Karin Mascher, BSc.

Soccer Activity Recognition using low-cost Inertial Measurement Units

Master's Thesis

to achieve the university degree of

Diplom-Ingenieurin

Master's degree programme: Geodäsie

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Manfred Wieser

Institute of Geodesy

Graz, August 2020

Affidavit

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

Date Signature

Acknowledgements

First of all, I would like to express my gratefulness to my supervisor, Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Manfred Wieser, for his guidance, support and reviewing this thesis with such care. I am incredibly thankful that he gave me the opportunity to write this thesis in the course of a research project as a student project assistant.

Many thanks to the team of the Working Group Navigation for always being cheerful and encouraging. In particular, I would like to thank Dipl.-Ing. Stefan Laller, who functioned as my co-supervisor. Throughout the entire work, he always had time for discussions and supported me during the measurement campaigns. Thanks as well to Dipl.-Ing. Christoph Schmied, who taught me the basics of Arduino programming and helped me to deal with some hard- and software problems in the data-acquisition. I would also like to thank Fachoberinspektor Bernd Mölg for constructing the custom-made shin guards.

I owe many thanks to c.c.com Moser GmbH for initiating the project and for giving me the possibility to be a part of it. Huge thanks to the Exercise Physiology, Training and Training Therapy Research Group, in particular MSc. Philipp Birnbaumer, for helping me with the data collection.

Moreover, I am incredibly grateful for having the fortune of great fellow students. Over the past years, strong friendships have been developed.

The greatest thanks are due but to my loving family, especially to my parents. They always support and encourage me in my life choices. I am very proud to be a part of this family.

Last but not least, I wish to express my sincere thanks to my beloved fiancé, David, for encouraging me to work hard on this thesis and for always being there for me.

Abstract

Soccer is one of the most popular sports in the world. Therefore, the performance analysis of soccer teams has become a central issue. Statistics like number of passes, shots or sprints give the trainer information about the game quality. Currently, available systems uses high-tech measurement technologies that are based on video analysis. Such systems are cost-expensive and, thus, only affordable for elite teams. Wearables for performance analysis represent an alternative at low cost compared to video-based analysis.

This thesis focuses on the recognition of simple, soccer-specific activities by utilisation of MEMS accelerometer data. The defined movements are *standing*, *walking*, *running*, *passing* and *shooting*. The data-acquisition takes place through two low-cost inertial measurement units: one sensor is mounted on each shin guard. The diverse analytics are based on Machine Learning techniques. Suitable features were selected in the time- and frequency domain. Feature in the frequency-domain are based on wavelet analysis. A comparison among dimensionality reduction techniques was made, namely Principal Component Analysis versus Linear Discriminant Analysis.

A classification scheme for recognising the above-mentioned activities was developed and tested on real data. The evaluation of the performance showed that Logistic Regression, Support Vector Machine and Random Forest in combination with the Linear Discriminant Analysis outperformed the other approaches. All shots and passes were detected. Only a few instances were misclassified. Those faults took place in the transition areas between standing, walking and running. Thus, no fundamental errors were made. The best overall performance provides Logistic Regression and Random Forest with an overall accuracy of 95%, an macro-precision score of 97% and an macro-recall score of 96%. Last but not least, it has been shown that the proposed classification scheme is real-time capable.

Kurzfassung

Fußball zählt zu einer der populärsten Sportarten der Welt. Folglich sind Leistungsanalyse von Fußballteams zu einem zentralen Thema geworden. Verschiedene Parameter, wie die Anzahl der Pässe, Schüsse oder Sprints können herangezogen werden um die Spielqualität zu evaluieren. Die derzeitigen Systeme basieren auf Videoanalysen, sind kostenintensiv und nur für Profiteams leistbar. Die Verwendung von Wearable für Leistungsanalysen bietet eine kostengünstige Alternative zur videobasierten Analyse.

Diese Masterarbeit behandelt die Erkennung simpler, fußballspezifischer Aktivitäten basierend auf MEMS-Beschleunigungsdaten. Die definierten Bewegungen sind *Stehen*, *Gehen*, *Laufen*, *Passen* und *Schießen*. Die Datenerfassung erfolgt mithilfe von zwei inertialen Messeinheiten, die jeweils an einem Schienbeinschützer angebracht sind. Die Analyseverfahren basieren auf Techniken des maschinellen Lernens. Features wurden im Zeit- und Frequenzbereich (Wavelet-Analyse) ausgewählt. Zwei verschiedene Algorithmen zur Dimensionsreduktion wurden miteinander verglichen, nämlich die Hauptkomponentenanalyse und die linearen Diskriminanzanalyse.

Ein Klassifikationsschema zur Erkennung der oben genannten Aktivitäten wurde entwickelt und an realen Daten getestet. Die Auswertung zeigte, dass Logistic Regression, Support Vektor Maschine und Random Forest in Kombination mit der linearen Diskriminanzanalyse die anderen Ansätze übertrafen. Alle Schüsse und Pässe wurden erkannt. Nur einige wenige Instanzen wurden falsch klassifiziert. Diese Fehlklassifizierungen fanden in den Übergangsbereichen zwischen Stehen, Gehen und Laufen statt. Es wurden also keine grundlegenden Fehler gemacht. Die beste Gesamtleistung lieferten Logistic Regression und Random Forest mit einer Gesamtgenauigkeit von 95%, einer Macro-Precision von 97% und einem Macro-Recall von 96%. Es wurde gezeigt, dass das vorgeschlagene Klassifikationsschema echtzeitfähig ist.

Acronyms

ADC	Analog-to-Digital Converter
ANN	Artificial Neural Network
AR	Activity Recognition
AUC	Area Under the Curve
BF	Body Frame
BGD	Batch Gradient Descent
CART	Classification and Regression Tree
CWT	Continous Wavelet Transform
DT	Decision Tree
DWT	Discrete Wavelet Transform
FN	false negative
FP	false positive
FPR	false positive rate
FT	Fourier Transform
FWT	Fast Wavelet Transform
GD	Gradient Descent
GNSS	Global Navigation Satellite System
I2C	Inter-Integrated Circuit
IF	Inertial Frame
IID	independent and identically distributed
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IQR	Interquartile Range
LD	Linear Discriminant
LDA	Linear Discriminant Analysis
MCU	Microcontroller Unit
MEMS	Micro-Electro-Mechanical System

ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multilayer Perceptron
OvO	One-Versus-One
OvR	One-Versus-Rest
PC	Principal Component
PCA	Principal Component Analysis
PCB	Printed Circuit Board
PTP	Precision Time Protocol
QFN	Quad Flat No-lead package
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RMS	Root Mean Square
ROC	Receiver Operating Characteristic
SELU	Scaled Exponential Linear Unit
SGD	Stochastic Gradient Descent
SMA	Signal Magnitude Area
STFT	Short-Time Fourier Transform
SVM	Support Vector Machine
t-SNE	t-distributed Stochastic Neighbor Embedding
TLU	Threshold Logic Unit
TN	true negative
TNR	true negative rate
TP	true positive
TPR	true positive rate
WT	Wavelet Transform

Contents

List of Figures

List of Tables

1. Introduction

This chapter gives a short overview of the master's thesis on *Soccer Activity Recognition using low-cost Inertial Measurement Units*. First, the fundamental motivation of this thesis will be discussed. Second, the research aim will be explained. And third, the structure of this thesis is stated.

1.1. Motivation

Trainers and soccer players are interesting in profiling the physical fitness and the movements of playing during a soccer match. The number of sprints, passes and shots gives information of a player's performance and the collective activity within the team. Based on the gained information, various strategies and tactics can be developed for a soccer match. The systems, that are currently available, are based on cost-expensive video-analysis. As an example, *skills.lab* at Graz can be named. *skills.lab* (Anton Paar SportsTec GmbH) is a high-tech training facility for soccer players. Based on state-of-the-art measurement technologies, specific game situations can be simulated and individually adapted to the respective player [1]. However, a specific infrastructure is required, along with high costs. Thus, such performance analysis are only affordable for professional soccer teams.

The usage of smartphones and commercial wearables, such as smart watches, to profile the activity and fitness of a person has gained popularity in the last decades. According to statista [2], a total of around 527 million wearables should be sold in 2024. In particular, the sports sector is interested in cost-efficient ways to monitor an athlete's performance. Wearables, containing low-cost sensors, represent a promising alternative to video-based analysis.

Such smart sport gadgets comprise, for example, Global Navigation Satellite System (GNSS) sensors, inertial sensors and heart rate monitors.

The low-cost inertial sensors can be utilised for activity recognition that can help hobby teams to improve the quality of their moves. Since wrist-worn devices are at not allowed during a soccer match, it is desirable to embed the sensors in the player's equipment. Schuldhaus *et al.* [3] proposed an inertial-based approach to detect passes and shots during a soccer game. The inertial sensors were located in a hollow of the soccer shoe. The study showed that it is, in general, possible to use low-cost inertial sensors for pass/shot classification. An overall mean classification rate of 84.2% was reached. This study represent one development stage for the production of commercial smart sport gadgets.

However, further investigations revealed that currently no low-cost and smart soccer equipment is available on market. Thus, the motivation to develop a smart sports gadget for soccer players at low cost is high.

1.2. Research Aims

This thesis aims to develop a suitable classification algorithm for the recognition of soccer-specified activities using shin guards equipped with inertial measurement units. Such movements are standing, walking, running, passing and shooting. The data needed for this task is acquired from a Micro-Electro-Mechanical System (MEMS) accelerometer. Different Machine Learning algorithms will be evaluated and compared with each other. Various feature extraction methods and their effect on the algorithms will be analysed. The algorithms will be tested in simple, real applications to determine, which Machine Learning concept is best suited for this specific application. This thesis shall serve as a feasibility study and should indicate whether it is possible to detect soccer-specific activities based on low-cost inertial sensors.

1.3. Outline

This thesis consists of three parts. Part I corresponds to the theoretical foundations. The theoretical part is divided into three chapters that give a brief introduction in MEMS inertial sensors, in fundamentals of wavelet theory and in the basic concepts of Machine Learning. Part II describes the development process of the test setup as well as the developed classification scheme for soccer activity recognition. That includes a brief description of the activities to detect and the used sensor. The data collection, including the test setup as well as the sensor calibration, will be a topic; followed by a wavelet analysis of the collected data. The final chapter of this part introduces the developed activity recognition scheme. Part III deals with the model evaluation and the results. It comprises the evaluation of the trained Machine Learning algorithms and their suitability in real applications. Finally, the drawn conclusion are summarised and looked at the future fields of research.

Part I
Theoretical Foundations

2. MEMS Inertial Sensors

Inertial sensors comprise accelerometers and gyroscopes, which measure specific forces and angular rates, respectively [4]. Usually, three mutually orthogonal accelerometers and three mutually orthogonal gyroscopes are attached on a platform, also known as *Inertial Measurement Unit (IMU)*. An IMU is a principal component of an Inertial Navigation System (INS). An INS is capable of determining an independent navigation solution (position, velocity and attitude) of a moving object relative to a known reference system. However, navigation is not the only application field of inertial sensors. With the introduction of MEMS technology, the scope of application also includes Activity Recognition (AR) based on wearables [5].

A Micro-Electro-Mechanical System (MEMS) is associated with any sensor that is produced by microelectronic fabrication techniques [6]. It refers in general to the sensor's architecture and construction [7]. These techniques enable the production of miniatured forms of various sensors at a low-cost level and in large quantity. The integration of microelectronic circuits allows to record physical parameters like accelerations [6] or angular rates. A low-cost IMU consists of a MEMS accelerometer and a MEMS gyroscope. They are usually used in combination with a MEMS magnetometer. Magnetometers measure the magnetic field strength [7] in units of microteslas (μT). A magnetometer gives the direction to the magnetic poles. Together with the accelerometer and gyroscope, the magnetometer helps to stabilise the direction determination in navigational tasks.

Since this thesis deals with AR based on MEMS accelerometer data, the accelerometer will be explained in more detail. The gyroscope, however, will be discussed merely superficially.

2.1. Accelerometer

The working principle of an accelerometer is to measure the forces, which have an effect on a proof mass m [8]. Figure 2.1a shows a spring accelerometer, where the sensitive axis is aligned with the horizontal axis. The accelerometer measures the displacement of the proof mass that is proportional to the acting acceleration. Supposing that no acceleration appears, the proof mass is at equilibrium. When an acceleration occurs to the right, the proof mass is pushed to the left side relative to its casing. The output is a positive acceleration. In the event that the sensitive axis is aligned vertically with the gravitational field (Figure 2.1b), the proof mass is pulled downwards and also outputs a positive acceleration, which equals the gravitational acceleration.

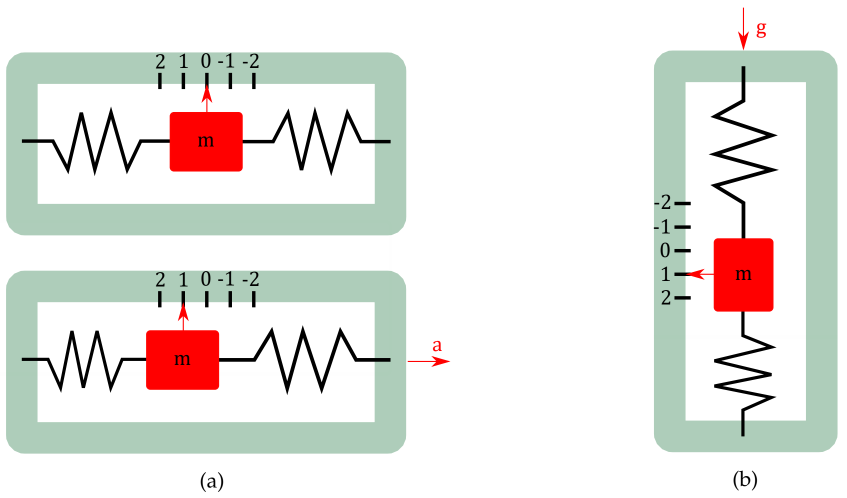


Figure 2.1.: Working principle of an accelerometer based on Noureldin *et al.* [8]: Accelerometer (a) in equilibrium position without an acting acceleration and with an acting acceleration a, (b) with gravitational acceleration g acting on it.

