and Johannes Schöning. 2015. WatchMe: A Novel Input Method Combining a Smartwatch and Bimanual Interaction. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '15)*. ACM, New York, NY, USA, 2091–2095. DOI:http://dx.doi.org/10.1145/2702613.2732789 (Cited on pages 8 and 9.)

V. Vapnik and A. Lerner. 1963. Pattern Recognition using Generalized Portrait Method. *Automation and Remote Control* 24 (1963). (Cited on pages 11, 12, and 21.)

P. Vepakomma, D. De, S. K. Das, and S. Bhansali. 2015. A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities. In *2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*. 1–6. DOI:http://dx.doi.org/10.1109/BSN.2015.7299406 (Cited on page 11.)

Juan Pablo Wachs, Mathias Kölsch, Helman Stern, and Yael Edan. 2011. Vision-based Hand-gesture Applications. *Commun. ACM* 54, 2 (Feb. 2011), 60–71. DOI:http://dx.doi.org/10.1145/1897816.1897838 (Cited on pages 7 and 9.)

A. Waldeyer, F. Anderhuber, F. Pera, and J. Streicher. 2012. *Waldeyer - Anatomie des Menschen: Lehrbuch und Atlas in einem Band ; [44 Tabellen]*. De Gruyter. https://books.google.at/books?id=QCtOygAACAAJ (Cited on page 3.)

J.A. Ward, P. Lukowicz, G. Troster, and T.E. Starner. 2006. Activity Recognition of Assembly Tasks Using Body-Worn Microphones and Accelerometers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 28, 10 (Oct 2006), 1553–1567. DOI:http://dx.doi.org/10.1109/TPAMI.2006.197 (Cited on pages 7, 16, and 48.)

Jamie A. Ward. 2006. *Activity monitoring: Continuous recognition and performance evaluation*. Ph.D. Dissertation. Swiss Federal Institute of Technology (ETH) Zürich. DOI:http://dx.doi.org/10.3929/ethz-a-005228941 (Cited on page 48.)

J. Weissmann and R. Salomon. 1999. Gesture recognition for virtual reality applications using data gloves and neural networks. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, Vol. 3. 2043–2046 vol.3. (Cited on page 7.)

Alan Wexelblat. 1995. An Approach to Natural Gesture in Virtual Environments. *ACM Trans. Comput.-Hum. Interact.* 2, 3 (Sept. 1995), 179–200. DOI:http://dx.doi.org/10.1145/210079.210080 (Cited on page 5.)

Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. 2009. User-defined Gestures for Surface Computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1083–1092. DOI:http://dx.doi.org/10.1145/1518701.1518866 (Cited on page 5.)

Curtis E. Woodcock and Alan H. Strahler. 1987. The factor of scale in remote sensing. *Remote Sensing of Environment* 21, 3 (1987), 311 – 332. DOI:http://dx.doi.org/https://doi.org/10.1016/0034-4257(87)90015-0 (Cited on page 54.)

Oliver J. Woodman. 2007. *An introduction to inertial navigation*. Technical Report UCAM-CL-TR-696. University of Cambridge, Computer Laboratory. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf (Cited on pages 25 and 26.)

Chao Xu, Parth H. Pathak, and Prasant Mohapatra. 2015. Finger-writing with Smartwatch: A Case for Finger and Hand Gesture Recognition Using Smartwatch. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile '15)*. ACM, New York, NY, USA, 9–14. DOI:http://dx.doi.org/10.1145/2699343.2699350 (Cited on pages 8, 9, and 22.)

Deyou Xu. 2006. A Neural Network Approach for Hand Gesture Recognition in Virtual Reality Driving Training System of SPG. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, Vol. 3. 519–522. (Cited on pages 7 and 9.)

Lei Xu, Pingfan Yan, and Tong Chang. 1988. Best first strategy for feature selection. In *[1988 Proceedings] 9th International Conference on Pattern Recognition*. 706–708 vol.2. DOI:http://dx.doi.org/10.1109/ICPR.1988.28334 (Cited on page 17.)

Wei Xu. 2011. Towards Optimal One Pass Large Scale Learning with Averaged Stochastic Gradient Descent. *CoRR* abs/1107.2490 (2011). http://arxiv.org/abs/1107.2490 (Cited on pages 11, 12, and 20.)

Ke Yan and David Zhang. 2015. Feature selection and analysis on correlated gas sensor data with recursive feature elimination. *Sensors and Actuators B: Chemical* 212 (2015), 353 – 363. DOI:http://dx.doi.org/https://doi.org/10.1016/j.snb.2015.02.025 (Cited on pages 17 and 52.)

J. Yang, W. Sheng, and G. Yang. 2018. Dynamic Gesture Recognition Algorithm based on ROI and CNN for Social Robots. In *2018 13th World Congress on Intelligent Control and Automation (WCICA)*. 389–394. DOI:http://dx.doi.org/10.1109/WCICA.2018.8630497 (Cited on page 8.)

Y. Yin and R. Davis. 2014. Real-time continuous gesture recognition for natural human-computer interaction. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 113–120. DOI:http://dx.doi.org/10.1109/VLHCC.2014.6883032 (Cited on page 8.)

Mattia Zanon, Giovanni Sparacino, Andrea Facchinetti, Mark Talary, Andreas Caduff, and Claudio Cobelli. 2013. Regularised Model Identification Improves Accuracy of Multisensor Systems for Noninvasive Continuous Glucose Monitoring in Diabetes Management. *J. Applied Mathematics* 2013 (2013), 793869:1–793869:10. DOI:http://dx.doi.org/10.1155/2013/793869 (Cited on page 54.)

P. Zappi, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Troster. 2007. Activity recognition from on-body sensors by classifier fusion: sensor scalability and robustness. In *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*. 281–286. DOI:http://dx.doi.org/10.1109/ISSNIP.2007.4496857 (Cited on page 7.)

J. Zhang and I. Mani. 2003. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*. (Cited on page 72.)

Tong Zhang. 2004. Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms. In *ICML 2004: PROCEEDINGS OF THE TWENTY-FIRST INTERNATIONAL CONFERENCE ON MACHINE LEARNING. OMNIPRESS*. 919–926. (Cited on pages 11, 12, and 20.)

Xu Zhang, Xiang Chen, Wen-hui Wang, Ji-hai Yang, Vuokko Lantz, and Kong-qiao Wang. 2009. Hand Gesture Recognition and Virtual Game Control Based on 3D Accelerometer and EMG Sensors. In *Proceedings of the 14th International Conference on Intelligent User Interfaces (IUI '09)*. ACM, New York, NY, USA, 401–406. DOI:http://dx.doi.org/10.1145/1502650.1502708 (Cited on pages 7 and 9.)

Yixin Zhao, Parth H. Pathak, Chao Xu, and Prasant Mohapatra. 2015. Demo: Finger and Hand Gesture Recognition Using Smartwatch. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '15)*. ACM, New York, NY, USA, 471–471. DOI:http://dx.doi.org/10.1145/2742647.2745922 (Cited on pages 9, 21, and 23.)

F. Zhou and F. De la Torre. 2012. Generalized time warping for multi-modal alignment of human motion. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 1282–1289. DOI:http://dx.doi.org/10.1109/CVPR.2012.6247812 (Cited on page 10.)

Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the Elastic Net. *Journal of the Royal Statistical Society, Series B* 67 (2005), 301–320. (Cited on pages 20 and 21.)

Dan Zwillinger and Stephen Kokoska. 2000. *CRC Standard Probability and Statistics Tables and Formulae*. CRC Press. DOI:http://dx.doi.org/10.1201/b16923 (Cited on page 17.)