According to the equivalence principle, it is not possible to distinguish inertial mass from gravitational mass. Therefore, the output of an accelerometer consists of the acceleration of the object superimposed with the gravitational acceleration concerning the inertia space and is called *specific force* [8]:

$$f = a - g, \tag{2.1}$$

where a is denoted as the acceleration with respect to the inertial frame (Chapter 2.5) and g is the gravitational acceleration.

The displacement of a proof mass due to acceleration of a MEMS accelerometer is measured as a change in voltage by usage of different transduction principles. Such transduction principles can be capacitive, piezoresistive, tunneling, resonant, optical, an so on [9]. This displacement is directly proportional to the output voltage. Beside the transduction principle, it can be also differenced between the sensor architectures [4]:

- **Open-loop**: Actual movement of the proof mass is measured by a suitable pickoff mechanism like a mechanical scale, a capacity pickoff, a piezo-electric pickoff or an optical detector.

- **Closed-loop**: Determines the force, which is necessary to maintain the proof mass in its original state by using an electric or magnetic force generator, called *forcer*. Sensors, which operate in closed-loop mode show, in general, a better linearity compared to open-loop mode.

2.1.1. Types of MEMS Accelerometers

MEMS accelerometers can be found in a wide range of applications such as in air bags of cars or in smartphones. The different types of MEMS accelerometers differ in resolution, range, bandwidth, signal transduction and in microelectronic fabrication [9]. For example, a high resolution as *micro-g* is required in space applications and inertial navigation. Accelerometers with medium and low resolution are relevant in air bags or seismometry. The most common commercially available MEMS accelerometers are using the capacitive or piezoresistive principle.

Capacitive accelerometers

Capacitive accelerometers have a range of sensing of 2 μg up to several g [9]. Areas of application are inertial navigation or microgravity detection. Figure 2.2 illustrates the simplified architecture of a capacitive accelerometer, which consists of some pairs of fixed electrodes and a flexible proof mass [10]. When there is no acceleration acting on the proof mass, the distances d_1 and d_2 are equidistant. Therefore, the two capacitors C_1 and C_2 measure no change. When an acceleration is applied, the moveable proof mass is shifted. As a result, the distances d_1 and d_2 are not equal so that a capacitance change is detected. The capacitance change is perceived as a change in voltage.

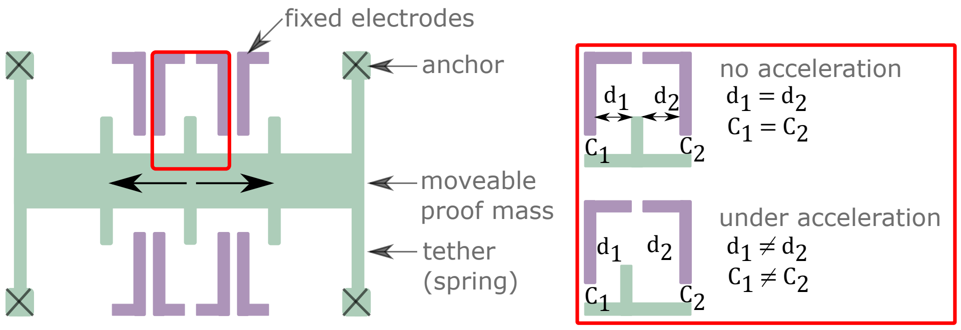


Figure 2.2.: Functional principle of capacitive accelerometers based on Amerini *et al.* [10].

The advantages of such sensors are high sensitivity, good noise performance, low drift and low-temperature sensitivity [9]. However, they are vulnerable to electromagnetic interferences.

Piezoresistive accelerometers

Piezoresistive accelerometers show a higher sensing range compared to capacitive accelerometers (0.001-50 g [9]). These sensors are commonly used for impact testing, such as in air bags. The principle of a piezoresistive accelerometer is that the displacement of the proof mass due to bending causes strains in the suspension beams. That results in a change in resistance detected by a piezoresistor. Such accelerometers are convincing regarding their simplicity in architecture, but they are highly sensitive to temperature.

A detailed comparison of various types of MEMS accelerometers can be found in Krishnan *et al.* [9].

2.2. Gyroscope

A gyroscope measures angular rates concerning an inertial frame and can be used to determine the attitude of an object. A gyroscope, in a mechanical sense, can be described as a rotating wheel or disk that maintain its orientation regardless of the movement of its axes. It uses the principle of conservation of angular momentum [10].

A MEMS gyroscope measures angular rates based on the Coriolis effect [10]. Considering a moving object, which experiences a rotation caused by angular velocity, the Coriolis force is acting perpendicular to its rotation axis and the trajectory. The Coriolis force affects the proof mass and causes it to vibrate. That leads to a capacitive change sensed by capacitive sensors.

2.3. Sensor Errors

Inertial sensor errors are divided into [11], [12]:

- **Stochastic errors**: Errors are random (random noise). They can be controlled by different signal processing methods.

- **Systematic errors**: Constant errors or errors that can be described as a function of the input and environmental factors. Common systematic errors are biases, scale factor errors, nonlinearity, asymmetry, dead zone, quantisation errors and hystereses. The main part of systematic errors are determined during a calibration process.

The major source of errors in an IMU are the bias and the scale factor error (Figure 2.3). The bias describes a constant offset, which is the difference between the output and the correct value. The scale factor error is a result from an incorrect sensitivity. For example, the output corresponds to 98% of the input. However, such errors can show small changes over time and day-to-day uncertainties. MEMS inertial sensors are usually uncalibrated and have large temperature-dependent biases and scale factor errors.

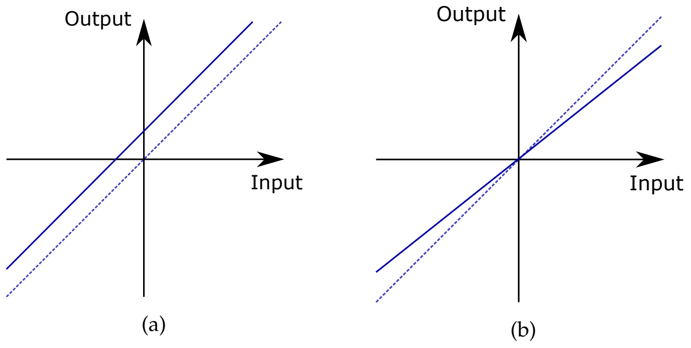


Figure 2.3.: Bias and scale factor error based on Grewal *et al.* [13]: (a) Bias, (b) Scale factor error. The dashed line represents an ideal system.

2.4. Sensor Calibration

This section deals with the calibration of a MEMS accelerometer. The calibration approach uses the fact that the axes of an accelerometer are sensitive to gravity (gravitation plus centrifugal force). The IMU is aligned to a horizontal platform so that the vertical axis follows the local plumb line. The sensor should only measure the vertical acceleration, which equals to ± 1 g ($\approx \pm 9.807$ m/s^2 at latitude for Graz), depending on the direction of the sensitive axis. However, since accelerometers suffer from sensor errors, the accelerometer measurements deviate from the actual values. With this knowledge, the biases and scale factor errors can be determined easily for each axis.

The calibration process is illustrated in Figure 2.4. Each pair represents the calibration measurements for a specific coordinate axis. Multiple measurements in static phases are done in all six sensor orientations and averaged. The measurements between the rotations are neglected.

The determination of the bias b_i for each axis ($i \in \{x, y, z\}$) is shown in Equation 2.2. The used formulas are based on Bolder Flight Systems [15]. f_i refers to the raw measurements of each pair.

$$b_i = \frac{\min(f_i) + \max(f_i)}{2} \tag{2.2}$$

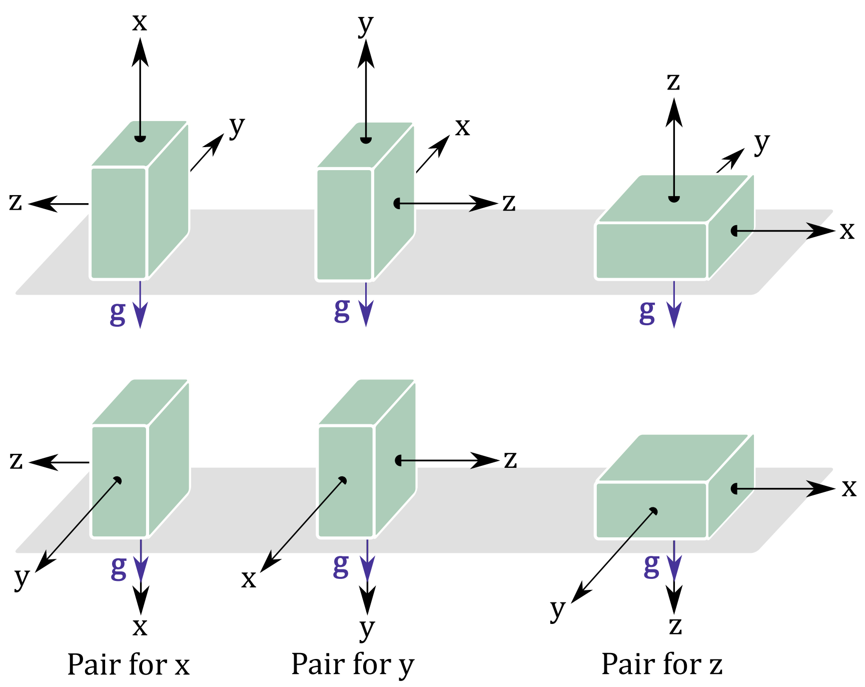


Figure 2.4.: Orientations of the accelerometer during the six calibration measurements based on Stančin and Tomažič [14]. The gravity acceleration g points downwards in the direction of the local plumb line.

The scale factor s_i is obtained by comparing the measurements with the acceleration of gravity g:

$$s_i = \frac{|\min(f_i)| + |\max(f_i)|}{2g}. \tag{2.3}$$

The gravity acceleration g is assumed a value of 9.807 $\mathrm{m/s^2}$ at current latitude (Graz). To achieve the corrected measurements f_i^{cal}, the calibration values are applied to the raw sensor data f_i as follows:

$$f_i^{\mathrm{cal}} = (f_i - b_i)\, s_i. \tag{2.4}$$

2.5. Coordinate Systems

The specific force is determined along the axis of a pre-defined reference frame, also termed as *Body Frame (BF)*. The gyroscope outputs angular rates between the *BF* and the *Inertial Frame (IF)* [4], [12].

Those coordinate systems are usually defined as a right-handed three-dimensional Cartesian systems. The inertial system corresponds to a reference system for which the laws of Newtonian mechanics are valid. In other words, such a system is non-accelerated and at rest or is in uniform motion, respectively. The IF refers to an inertially non-rotating frame. However, a strict realisation of an inertial system is impossible since the earth is orbiting around the sun in a non-steady motion.Thus, quasi-inertial systems are introduced. A quasi-inertial system is inertial to rotation but not to translation.

The BF is related to an object or an IMU itself. The origin of the coordinate system is situated within a pre-defined point of the INS. Its axes are typically aligned with the principle rotation axes of the object. The BF is described as an inertially rotating frame. In this application, the BF is associated with the sensor's coordinate system, which is attached to the shin guard.

3. Fundamentals of Wavelet Theory

The wavelet theory has become a popular "time–frequency decomposition tool for data analysis" (Addison [16, p. xv]). Therefore it has given many new opportunities in several applications like signal processing, image compression, filter design, pattern recognition, and so on [16]–[18]. As shown in Ayachi *et al.* [19] and Barralon *et al.* [20], wavelets can be used successfully for activity recognition. This chapter will give a brief introduction to wavelet theory.

3.1. Motivation for Wavelets

A traditional method for frequency analysis is represented by the Fourier Transform (FT) [21]:

$$S(f) = \int_{-\infty}^{\infty} s(t) e^{-i2\pi f t} dt, \tag{3.1}$$

where i is the imaginary unit, f the frequency and $s(t)$ represents the signal in time-domain. As a result, $S(f)$ gives the frequency spectrum of $s(t)$.

According to Addison [16] and Lee A. Barford *et al.* [17] this approach works well for stationary time series. For non-stationary/dynamic signals (spectral content changes over time) the FT fails to detect changes in the signal magnitude, frequency or phase sufficiently. However, most of the signals in real life are typically non-stationary. Furthermore, the FT lacks of temporal resolution (Figure 3.1). To solve this issue, the Short-Time Fourier

Transform (STFT) has been invented:

$$STFT(\tau, f) = \int_{-\infty}^{\infty} s(t)w(t-\tau)e^{-i2\pi ft}dt, \tag{3.2}$$

where w is the window and τ the time location.

The STFT is defined by applying the FT over a windowed signal. For short enough periods of time, the signal can be assumed to be stationary. If the window is fixed, then a uniform time resolution for all frequencies [22] is yielded. This leads to difficulties regarding the analysis of signals with wide bandwidth, which vary strongly with time [23]. The Wavelet Transform (WT) has been developed as an alternative to the STFT, and provides a

- high time resolution and low frequency resolution for high frequencies,
- low time resolution and high frequency resolution for low frequencies.

Therefore, the WT is well suited for analysing signals, which are characterised by being aperiodic, noisy, dynamic, and so on. Compared to the FT, the WT requires more complex base functions (wavelets) than sines and cosines. Besides, the analysis is done at multiple scales. The WT will be discussed more precisely in the following chapters. An illustration of the differences between the Fourier Transform, the Short-Time Fourier Transform and the Wavelet Transform can be seen in Figure 3.1.

3.2. Wavelet Transform

> "Wavelets are used to transform the signal under investigation into another representation which presents the signal information in a more useful form. This transformation of the signal is known as the *Wavelet Transform.* Mathematically speaking, the Wavelet Transform may be interpreted as a convolution of the signal with a wavelet function,.." (Addison [16, p. 2])

As demonstrated and explained in Addison [16], a wavelet can be understood as a scalable and sliding function with certain characteristics. In other terms: A wavelet can be shifted, stretched or squeezed as schematically

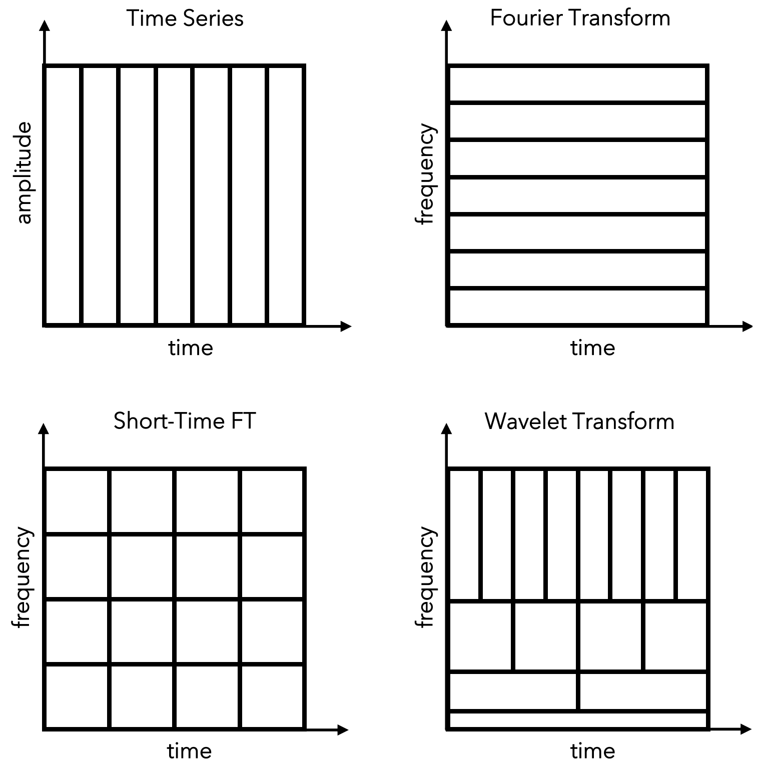


Figure 3.1.: Comparison of time series analysis, Fourier Transform, Short-Time Fourier Transform and Wavelet Transform in time-frequency plane based on Vandeput [24].

shown in Figure 3.2b and 3.2c. The mathematical details will be discussed in Chapter 3.3. Some common examples of wavelets on a single scale are shown in Figure 3.2a. As it can be seen, there exist different types of wavelets, called *wavelet families*. Each *family* has different features regarding complexity (shape), smoothness and compactness. A wavelet can be

- symmetric, near symmetric or asymmetric,
- orthogonal or not orthogonal,
- bi-orthogonal or not,
- real or complex.

Each *wavelet family* can be divided into subcategories and they differ in the number of vanishing moments and level of decomposition. These features play an important role in filter design (Chapter 3.6) [18]. A brief description of the characteristics, advantages and disadvantages of some common *wavelet families* is summarised by Chaudhary and Dhamija [25]. Which wavelet to choose, depends on the application needs.

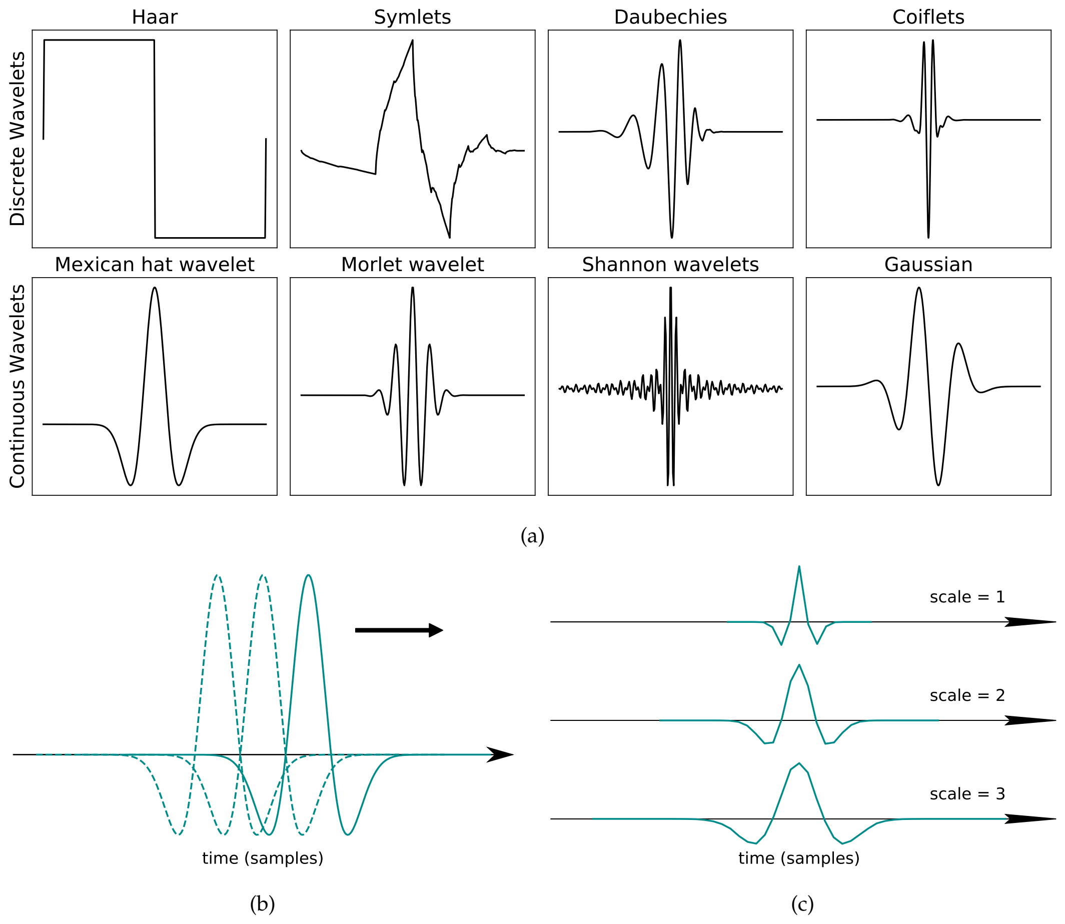


Figure 3.2.: Wavelets: (a) Examples for wavelets, (b) Shift (Location/Time) and (c) Scale. Illustrations based on Addison [16].

For the WT, the signal will be correlated with a wavelet. The correlation value is high, if the wavelet matches the form of the signal well. Conversely, this means that if the wavelet and the signal do not show similarities, the transform value is low. The transform value is computed for different scales and different locations. This leads to a two-dimensional transform plane (Figure 3.3). This transform plane can be generated via the Continous Wavelet Transform (CWT) or by using the discrete approach: the Discrete Wavelet Transform (DWT). Scales can be converted into pseudo-frequencies f_a at specific scales a: $f_a = \frac{f_c}{a}$, where f_c is the central frequency of the corresponding mother wavelet [26]. Consequently, small scales refer to high frequencies.

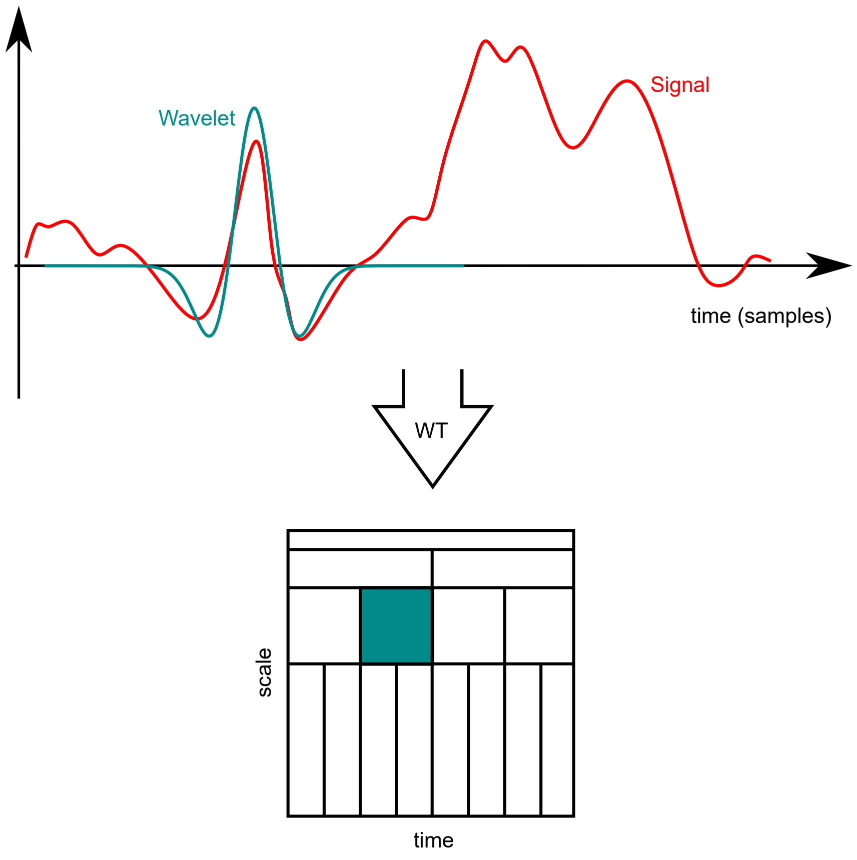


Figure 3.3.: Local matching of wavelet and signal based on Addison [16].

3.3. Requirements for Wavelets

A wavelet $\psi(t)$, also called *mother wavelet*, is a localised waveform, which satisfies the following conditions [16]:

1. The energy E of the wavelet must be finite:
$$E = \int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty. \tag{3.3}$$
This equations says, that the energy E is equal to the squared magnitude.
2. The second condition says:
$$C_g = \int_{0}^{\infty} \frac{|\hat{\psi}(f)|^2}{f} df < \infty, \tag{3.4}$$
where $\hat{\psi}(f)$ is Fourier transform of $\psi(t)$:
$$\hat{\psi}(f) = \int_{-\infty}^{\infty} \psi(t) e^{-i2\pi f t} dt. \tag{3.5}$$
The second condition is also called the *admissibility condition*. The admissibility condition implies: $\hat{\psi}(0) = 0$. Thus, a wavelet ψ must have a zero-mean. This condition leads to the fact that a wavelet can be interpreted as a bandpass filter.
3. In case of complex wavelets it applies, that $\hat{\psi}(f)$ must be real and vanishes for negative frequencies.

An example of the Wavelet Transform is the Mexican hat wavelet (Figure 3.2a). Mathematically, the Mexican hat wavelet is defined as [18]:
$$\psi(t) = \frac{2}{\pi^{1/4}\sqrt{3\sigma}} \left(\frac{t^2}{\sigma^2} - 1\right) \exp\left(-\frac{t^2}{\sigma^2}\right), \tag{3.6}$$
where σ is the standard deviation. The term $\frac{2}{\pi^{1/4}\sqrt{3\sigma}}$ is for normalization. The Mexican hat wavelet is the second derivative of a Gaussian distribution function
$$\exp\left(-t^2/\left(2\sigma^2\right)\right).$$

3.4. Continuous Wavelet Transform

As already mentioned in Chapter 3.2, a wavelet can be stretched and squeezed (dilation) or it can be shifted along the time axis (location). This two operations are now expressed via two parameters:

- dilation parameter a ($a > 0$) and
- location or translation parameter b ($b \in \mathbb{R}$).

Taking the Mexican hat wavelet as an example, the dilation (a_1, a_2) and translation parameters (b_1, b_2) are demonstrated in Figure 3.4.

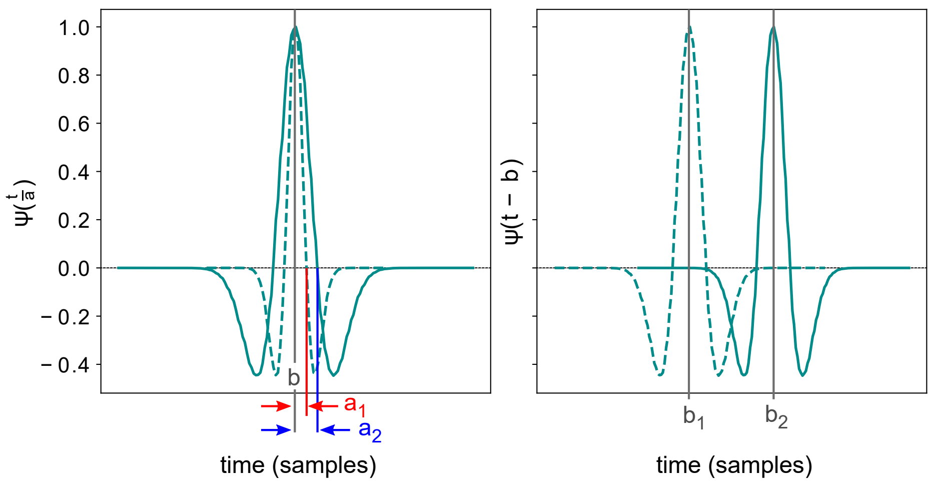


Figure 3.4.: Dilation (left) and translation (right) of a wavelet based on Addison [16].

Addison [16] introduces

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right), \tag{3.7}$$

which defines the scaled and shifted wavelet. The factor $a^{-1/2}$ ensures, that the energy for all scales will be preserved. As a consequence, the WT of a continuous signal $s(t)$ with the corresponding mother wavelet can be expressed as:

$$T(a,b) = \langle s, \psi_{a,b}\rangle = \int_{-\infty}^{\infty} s(t)\psi^*_{a,b}(t)dt. \tag{3.8}$$

$\psi^*_{a,b}$ symbolizes the complex conjugate of Equation 3.7.

3.5. Discrete Wavelet Transform

In the previous chapter, the wavelet function was expressed at scale a and location b (Equation 3.7). Now, discrete values for the scale and the location are needed. Discretising the dilation parameter

$$a = a_0^m$$

and the translation parameter

$$b = nb_0a_0^m,$$

the wavelet can be written as [16]:

$$\psi_{m,n}(t) = \frac{1}{\sqrt{a_0^m}}\psi\left(\frac{t - nb_0a_0^m}{a_0^m}\right) \tag{3.9}$$

where m is the control parameter for dilation, n the control parameter for translation, both must be integers. a_0 defines a fixed dilation step parameter set > 1 and b_0 refers to the location parameter > 0.

Consequently, the WT is written as

$$T_{m,n} = \langle s, \psi_{m,n} \rangle \,. \tag{3.10}$$

Via $\psi_{\mathrm{m,n}}(\mathrm{t})$, discrete samples are generated in the two-dimensional transform plane [27]:

- linear sampling along the location/time-axis,
- logarithmic sampling along the scale-axis.

3.5.1. Dyadic Grid

As follows, the discrete wavelet transform values $T_{m,n}$ are as well defined on the two-dimensional grid (m x n) [16]. In literature, the transform values are known as *wavelet coefficients* or *detail coefficients*. In practice, the values a_0 and b_0 are chosen as follows:

$$\begin{aligned} a_0 &= 2, \\ b_0 &= 1. \end{aligned} \tag{3.11}$$

Substituting 3.11 into Equation 3.9 leads to

$$\psi_{m,n}(t) = \frac{1}{\sqrt{2^m}}\psi\left(\frac{t - n2^m}{2^m}\right) = 2^{-m/2}\psi\left(2^{-m}t - n\right). \tag{3.12}$$

This representation is called *dyadic grid or octave wavelet* (power-of-two logarithmic scaling). Using such a grid, the wavelet coefficients can be written as

$$T_{m,n} = \int_{-\infty}^{\infty} s(t)\psi_{m,n}(t)dt. \tag{3.13}$$

To avoid redundant information in the wavelet coefficients $T_{m,n}$, the dyadic grid can be formed as an orthonormal wavelet basis (Equation 3.14).

$$\int_{-\infty}^{\infty} \psi_{m,n}(t)\psi_{m',n'}(t)\mathrm{d}t = \begin{cases} 1 & \text{if } m = m' \text{ and } n = n' \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

Using an orthonormal basis, the reconstruction of a signal $s(t)$ based on *discrete wavelet coefficients* can be formulated as

$$s(t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t). \tag{3.15}$$

There also exist *approximation coefficients*

$$S_{m,n} = \int_{-\infty}^{\infty} s(t)\phi_{m,n}(t)dt, \tag{3.16}$$

where

$$\phi_{m,n}(t) = 2^{-m/2}\phi\left(2^{-m}t - n\right), \tag{3.17}$$

$\phi_{m,n}(t)$ are called *scaling functions* and are related to orthonormal dyadic discrete wavelets. The scaling function shows the same structure as the wavelet (Equation 3.12). The scaling function has a smoothing effect on the signal. The following applies:

$$\int_{-\infty}^{\infty} \phi_{0,0}(t)dt = 1,$$

where $\phi_{0,0}(t) = \phi(t)$. $\phi(t)$ is also called *father wavelet* and characterises the basic wavelet scale. Another property of it is, that it is orthogonal regarding translations, but not regarding scaling. The approximation coefficients represent simply locally weighted averages of the signal scaled by the factor $2^{-m/2}$. A visualisation of a block scaling function is given in Figure 3.5.

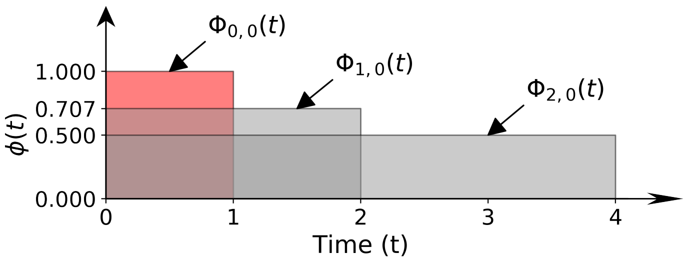


Figure 3.5.: Father wavelet (red) with two of its corresponding dilations based on Addison [16].

Now the signal can be represented through detail and approximation coefficients

$$s(t) = \sum_{n=-\infty}^{\infty} S_{m_0,n}\phi_{m_0,n}(t) + \sum_{m=-\infty}^{m_0}\sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t), \tag{3.18}$$

where m_0 is an arbitrary scale index. Equation 3.18 leads to the so called *multiresolution representation*.

3.5.2. Scaling Equation and Wavelet Equation

Another term related to the scaling function is the *scaling equation*:

$$\phi(t) = \sum_k c_k \phi(2t - k), \tag{3.19}$$

where $\phi(2t - k)$ is a contracted implementation of $\phi(t)$ (shifted by an integer step k). c_k are called *scaling coefficient*. $\phi(t)$ can be achieved by solving this two-scale difference equation (Equation 3.19). The scaling coefficients must satisfy

$$\sum_k c_k = 2. \tag{3.20}$$

Additionally, only taking into account wavelets with compact support[1] and being orthogonal

$$\sum_k c_k c_{k+2k'} = \begin{cases} 2 \text{ if } k' = 0 \\ 0 \text{ otherwise} \end{cases}, \tag{3.21}$$

the differencing of the corresponding *wavelet equation* can be written as

$$\psi(t) = \sum_k (-1)^k c_{N_k - 1 - k}\, \phi(2t - k), \tag{3.22}$$

where N_k is the number of scaling coefficients. As seen in Equation 3.22, the same coefficients appear reversed with alternate signs. Equation 3.21 also implies, that the square sum of the scaling coefficients must be 2. Defining

$$b_k = (-1)^k c_{N_k - 1 - k} \ldots (\sum_k b_k = 0)$$

yields to a different representation of the wavelet equation

$$\psi(t) = \sum_{k=0}^{N_k - 1} b_k \phi(2t - k). \tag{3.23}$$

[1]finite sequences of non-zero scaling coefficients

An example of an orthonormal wavelet represents the *Haar Wavelet*. The Haar wavelet is very popular due to its simple definition. Following, the solution of the Haar scaling equation

$$\phi(t) = \phi(2t) + \phi(2t-1) \tag{3.24}$$

leads to a single block pulse

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & \text{elsewhere} \end{cases} . \tag{3.25}$$

The solution of the Haar wavelet equation

$$\psi(t) = \phi(2t) - \phi(2t-1) \tag{3.26}$$

is carried out in an analogous manner:

$$\psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2} \\ -1 & \frac{1}{2} \leq t < 1 \\ 0 & \text{elsewhere} \end{cases} . \tag{3.27}$$

Both, the scaling and the wavelet function are shown in Figure 3.6.

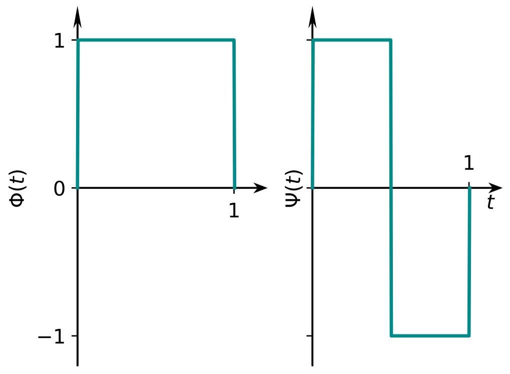


Figure 3.6.: Haar transform: scaling function (right), wavelet function (left)

3.6. Discrete Wavelet Transform as a Filter Bank

Mallat [28] introduces the idea of multiresolution signal decomposition, which yielded to the *Fast Wavelet Transform (FWT)*, also called *multiresolution algorithm* or *pyramid algorithm*. Multiresolution decomposition algorithm represents one part of the FWT, the second part is the reconstruction algorithm, which will not be further discussed. Basically, this *multiresolution decomposition algorithm* consists of two main recursive formulas [16]:

$$S_{m+1,n} = \frac{1}{\sqrt{2}} \sum_k c_k S_{m,2n+k} = \frac{1}{\sqrt{2}} \sum_k c_{k-2n} S_{m,k} \tag{3.28}$$

$$T_{m+1,n} = \frac{1}{\sqrt{2}} \sum_k b_k S_{m,2n+k} = \frac{1}{\sqrt{2}} \sum_k b_{k-2n} S_{m,k}. \tag{3.29}$$

In other words: if the approximation coefficients $S_{m,n}$ at an arbitrary scale $m = m_0$ are given, then it is possible to compute all approximation and detail coefficients larger than scale m_0.

From now on discrete input signals are taken into account. Starting at index scale $m = 0$, the discrete input signal is defined as

$$S_{0,n} = \int_{-\infty}^{\infty} s(t)\phi(t-n)dt. \tag{3.30}$$

From there it is possible to generate a wavelet multiresolution framework by applying Equation 3.28 and 3.29 several times. The implementation can be seen as a cascade of high-pass and low-pass filters:

- $(1/\sqrt{2})c_k$: low-pass filter (smoothing effect),
- $(1/\sqrt{2})b_k$: high-pass filter (signal details).

In practice, the discretely sampled signal s_t $[s(t_i) : i = 0, 1, 2, \ldots, N-1$, N length of finite signal $s_t]$ is equated with the approximation coefficients at scale $m = 0$: $S_{0,n}$. Strictly speaking, this is not correct due to the fact that $S_{0,n}$ represents the weighted average of a continuous signal $s(t)$ in the vicinity of n and not the signal itself. However, since $s(t)$ is usually not perfectly known: $S_{0,n} = s_t$.

Note that the discrete input signal $S_{0,n}$ is finite (length of the signal corresponds to an integer power of 2: $N = 2^M$). Thus, the range of the scales is $0 < m < M$ and of the translation $0 < n < 2^{M-m} - 1$, respectively. The decomposition of approximation coefficients into approximation and detail coefficients for several decomposition levels is sketched in Figure 3.7a. As a result, the discrete input signal is split in several detail coefficients and one single remaining approximation coefficient $S_{M,0}$, which is associated as the signal mean component.

Figure 3.7b shows one of the most common indexing methods for dyadic grid wavelet transform coefficients. This representation leads to the *discrete transform plot*. In this example, the discrete input signal contains 16 samples. The scale index $m = 1$ refers to the lowest scale on the grid and yields to a spacing of $2^1 = 2$. The discrete input signal would be located at $m = 0$. The DWT can be applied to any signal with a length of an integer power of 2, else the input signal has to be extended before computing the DWT, for example by adding Zeros at the beginning, as well as at the end of the signal. This technique is also known as *Zero Padding* [29].

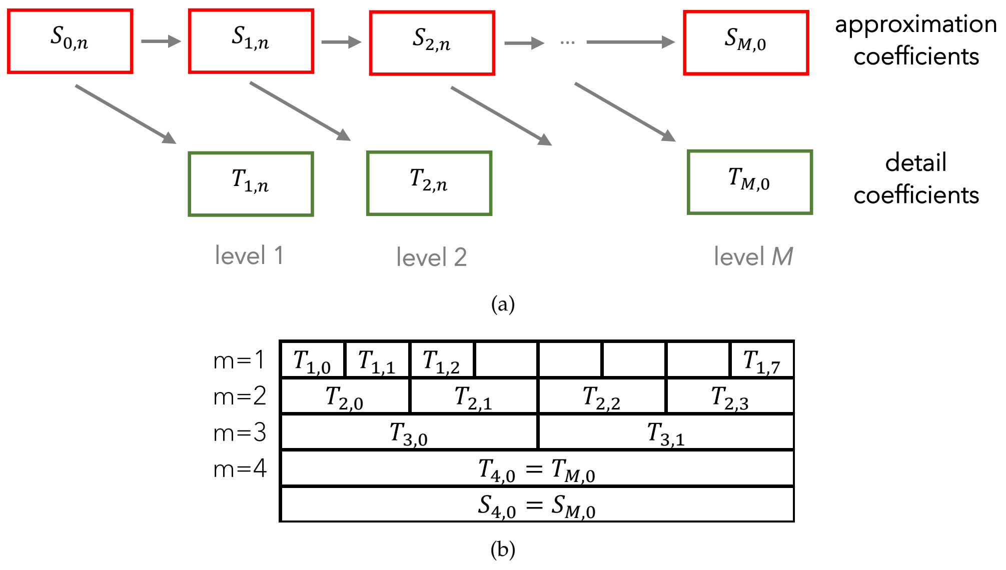


Figure 3.7.: Multiresolution signal decomposition: (a) Decomposition of approximation coefficients into approximation and detail coefficients, (b) Scale indexing: m. Illustrations are based on Addison [16].

The associated filtering process is displayed in the following block diagram (Figure 3.8). An approximation coefficient at a specific scale $S_{m,n}$ is convolved with a low-pass filter for approximation and with a high-pass filter for detail, subsampled ($2\downarrow$: keeping every second sample) to get the approximation and detail coefficients at the next scale $m+1$. This step is repeated to get the next coefficient pair at scale $m+2$. Note that each sub-band contains half the samples of the previous scale and as a result, the number of the computed consecutive coefficients stays the same as the number of input samples [22] yielding to the multiresolution framework (Figure 3.7b).

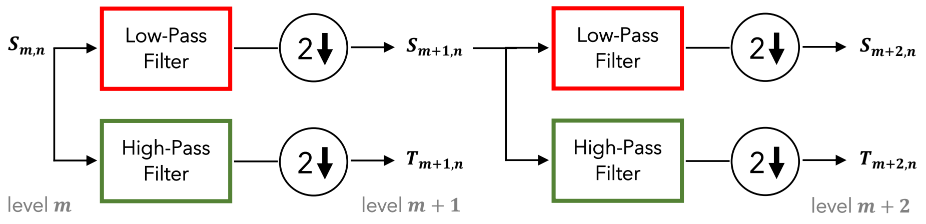


Figure 3.8.: Filtering scheme: approximation coefficients and detail coefficients at consecutive scales based on Addison [16].

The result is a pyramid-like structure of filters, called filter bank.

> "A filter bank is a set of bandpass filters with staggered center frequencies so that the whole frequency range is covered. The first and the last filter are lowpass and highpass filters, respectively." (Wanhammar and Yu [30, p. 259])

A filter bank has the purpose to divide the input signal into two or more sub-frequency-bands. So a filter bank is very useful, if only special frequency-bands are of interest. However, the Nyquist Rule must be taken into account: To not lose any information of a signal, it must be sampled at a frequency f_s greater or equal twice the highest frequency f_{max} [19]:

$$f_s \geq 2 f_{max}. \tag{3.31}$$

4. Fundamentals of Machine Learning

Machine Learning (ML) has become a widely used tool to solve problems, which are too complicated for traditional approaches or for which no known algorithm exists [31]. Therefore, many tasks, which deal with Activity Recognition (AR), are based on ML algorithms. This chapter discusses the fundamental concepts of ML, including its definition, the main challenges, and some well-known classification algorithms, like Logistic Regression, Support Vector Machine, Decision Tree or Neuronal Net.

A definition of ML is given by Bhavsar *et al.* [32, p. 283]:

> "Machine learning is a collection of methods that enable computers to automate data-driven model building and programming through a systematic discovery of statistically significant patterns in the available data."

ML consists of selecting an appropriate algorithm/model and training it. In this context, training means that the algorithm is fed with data from which it tries to learn, without being especially programmed. This process includes finding the ideal model parameters (Chapter 4.3). However, there exist several different types of learning. The basic types are categorised into [31]:

1. **Supervised Learning:**
   In supervised learning, the algorithms are trained on labelled data (data include input and output (solution)). For a given input and a known output, the algorithm tries to learn a basic rule, which maps the input to the desired output. Supervised learning is usually used to solve *Classification* or *Regression* tasks. Classification algorithms predict

classes, such as *True* or *False*. Regression algorithms predict numeric values such as house prices.

2. **Unsupervised Learning:**
   Algorithms are trained on unlabelled data and are very useful in the detection of hidden patterns. Therefore they are often found in clustering problem (k-Means), anomaly detection (Isolation Forest), visualisation algorithms or dimensionality reduction tasks (t-distributed Stochastic Neighbor Embedding (t-SNE)).

3. **Semisupervised Learning:**
   Algorithms are trained on partially labelled data since the generation of labelled data is a resource-intensive task in general.

4. **Reinforcement Learning:**
   The algorithm bases on a trial and error strategy. Learning a roboter how to walk is realised with the help of reinforcement learning.

4.1. Main Challenges

As already mentioned, ML consists of selecting an appropriate algorithm and training it. Therefore, the main challenge is to choose a “good” algorithm and feed it with “good” data. The following subsections are inspired by Géron [31].

4.1.1. Data Quality

The first requirement is to collect a sufficient amount of training data, depending on the complexity of the given task. The more data are available to train the algorithm, the better it can generalise new inputs, provided that the data is representative. In other terms: If the training set contains too few instances, it results in *sampling noise*. Large but skewed or imbalanced data sets can also be misrepresentative (*sampling bias*).

Noisy data with outliers also complicate a good performance of the algorithm. Therefore, it is very important to preprocess the data before feeding it into the ML system. This leads to another issue, namely the selection of relevant features (Chapter 4.1.2).

4.1.2. Feature Engineering

A *feature* is defined as a prominent attribute (variable) plus its value. Gender = "male", Age = "32", Height = "182 [cm]" are only some examples that can be features. Features can be extracted from data to simplify the classification or regression task. The success of a ML system relies significantly on finding good features. This process is called *Feature Engineering* and consists of

- **Feature selection**: exclusion of features with low or redundant information,
- **Feature extraction**: consideration of using feature combinations, which are more representative.

Variables, which are highly correlated, will not give out any additional information, quite contrary, it introduces additional noise in the model. In case that two variables would be perfectly correlated, both features would contain the same information. As a result, only one of them is needed to represent the data set. The same effect can be seen with uncorrelated features. Those features also provide noise and might even bias the model [33]. Only using relevant features does not only improve the model performance, but also reduces the computational time. Chandrashekar and Sahin [34] provide a good overview of different feature selection methods.

4.1.3. Different Feature Scales

Features with different scales might be problematic for the majority of ML algorithms. Due to the fact that models deal with number, they tend to weight larger values higher and small values lower, notwithstanding which dimension the feature has. Scaling the features is an important pre-processing step to make them comparable. That process is called *Feature Scaling*. One common way is *standardisation*.

Feature scaling through *standardisation* describes the process of rescaling the features to achieve properties of a standard normal distribution with zero-mean and unit-variance. The mean value from each feature is subtracted and divided by its standard deviation [35]:

$$x_i^{\text{new}} = \frac{x_i - \bar{x}}{\sigma}, \tag{4.1}$$

where x_i is the i^{th} element of the vector $\boldsymbol{x}$. The vector $\boldsymbol{x}$ contains multiple observations of one feature. $\bar{x}$ describes the mean of $\boldsymbol{x}$ and σ is the corresponding standard deviation.

4.1.4. Overfitting

In terms of ML, overfitting means that the model is trained too well. It works perfectly on the training set, but on new data, it performs poorly. An example is given in Figure 4.1. The model, represented by a higher degree polynomial, strongly overfits. In this case, a simple linear model performs better.

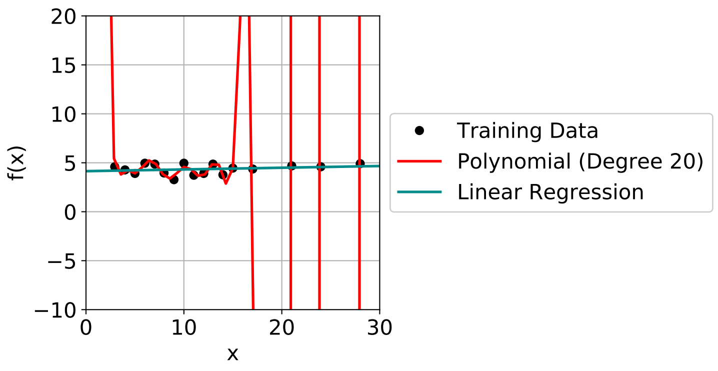


Figure 4.1.: Overfitting the training data

Overfitting appears when the model is too complex compared to the input data. Countermeasures to mitigate overfitting are collecting more data (if possible), constraining the model (*regularisation*) or reducing the number of features. However, each algorithm has different regularisation techniques (Chapter 4.3).

4.1.5. Underfitting

Underfitting happens when the model assumptions are too simple relative to the underlying data. It shows a negative impact on the training data, as well as on new data. Methods to avoid underfitting are the selection of a more complex model, better features (feature engineering) or reducing regularisation.

4.1.6. Hyperparameter Tuning

Hyperparameters are parameters, which have to be defined before the training process. A hyperparameter cannot be estimated from the input data, in contrast to model parameters. They can be seen as settings for the model to control its behaviour. A hyperparameter can be used for regularisation to prevent over- and underfitting. Another issue is that some hyperparameters are needed in the training process itself. Some models, like neuronal nets, need a pre-defined *learning rate* (Chapter 4.3.4). The finding of good hyperparameters represent a meta-optimisation task. However, since the optimal hyperparameters cannot be determined directly, they need to be searched. This process is called *hyperparameter tuning*. One approach is known as *Grid Search*. It searches through a manually-specified grid of hyperparameters and returns the best combination [36].

4.2. Training and Test Sets

Testing and validating is mandatory to verify how well the trained model generalises to new instances. In practice, the collected data is split into a

- Training set and a
- Test set.

Usually, the training data contain 80% of the whole data set, 20% are reserved for the test set [31]. As the terms indicate, the model is trained on the training set and tested on the withhold test set. The test set simulates new, unseen data. Both, the training and the test set can be used to evaluate the models' performance (Chapter 4.4). As already mentioned in Chapter 4.1.4, the model is overfitting, if it performs well on the training set, but gives a high error rate on the test set. When the model performs poor on the training set as well as on the test set, the model is underfitting (Chapter 4.1.5).
To ensure that the training and test sets are representative, it is necessary to sample the data so that each class (label) is equally represented (in percentage terms) in both sets. This approach is also called *stratified sampling*. Stratified sampling aims to reduce the *sampling bias*. Additionally, shuffle data guarantees that the samples are uniformly distributed (randomised).

4.3. Training Models

In this section some popular supervised classification algorithms in ML will be discussed:

- Logistic Regression
- Support Vector Machine
- Decision Tree
- Neural Network

The data set, given for classification, consists of m training data points $(\boldsymbol{x}^{(1)}, y^{(1)})$, $(\boldsymbol{x}^{(2)}, y^{(2)})$, ... , $(\boldsymbol{x}^{(m)}, y^{(m)})$, where $\boldsymbol{x}^{(i)}$ is the instance's feature vector and contains n features ($|\boldsymbol{x}| = n$). $y^{(i)}$ consists of the corresponding target class (label) [37]. The goal is to use those data points to train the algorithm (estimation of model parameters), so that it is capable of assigning new input data to a class.

There exist two types of classifiers:

- **Binary Classifiers**:
  Classification task distinguishes between two classes:
  $y^{(i)} \in \{0, 1\}$, $i = 1, \ldots, m$.
  In practice, $y^{(i)} = 1$ is termed as the positive class and $y^{(i)} = 0$ as the negative class.
- **Multiclass Classifiers**:
  Separation of more than two classes $K > 2$ is possible.

Some algorithms, which initially are intended to solve binary classification problems, can be turned into multiclass classifiers. Two strategies are worth to mention: One-Versus-Rest (OvR) and One-Versus-One (OvO) strategy [31]. Hence, the classification task is to classify an instance into K classes. The OvR strategy consists of training K binary classifiers: A classifier detects class 0, a classifier detects class 1, and so on. Each of them outputs a so-called *decision score* for a given input. The one classifier with the highest score defines the selected class. The second approach, OvO, deals with training several binary classifiers for all pairs of classes: The first classifier, for example, is trained to separate class 0 and class 1, the second one distinguishes between class 0 and class 2, and so on. The OvO strategy is more suitable for small data sets since it is faster to train many classifiers on small data sets than to run a few classifiers on large data sets.

4.3.1. Logistic Regression

Logistic Regression solves binary classification problems. It estimates the probability that a certain instance belongs to a class. Is the estimated probability higher than 50 % , the algorithm predicts that the instance is a member of the positive class ($y^{(i)} = 1$), else that it is part of the negative class ($y^{(i)} = 0$) [31]. Logistic Regression belongs to the so-called probabilistic discriminative models [38] and is a basic module of artificial neural networks (Chapter 4.3.4).

Logistic Regression model

In consideration of a single instance $(\boldsymbol{x}, y)$, the posterior probability of the positive class can be expressed in terms of the logistic sigmoid function, applied on a linear combination of the instance's feature vector $\boldsymbol{x}$ [31], [38]:

$$\hat{p} = p\left(y = 1|\boldsymbol{x}\right) = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma\left(\boldsymbol{\theta}^\top \boldsymbol{x}\right), \tag{4.2}$$

where

$$\begin{aligned}
\boldsymbol{\theta} &\text{ ... contains model parameters in a vecorized form:} \\
&\quad \text{bias term } \theta_0 \text{ plus feature weights } \theta_1, ...\theta_n \\
\boldsymbol{x} &\text{ ... instance's feature vector: } x_0, ..., x_n \text{ with } x_0 = 1 \\
h_{\boldsymbol{\theta}} &\text{ ... hypotesis function, using model parameters} \\
\sigma\left(\bullet\right) &\text{ ... logistic (sigmoid function)} \\
\boldsymbol{\theta}^\top \boldsymbol{x} &\text{ ... } \theta_0 x_0 + \cdots + \theta_n x_n \text{, called linear predictor [39]}
\end{aligned}$$

and with

$$p\left(y = 0|\boldsymbol{x}\right) = 1 - p\left(y = 1|\boldsymbol{x}\right). \tag{4.3}$$

The sigmoid function has the purpose to map real numbers from $(-\infty, \infty)$ to numbers in the range of $[0, 1]$. Constraining the range of the output is necessary since a binary output is required [39]. The sigmoid function is defined as

$$\sigma(t) = \frac{1}{1 + \exp(-t)}, \tag{4.4}$$

where t is called *logit*. As it can be seen in Figure 4.2, the sigmoid function is S-shaped and converges to zero for large negative values and advancing towards one for large positive values. Moreover, it is symmetric:

$$\sigma(-t) = 1 - \sigma(t). \tag{4.5}$$

The inverse of the logistic sigmoid (*logit function*)

$$t = \ln\left(\frac{\sigma}{1-\sigma}\right) \tag{4.6}$$

describes the log of the ratio of probabilities for the positive and negative class: $\ln\left(\frac{p(y=1|\boldsymbol{x})}{p(y=0|\boldsymbol{x})}\right)$ [38]. This ratio of probabilities is called *log odds*.

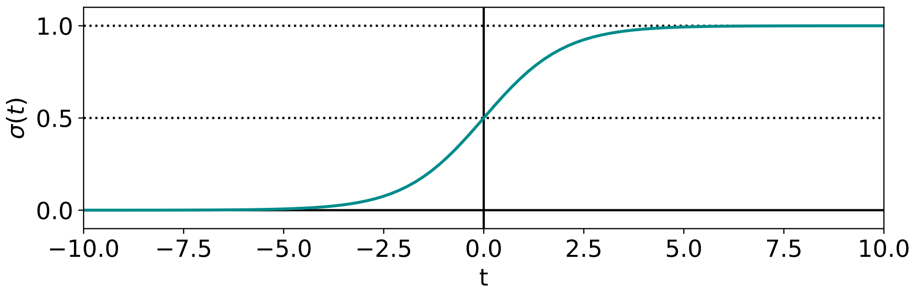


Figure 4.2.: Sigmoid function

In summary, a linear combination (plus its bias term) of features is taken, applied to a non-linear function, which constrains its output to a number between 0 and 1 (Equation 4.2). So, for the predicted class $\hat{y}$ the following applies:

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases} . \tag{4.7}$$

It implicates that the input is classified as the positive class whenever $\boldsymbol{\theta}^\top \boldsymbol{x}$ is positive and vice versa. Thus, logistic regression can be seen as a linear classifier. The corresponding decision boundary, which separates the two given classes, is achieved by solving $\boldsymbol{\theta}^\top \boldsymbol{x} = 0$ [40]. Generally, it is assumed that classes are not fuzzy so that they can be separated into *decision regions*. Those regions are circumscribed by so-called *decision boundaries* or *decision surfaces*. In the case of linear models, those decision boundaries/surfaces can be expressed as linear functions [38].

Cost Function for Logistic Regression

The model for Logistic Regression is defined in Equation 4.2. Another task is to estimate its model parameters $\boldsymbol{\theta}$ (complies with the training process). The fact that the logistic model deals with probabilities makes it possible to use Maximum Likelihood Estimation (MLE) for fitting [40].

The likelihood of a single data point can be expressed as follows [37]:

$$P(y|\boldsymbol{x}) = \sigma\left(\boldsymbol{\theta}^{\top}\mathbf{x}\right)^{y} \cdot \left[1 - \sigma\left(\boldsymbol{\theta}^{\top}\mathbf{x}\right)\right]^{(1-y)} \tag{4.8}$$

Note that the label y follows the Bernoulli distribution. Under consideration of the whole data set, the likelihood equation is written as

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{m} \sigma\left(\boldsymbol{\theta}^{\top}\mathbf{x}^{(i)}\right)^{y^{(i)}} \cdot \left[1 - \sigma\left(\boldsymbol{\theta}^{\top}\mathbf{x}^{(i)}\right)\right]^{\left(1-y^{(i)}\right)}, \tag{4.9}$$

where $y^{(i)}$ represents the target value (label) of the i^{th} data point and $\mathbf{x}^{(i)}$ refers to the corresponding instance's feature vector. Taking the logarithm of the likelihood equation (products turn into sums), it simplifies to

$$LL(\boldsymbol{\theta}) = \sum_{i=1}^{m} y^{(i)} \log \sigma\left(\boldsymbol{\theta}^{T}\mathbf{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left[1 - \sigma\left(\boldsymbol{\theta}^{T}\mathbf{x}^{(i)}\right)\right]. \tag{4.10}$$

Equation 4.10 is also known as *log likelihood*. The model parameters $\boldsymbol{\theta}$ can be achieved by solving the MLE. However, maximising the log likelihood $LL(\boldsymbol{\theta})$ is equivalent to minimising $-LL(\boldsymbol{\theta})$. Taking the average cost over the whole data set into account, the cost function for Logistic Regression (called *log loss*) reads as follows [31]:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} LL(\boldsymbol{\theta}). \tag{4.11}$$

Now the model parameters $\boldsymbol{\theta}$ are obtained by minimising Equation 4.11. Unfortunately, no closed-form exists as it is a transcendental equation [40]. It can be solved using an optimisation algorithm like Gradient Descent. Such optimisation algorithms require the partial derivatives with respect to the j^{th} model parameters ($j = 0, \dots, n$) of the cost function (Equation 4.11) [31]:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma \left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)}. \tag{4.12}$$

From the partial derivatives the gradient vector $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is built:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\boldsymbol{\theta}) \end{pmatrix}. \tag{4.13}$$

One way to regularise Logistic Regression models is to apply the L_1 or the L_2 penalties terms to the cost function.

Gradient Descent

Gradient Descent (GD) refers to a *generic optimisation algorithm* that uses an iterative approach to find the minimum value for a function [31]. A definition of GD is given by Ruder [41, p. 1]

> "Gradient descent is a way to minimize an objective function $J(\boldsymbol{\theta})$ parameterized by a model's parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ w.r.t. to the parameters."

So, the GD takes the local gradient of a given cost function and goes in the direction of descending slope until it converges to a minimum. This is realised by randomly initialising $\boldsymbol{\theta}$. Then the parameters get tweaked iteratively to minimise the cost function until the valley is reached (Figure 4.3).

Another term related to GD is the learning rate η. The learning rate defines the step size of the optimisation algorithm and is proportional to the slope of the corresponding function (Figure 4.3). Finding an appropriate learning rate is essential for performance-enhancing: On the one hand, is η too low,

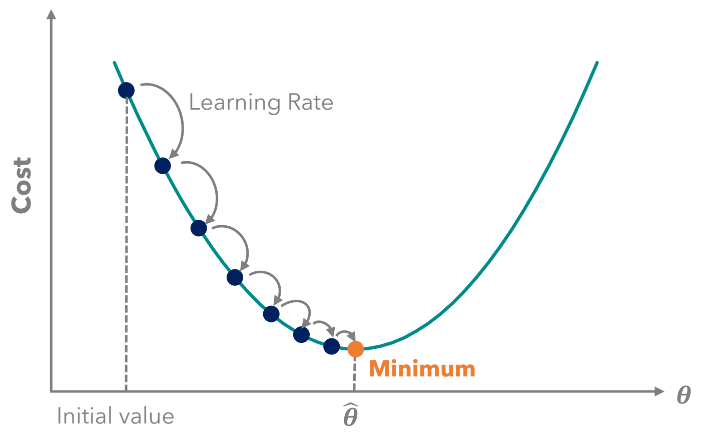


Figure 4.3.: Gradient Descent algorithm based on Géron [31].

the algorithm takes a long time to converge. On the other hand, if η is too high, the algorithm diverges.

When the cost function is convex, then it is implied that only one global minimum exists. In other words, it is guaranteed that the minimum can be found, provided that an appropriate learning rate is chosen. Once the algorithm deals with non-convex surfaces, it is not guaranteed that the global minimum is reached. It can get stuck in a local minima or in plateaus (Figure 4.4). It can be shown that the cost function of Logistic Regression is convex.

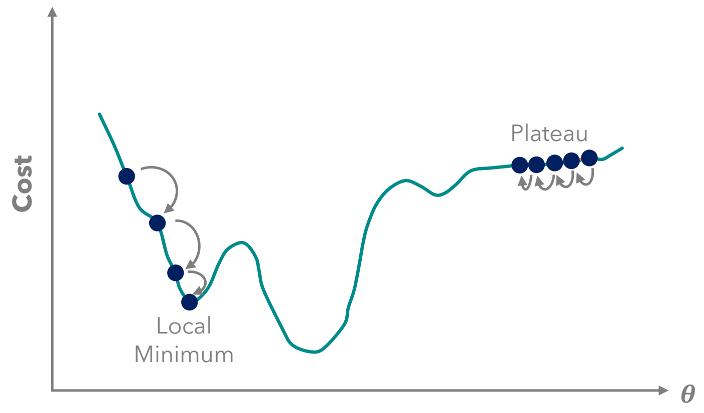


Figure 4.4.: Possible problems of Gradient Descent based on Géron [31].

GD works with the gradient vector of the objective function: $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (Equation 4.13). GD can be realised in three different ways [31], [41]:

1. **Batch Gradient Descent (BGD):**
   Computes the gradient of the cost function over whole training data set:
   $$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{4.14}$$
   The fact that the entire data set is used, makes this variant time-consuming for large training sets.

2. **Stochastic Gradient Descent (SGD):**
   Calculations are based on a single random instance at every step:
   $$\boldsymbol{\theta}^{\text{next step}} = \theta - \eta \cdot \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}, y^{(i)}\right). \tag{4.15}$$
   This variant is very fast, but due to its stochastic behaviour, it can never reach the optimum. The final parameters end up close to it[1].

3. **Mini-batch Gradient Descent:**
   Gradients are computed based on small random sets of instances, called *mini-batches*:
   $$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}, \boldsymbol{x}^{(i:i+l)}, y^{(i:i+l)}\right) \tag{4.16}$$
   where l is the size of a *mini-batch*. It represents a combination of BGD and SGD.

Softmax Regression

The Logistic Regression model is capable of solving binary classification problems. However, the model can be extended to a multiclass classifier. A multiclass classifier on basis of Logistic Regression is named *softmax regression*. The softmax regression uses the OvR scheme (Chapter 4.3). For each class k a score $s_k(\boldsymbol{x})$ [31]

[1]Assumption: training data is independent and identically distributed (IID)

$$s_k(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta}^{(k)} \tag{4.17}$$

is computed. Note that only one data point $(\boldsymbol{x}, y)$ is considered. The score only consists of the linear predictor with the difference that each class has its own fixedly assigned model parameters $\boldsymbol{\theta}^{(k)}$. Storing the model parameters for every class column by column leads to the *parameter matrix* $\boldsymbol{\Theta}$. Now the posterior probability $\hat{p}_k$ for every class is achieved by applying the *softmax function* to the scores:

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}, \tag{4.18}$$

where $\mathbf{s}(\mathbf{x})$ is a vector filled with the scores of each class concerning the instance feature vector $\mathbf{x}$. K is the total number of classes. The posterior probability $\hat{p}_k$ gives the probability that one instance $\boldsymbol{x}$ belongs to a class k based on of the given scores. The highest estimated probability defines the assigned class, as shown in Equation 4.19.

$$\hat{y} = \text{argmax } \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\text{argmax}}\, s_k(\mathbf{x}) = \underset{k}{\text{argmax}} \left(\left(\boldsymbol{\theta}^{(k)} \right)^\top \mathbf{x} \right) \tag{4.19}$$

The training process is carried out analogue to Logistic Regression. It consists of minimising the cost function

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right), \tag{4.20}$$

where $y_k^{(i)}$ refers to the target class k of the i^{th} data point: $y_k^{(i)} \in \{0, 1\}$ (the instance is member of the class, or not). The cost function in Equation 4.20 is also known as *cross entropy* and can also be seen as a performance measure of the model. It compares how well the estimations concur with the target labels. Note that the cross entropy cost function is identical with the *log loss* (Equation 4.11) if only two classes are present. As soon as the gradient vector for each class

$$\nabla_{\theta^{(k)}} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)} \right) \mathbf{x}^{(i)} \tag{4.21}$$

is computed, an optimisation algorithm can be chosen to solve the minimisation problem with regard to the parameter matrix $\mathbf{\Theta}$ [31].

Assumptions

Logistic Regression belongs to non-parametric models, so it does not require any information of the underlying distribution. However, there exist some assumptions, which should be considered when performing Logistic Regression [42]:

- **Appropriate outcome structure**
  The dependent variable (label) should be binary.
- **No or little collinearity of independent variables**
  Highly correlated features can cause problems in the parameter estimation process.
- **Linearity of independent variables and log odds**
  In other terms, the features should be linearly related to the log odds.
- **Independence of errors**
- **Lack of strongly influential outliers**
- **Adequate sample size**
  It requires a relatively large sample to avoid a biased model or overfitting. However, the correct number of training data depends on the complexity of the model and will continue to be a key subject of research.

Furthermore, all features should have similar scales (Chapter 4.1.3), else the optimisation algorithm (for example GD) will take much longer to converge [31].

4.3.2. Support Vector Machine

Support Vector Machine (SVM) is a supervised ML algorithm for binary classification problems, which combines the theory of ML, optimisation algorithms and kernel techniques [43]. It offers versatile use in the field of ML. It can be used for [31]

- linear classification
- nonlinear classification
- regression
- outlier detection

Another advantage compared to other algorithms is that it is capable of dealing with small or medium sized data sets of complex nature. SVM can be extended to solve multi-classification problems with the OvO or OvR strategy. The following subchapters discuss the principle idea behind SVM and then step deeper into the mathematical details.

Linear SVM Classification

In general, SVM tries to find an optimal hyperplane, which separates the training data. However, to demonstrate the basic idea behind SVM two randomly isotropic Gaussian blobs are generated and plotted in a two-dimensional feature space (Figure 4.5a). The axes x_1 and x_2 represent the features of the data set. The goal is to find a decision boundary, which separates the two given classes with a hyperplane (line in two-dimensional space) so that the distance between them is as far as possible. That "street", pictured by the dashed lines in Figure 4.5b, is only determined by specific data points, also called *support vectors* (indicated by the circles). In other words, SVM maximises the margin[2] between the clustered blobs and the decision boundary (black solid line in Figure 4.5b) by only using a handful of support vectors. This is also known as *Maximum Margin Classification* [38].

[2]"The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points,..." (Bishop [38, p. 327])

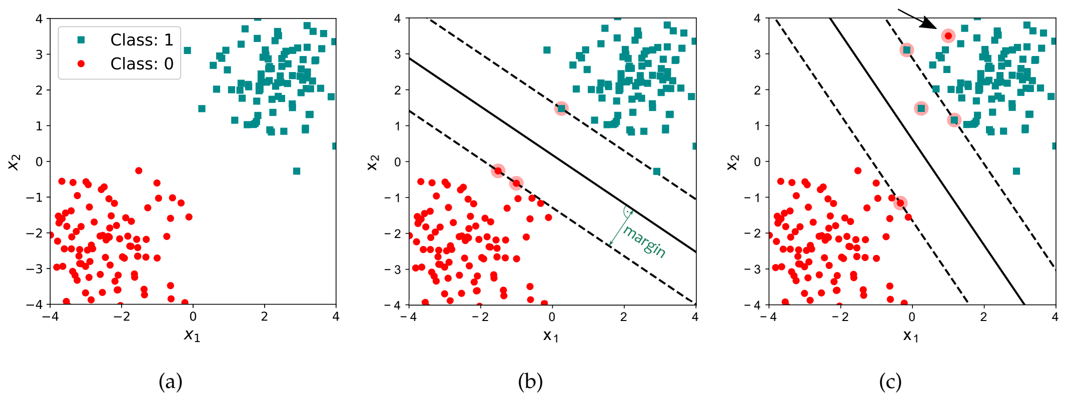


Figure 4.5.: (a) Data set in two-dimensional plane, (b) maximum-margin decision boundary, (c) Soft Margin Classification based on Géron [31].

Soft Margin Classification

As seen in Figure 4.5a, the data set is linearly separable and contains no outliers. It is possible to generate a decision boundary, where all instances are on the correct side of the "street", also known as *Hard Margin Classification*. However, it is not generally the case. Some data are not error-free. Figure 4.5c shows the same data points with the difference that one outlier (indicated with the arrow) is included. The *Hard Margin Classifier* cannot deal with outliers. Therefore, SVM introduces the *Soft Margin*. The *Soft Margin* allows instances to cross the decision boundary. This results in so-called *margin violations*: Instances that are in the middle or outside the "street". Figure 4.5c demonstrates a possible solution of a *Soft Margin Classification*. The resulting trade-off is to achieve a large margin while limiting the margin violations. This trade-off is useful to regularise the model. A larger margin may result in more misclassifications, but can also reduce overfitting [31], [44].

Kernel Functions

Figure 4.6a shows a non-separable one-dimensional data set (one feature x_1). A possible solution is to add one more feature, such as $x_2 = (x_1)^2$, to make it linearly separable [31] (Figure 4.6b).

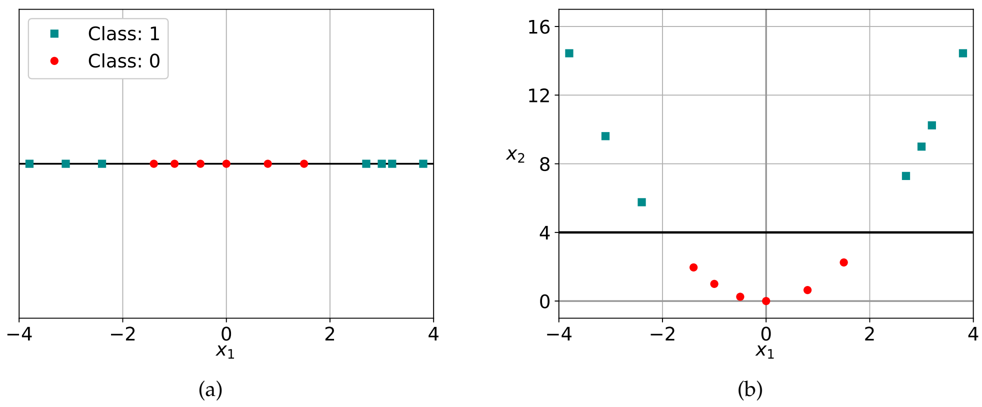


Figure 4.6.: (a) One-dimensional data set, (b) adding polynomial features to make classes separable based on Géron [31]

It represents the basic idea behind kernel functions: project training data from a low-dimensional space to a high-dimensional space so that it becomes separable. Using kernel functions also enables that the transformations can be done in the input space rather than in a high-dimensional one, which could be computationally expensive ("feature explosion"). This is accomplished by the so-called *kernel trick* [44]. The *kernel trick* will be discussed in the next subchapter, which deals with the mathematical details.

Adding polynomial features is equivalent to applying the polynomial kernel. There exist many other kernels. Some of them are listed in Table 4.1. The data set from Figure 4.5a, for example, can also be divided by using the Gaussian Radial Basis Function (RBF) kernel. The result is a curved line, which is shown in Figure 4.7a. The data is projected in a higher-dimensional space, where it can be linearly separated, and then projected back to the two-dimensional space [44]. This results in a curved line.
Furthermore, it can be shown, that a kernel function can be found, which makes it possible that any data set can be linearly separated, provided that the labels are consistent [44]. However, projecting the input space in a high-dimensional one can cause difficulties since the possible solutions increase with the number of dimensions. Using a very high dimensional kernel function results in the solution seen in Figure 4.7b. The decision boundary borders the instances. This solution strongly overfits.

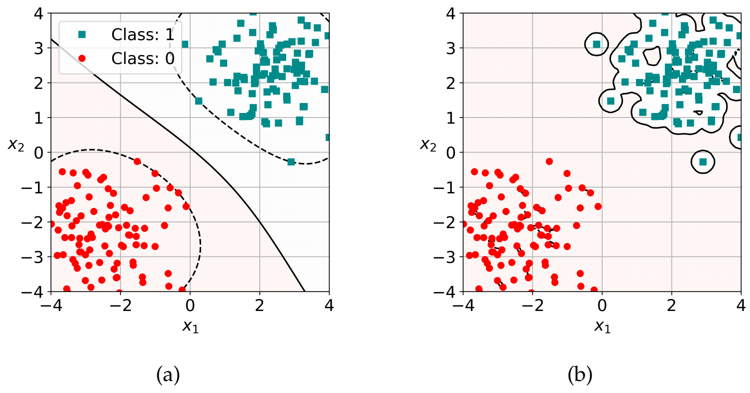


Figure 4.7.: (a) Decision Boundary using a Gaussian Radial Basis Function, (b) Overfitting of SVM based on Noble [44].

Decision Function, Prediction and Training

The model for a SVM classifier is given in Equation 4.22 [31], [38]. The predicted class $\hat{y}$ is either 1, if the linear combination of the transformed feature vector $\phi(\boldsymbol{x})$ is positive, else the new instance belongs to class 0. The mapping function ϕ maps data from the input space in the higher-dimensional feature space. For linear SVM, for example, the following applies: $\phi(\boldsymbol{x}) = \boldsymbol{x}$.

$$\hat{y} = \begin{cases} 0 \text{ if } \boldsymbol{w}^\top \phi(\boldsymbol{x}) + b < 0 \\ 1 \text{ if } \boldsymbol{w}^\top \phi(\boldsymbol{x}) + b \geq 0 \end{cases}, \tag{4.22}$$

where b is the bias term, and $\boldsymbol{w}$ is the feature weights vector. The term $\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b$ is called *decision function*. The decision boundary (indicated with the black solid line in Figures 4.5b and 4.5c) is determined by

$$\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b = 0. \tag{4.23}$$

The width of the "street" (dashed lines in Figure 4.5 and 4.7a) are the result of

$$\begin{aligned} \boldsymbol{w}^\top \phi(\boldsymbol{x}) + b &= -1 \\ \boldsymbol{w}^\top \phi(\boldsymbol{x}) + b &= +1. \end{aligned} \tag{4.24}$$

The main objective of training the model is to find values for the feature weights $\boldsymbol{w}$ and the bias term b so that the margin is as large as possible while limiting the margin violations.

For the *Hard Margin Classification* it is demanded that the decision function is greater than 1 for all positive instances and lower than -1 for all negative instances. Introducing the target values $t^{(i)} \in \{-1, 1\}$ and assuming that the training data are linearly separable, there exists at least one solution for $\boldsymbol{w}$ and b that satisfies $t^{(i)}(\boldsymbol{w}^\top \phi(\boldsymbol{x}^{(i)}) + b) \geq 1$ for $i = 1, \ldots, m$. Another constraint is to maximise the margin. The principle applies that the smaller the weights $\mathbf{w}$, the larger the margin. The task is to maximise $\|\mathbf{w}\|^{-1}$ or to minimise $\|\mathbf{w}\|^2$. The constrained optimisation problem of a *Hard Margin Classification* can be summarised as follows [31]:

$$\begin{aligned} &\underset{\mathbf{w},b}{\text{minimise}} \ \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to } t^{(i)}\left(\mathbf{w}^\top \phi(\boldsymbol{x}^{(i)}) + b\right) \geq 1 \text{ for } i = 1, 2, \cdots, m. \end{aligned}$$

The factor $\frac{1}{2}$ simplifies the optimisation task due to the fact that $\|\mathbf{w}\|$ is not differentiable at $\mathbf{w} = 0$, while $\frac{1}{2}\|\mathbf{w}\|^2$ is a differentiable function [31].

This optimisation problem is an example of a convex quadratic optimisation problem with linear constraints. The solution can be obtained by usage of *Lagrange multipliers* $\alpha^{(i)} \geq 0$ for each constraint [38]. This leads to the dual representation of the SVM optimisation problem [31], [38]:

$$\begin{aligned} &\underset{\alpha}{\text{minimise}} \ \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) - \sum_{i=1}^{m}\alpha^{(i)} \\ &\text{subject to } \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m \end{aligned} \tag{4.25}$$

where $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)})^\top \phi(\boldsymbol{x}^{(j)})$ represents the kernel function. Estimations $\hat{\boldsymbol{w}}$ and $\hat{b}$ can be achieved solving the quadratic programming problem.

The mapping function ϕ transforms data from the input space in a higher-dimensional feature space. This can be computationally expensive. SVM circumvents the transformation using the *kernel trick*. As seen in Equation 4.25, the kernel function consists of the dot product of the transformed feature input vectors. The kernel trick allows the computation of dot product $\phi(\boldsymbol{a})^\top \phi(\boldsymbol{b})$ by only using the input vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. Some common kernels are given in Table 4.1 for two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. The parameters γ, d and r can be used for regularisation. A high γ, for example, results in a more irregular decision boundary, which encircles the instances (Figure 4.7b). With the parameter d, the degree of the polynomial is defined and r describes a constant term.

Table 4.1.: Common kernels based on Géron [31].

Linear	$k(\boldsymbol{a},\boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$
Polynomial	$k(\boldsymbol{a},\boldsymbol{b}) = (\gamma \boldsymbol{a}^\top \boldsymbol{b} + r)^d$
Gaussian RBF	$k(\boldsymbol{a},\boldsymbol{b}) = \exp(-\gamma \vert\vert \boldsymbol{a} - \boldsymbol{b} \vert\vert^2)$

For the *Soft Margin Classification* the *slack variable* $\zeta^{(i)} \geq 0$ is introduced for each data point. The slack variable quantifies how much the i^{th} instance is allowed to cross the "street". Additional to the other constraints, it is now required that the slack variables are as small as possible, leading to [31]

$$\begin{aligned}
&\underset{\mathbf{w},b,\zeta}{\text{minimise}}\ \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{i=1}^{m} \zeta^{(i)} \\
&\text{subject to } t^{(i)}\left(\mathbf{w}^\top \phi(\boldsymbol{x}^{(i)}) + b\right) \geq 1 - \zeta^{(i)} \text{ and } \zeta^{(i)} \geq 0 \text{ for } i = 1, 2, \cdots, m
\end{aligned}$$

where C is a user-specified hyperparameter, which controls the trade-off between maximum margin and margin violations: The larger C the smaller the margin.

Assumptions

SVM does not require a normally distributed training set [44]. Moreover, the features must show similar scales since the feature scale influences the width of the decision boundary.

4.3.3. Decision Tree

Decision Tree (DT) can be used for classification, regression and multioutput tasks [31]. A multioutput task defines a classification task, where a label can belong to multiple classes. It is also a basic module for Random Forest classifiers, which will be a topic in this subchapter.

A DT performs the classification task using a tree-like structure (Figure 4.8a), containing [45]

- internal node: represents a feature,
- branch: represents a decision rule,
- leaf node: represents the classes,
- root node: represents topmost node with the most information gain,

to make decisions. A node, which can be divided into sub-nodes, is also known as a parent node. Its sub-nodes are called child nodes [46].

An example of a simple DT is illustrated in Figure 4.8b. Starting at the root node: if a feature value x_1 is smaller than a specific threshold a, the process continues at the root's left child node. If the second feature value x_2 is greater than b, the instance is classified as C_2. That is how a DT make predictions. The corresponding decision boundaries are visualised in Figure 4.8c. A DT is a non-linear model, which builds many linear decision boundaries [47].

There exist many different implementations for DTs. One way is to use the *CART* algorithm. It is only capable of creating *binary trees* (True or False), but represents the basic component of the Random Forest classifier. Other algorithms, like ID3, construct multi-way trees (nodes have more than two children). However, CART algorithms represent one way to train DTs [31].

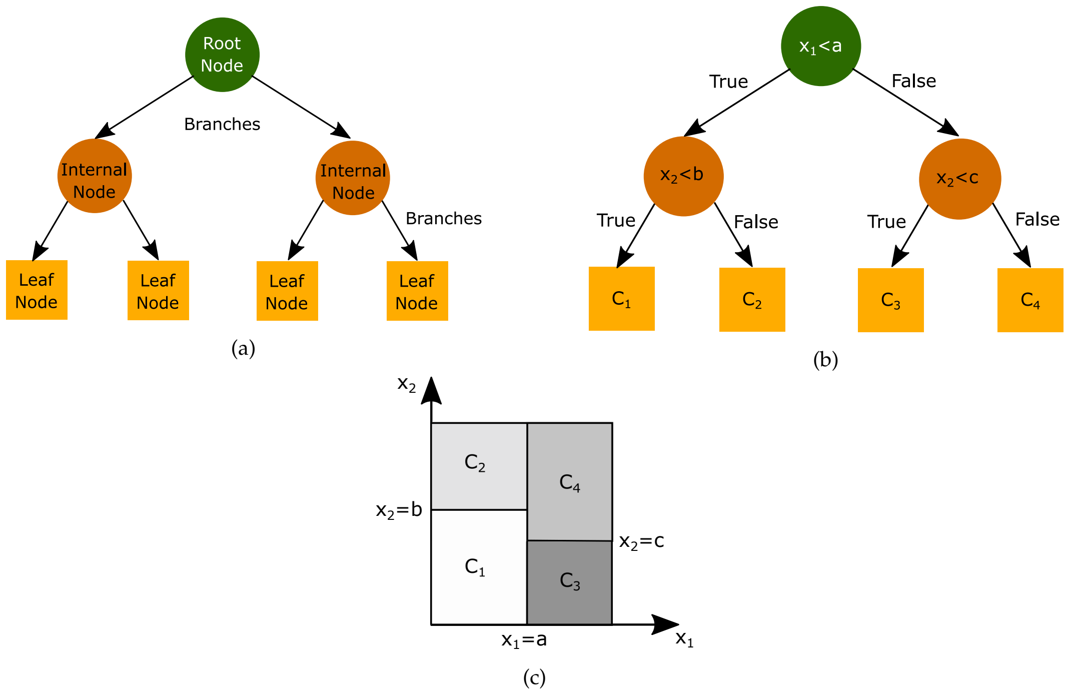


Figure 4.8.: (a) Components of DT, (b) example of DT, (c) DT decision boundaries based on Sá *et al.* [45].

The CART Training Algorithm

The Classification and Regression Tree (CART) algorithm creates binary trees based on the selection of features and thresholds, which leads to the largest information gain at the nodes or to the greatest reduction in *Gini Impurity* [47]. The Gini Impurity measures the *impurity* of each node: A node is declared as *pure*, when its Gini score G_i is equal to 0. This means that all training instances in the corresponding node, belong to the same class. The Gini Impurity is defined as follows [31]:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \quad , \tag{4.26}$$

where $p_{i,k}$ represents the ratio of the instances, which belongs to the class k, among the training instances in the i^{th} node. An example of the computation of Gini Impurity is given in Table 4.2, where the example refers to Figure 4.8b. Supposing that there are 80 samples, which are equally split into four classes $C_k \in \{1,2,3,4\}$. The Gini score in the root note is equal to

$$1 - \left(\frac{20}{80}\right)^2 - \left(\frac{20}{80}\right)^2 - \left(\frac{20}{80}\right)^2 - \left(\frac{20}{80}\right)^2 = 0.75 \text{ ... (maximum impurity).}$$

The most left leaf node, which corresponds to C_1, contains 20 samples, where 19 samples belong to C_1 and one sample is misclassified. This leads to a Gini score of

$$1 - \left(\frac{19}{20}\right)^2 - \left(\frac{1}{20}\right)^2 - \left(\frac{0}{20}\right)^2 - \left(\frac{0}{20}\right)^2 = 0.095.$$

Table 4.2.: Example of computation of Gini Impurity based on Figure 4.8b.

Class	1	2	3	4	**Gini Impurity [-]**
Counts in root node: $x_1 < a$	20	20	20	20	0.750
Counts in left child node: $x_2 < b$	20	20	0	0	0.500
Counts in most left leaf node: C_1	19	1	0	0	0.095

Now the CART algorithm splits the training set into two subsets based on a single feature x and a threshold t_x. The main objective is to find a feature x plus its corresponding threshold t_x, which creates the purest subsets [31]. Once a pair (x, t_x) is found, the process is repeated for the next sub-subsets until the impurity cannot be reduced or the maximum depth is reached. DTs belongs to the category of *nonparametric* models, which mean that the number of parameters is not declared a priori in contrast to linear models, which shows limitations in degree of freedom. So the model is likely to overfit when no constraints are set. One way to regularise the model is to set the maximum depth of the tree, or to define the minimum number of samples that a leaf node must contain. A DT is also capable of giving the probability that an instance belongs to a specific class.

Random Forest

Random Forest is an ensemble of *Decision Trees*. Instead of looking for the best feature while a node is split, Random Forest classifier selects the best feature from a random quantity of features. As a result, it adds additional randomness to the model while growing. So it ends up in a wide variety, which generally leads to a better generalisation.

Assumptions and limitations

The features do not need to be scaled or centred. Furthermore, DT does not make any distributional assumptions. A disadvantage is that DTs are sensitive to rotations in the training set since the decision boundaries are orthogonal and linear. Another issue is that DTs are highly sensitive to small changes in the data set. Small permutations can result in a completely different tree.

4.3.4. Neural Network

Artificial Neural Network (ANN) is a highly versatile and adaptable ML algorithm, which is capable of dealing with large, as well as with complex data sets. It can solve regression, classification and multi-output problems. ANNs are inspired by the operating principle of the biological neurons in the human brain. The following subsections are based on Géron [31].

Perceptron

The perceptron represents a simple realisation of an ANN and is based on a "neuron" called *Threshold Logic Unit (TLU)* (Figure 4.9a). A TLU takes the inputs (instance feature vector $\boldsymbol{x}$) and computes the weighted sum of them: $z = \boldsymbol{x}^\top \boldsymbol{w}$. To get the output, a step function is applied to the linear combination: $\mathrm{h}_{\boldsymbol{w}} = \mathrm{step}(z)$, where $\boldsymbol{w}$ represents the weight vector. A frequently used step function is the *Heavside step function*:

$$\mathrm{heaviside}\ (z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} . \tag{4.27}$$

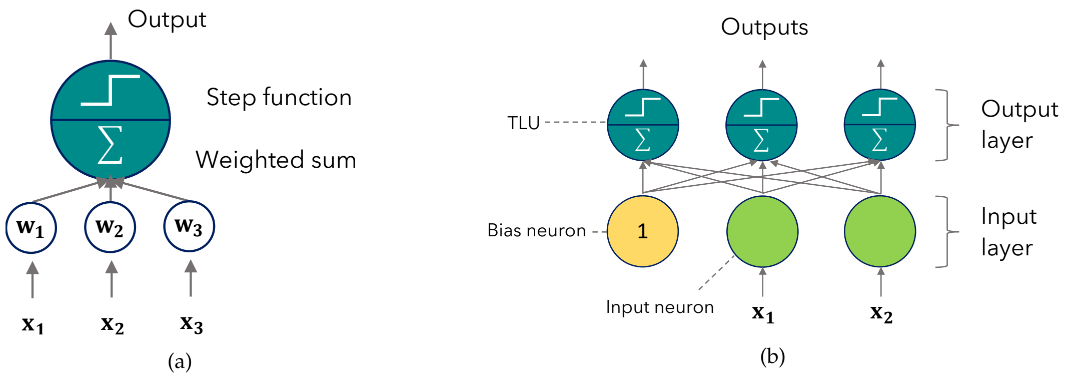


Figure 4.9.: (a) Threshold logic unit (TLU), (b) Topology of the perceptron with two input neurons and three TLUs based on Géron [31].

A perceptron consists of a single layer of TLUs, where each TLU is fully connected to its inputs (*input layer*). An additional bias feature ($x_0 = 1$) is added in the input layer. This bias feature is also known as *bias neuron* and outputs 1. The architecture of a perceptron is exemplified in Figure 4.9b. The output of a fully connected layer is computed via the following formula:

$$\mathrm{h}_{\boldsymbol{W},\boldsymbol{B}}(\boldsymbol{X}) = \phi(\boldsymbol{X}\boldsymbol{W} + \boldsymbol{B}), \tag{4.28}$$

where $\boldsymbol{X}$ is the matrix of input features and $\boldsymbol{W}$ represents the matrix of the connection weights (exclusive the bias neuron). The bias matrix $\boldsymbol{B}$ contains the bias vector $\boldsymbol{b}$ copied into each row. The bias vector $\boldsymbol{b}$ is filled with the connection weights between the bias neurons and the artificial neurons. The *activation function* is defined by ϕ. In case of TLUs, the activation function is the step function. However, there exist more activation functions, which will be discussed later. The training objective is to find the correct connection weights. The basic principle is defined by the *Hebb's rule*, which says: "neurons wire together, if they fire together" (Löwel and Singer [48]). Perceptrons use a slightly different approach: The perceptron learning rule considers the prediction error made by the network. When an output neuron makes a wrong prediction, then it reinforces those input neurons that would lead to the correct answer. The connection weight $w_{i,j}$ of the present training instance between the i^{th} input neuron and the j^{th} output neuron is computed as shown in Equation 4.29:

$$w_{i,j}^{\text{(next step)}} = w_{i,j} + \eta \left(y_j - \hat{y}_j \right) x_i, \tag{4.29}$$

where x_i is the i^{th} input value, $\hat{y}_j$ is the j^{th} output neuron. The target output of the j^{th} output neuron is declared as y_j and η represents the learning rate.

Such perceptrons can, however, cannot solve problems that are too complex. Furthermore, they make predictions based on a hard threshold. However, stacking multiple perceptrons can overcome these problems.

The Multilayer Perceptron and Backpropagation

The architecture of a *Multilayer Perceptron (MLP)* is demonstrated in Figure 4.10. The MLP introduces *hidden layers* (one or more layer of TLUs). The bias neuron is present in every layer except the output layer and is fully connected to its subsequent layer.

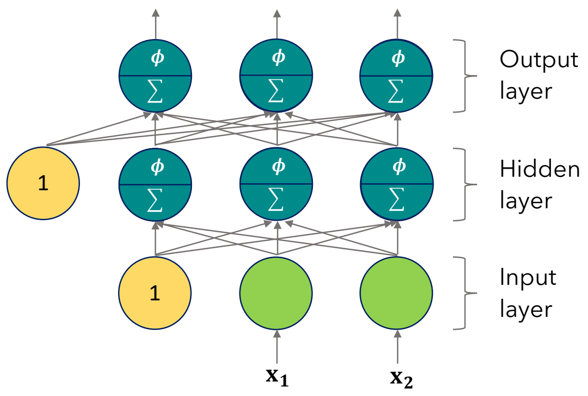


Figure 4.10.: Topology of a Multilayer Perceptron with two input neurons, one hidden layer of three neurons and three output neurons based on Géron [31]. ϕ indicates an arbitrary activation function.

The scientific breakthrough brings RUMELHART *et al.* [49], who proposed the *back propagation* training algorithm. It presents an alternative approach for implementing Gradient Descent (GD), introduced in Chapter 4.3.1, which computes the gradients automatically: This is achieved in two passes through the system: one forward and one backwards. Each training instance is passed through all hidden layers until the output layer is reached (forward pass). Now the predictions are made. Those predictions are stored since they play an important role for the backward pass. The next step is to determine the network's output error. Therefore, for example, a loss function is used, which computes the error between the actual and the desired output. The cross entropy usually serves as a loss function (Equation 4.20). The next step is to go backwards (through all layers) to quantify the error contribution and the error gradient from the weights, respectively (backward pass). Mathematically, the *chain rule* is utilised for this step. Ultimately, the connection weights and bias terms are tweaked so that the error is reduced. Therefore, a Gradient Descent Step is applied by using the error gradients from the previous step. In contrast to perceptrons, the activation function is also different. Since the step function is a piecewise constant function, GD cannot move along. However, there exist several alternative activation functions. In practice the Rectified Linear Unit (ReLU) function

$$\mathrm{ReLU}(z) = \max(0, z) \tag{4.30}$$

is frequently used, since it is fast to compute. Unfortunately, the ReLU activation function suffers from the problem called *dying* ReLUs. During the training process, it is possible that some neurons only output 0. The output is 0 if the weighted sum of the inputs is negative (Equation 4.30). An advanced alternative activation function is the Scaled Exponential Linear Unit (SELU) activation function (Equation 4.31), which represents a scaled variant of the *exponential linear unit* [50].

$$\mathrm{SELU}(z) = \lambda \begin{cases} z & \text{if } z > 0 \\ \alpha e^{z} - \alpha & \text{if } z \leq 0 \end{cases}, \tag{4.31}$$

The values $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$ are fixed. The SELU function does not only circumvent the dying ReLU problem, the network also *self-normalises*.

The output of each layer tries to maintain zero-mean and unit-standard deviation.

Classification Multilayer Perceptrons

For the classification task, the activation function of the output layer (Figure 4.10) must be adapted. For binary classification tasks, a single output neuron is needed as well as a output between 0 and 1. This is achieved by using the *logistic (sigmoid) function* (Equation 4.4) as output activation. The output can be interpreted as the probability of the positive class. For multiclass classification problems, one output neuron per class is needed. Thus, the *softmax activation function*, introduced in Chapter 4.3.1, is used as the output layer activation.

However, training an ANN model is very complex, since there exist many hyperparameters and strategies to optimise the model. Some hyperparameter are worth to mention:

- Number of hidden layers
- Number of neurons per hidden layer
- Learning rate
- Optimiser: a traditional optimiser represents the Mini-batch Gradient Descent (Chapter 4.3.1). However, there exist advanced optimisers such as *Momentum Optimisation*.
- Batch Size
- Activation function
- Number of iterations

Just as the Logistic Regression model, the ANN can be regularised using the L_1 and L_2 norm. However, there also exist much more regularisation techniques, which lie beyond the scope of this thesis. More information can be found in Géron [31].

Assumptions

MLP does not assume that the data is normally distributed. Another condition is that the connection weights of all hidden layers are initialised randomly so that the backpropagation works properly. Last but not least, MLP networks are sensitive to feature scaling.

4.4. Evaluation Metrics

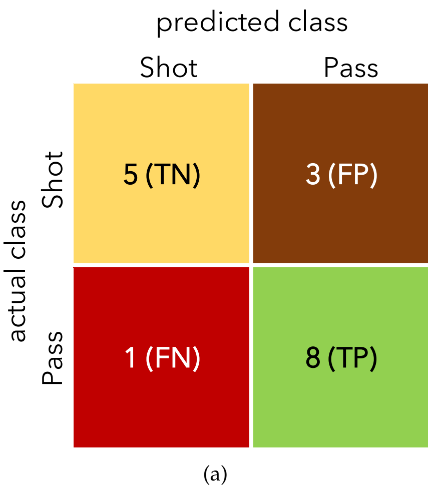
The performance of a model can be evaluated using the *confusion matrix*. The confusion matrix counts the number of correct/incorrect predictions made by the trained model and displays the result in a $K \times K$ matrix, where K is the number of classes [51]. The columns indicate the predictions, while the rows represent the actual labels. The diagonal elements contain the correctly classified instances. Figure 4.11a exemplifies a confusion matrix (2×2) for binary classification for a soccer game: Assuming that the class "Pass" corresponds to the positive class and "Shot" to the negative class, there exist four possible classification results:

- **true negative (TN)**: The matrix entry indicates the number of instances classified correctly as the negative class. 5 instances are predicted correctly as class non-pass ($\widehat{=}$ shot).

- **true positive (TP)**: The matrix entry indicates the number of instances classified correctly as the positive class. In this case, it means that 8 passes are correctly assigned to class pass.

- **false negative (FN)**: The matrix entry indicates the number of instances, which were classified wrongly as the negative class. So one pass has been misclassified as shot.

- **false positive (FP)**: The matrix entry indicates the number of instances, which were classified wrongly as the positive class. Figure 4.11a says that 3 instances are classified as pass, although the instances belong to shot.



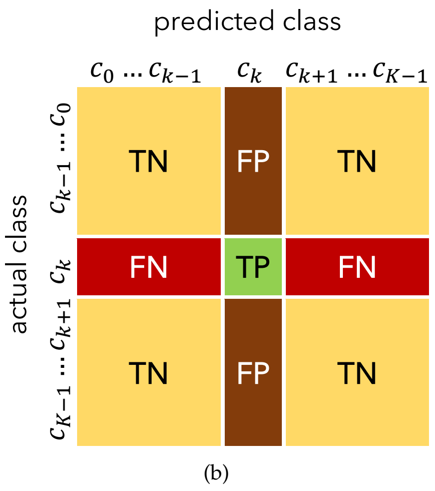


Figure 4.11.: Confusion matrix for a multi-class and binary-class problem: (a) Example of a confusion matrix for a binary-class problem where "Pass" is assigned as the positive class, (b) Confusion matrix for a multi-class problem based on Krüger [52].

The confusion matrix for a multi-classification task with respect to the target class k is shown in Figure 4.11b. Several metrics, also called *scores*, are derived from the confusion matrix $C := (c_{kj})$. The most important evaluation metrics are listed below [52], [53]:

- **Accuracy:** The accuracy measures the overall-performance of the correctly predicted labels.

$$\text{accuracy} := \frac{\sum_{k=0}^{K-1} c_{k,k}}{\sum_{k=0}^{K-1} \sum_{j=0}^{K-1} c_{k,j}} \tag{4.32}$$

  Accuracy as a performance measure can be misleading, especially when dealing with imbalanced data sets since classes, which are more present, are preferred [31].

- **Precision:** represents the accuracy of positive predictions. The precision for a target class k is computed as follows:

$$\text{precision}_k := \frac{TP_\text{k}}{TP_k + FP_k}. \tag{4.33}$$

- **Recall:** also called *sensitivity* or *true positive rate (TPR)*. It quantifies the ability of identifying the correct class:

$$\text{recall}_k := \frac{TP_\text{k}}{TP_k + FN_k}. \tag{4.34}$$

  Increasing the precision results in a lower recall and vice versa. Depending on the application, the right trade-off between precision and recall must be found.

- **F_1-Score:** is the harmonic mean of precision and recall:

$$F_{1_k} := \frac{2 \cdot TP_k}{2TP_k + FN_k + FP_k}. \tag{4.35}$$

- **Specificity:** also called *true negative rate (TNR)*. It defines the number of true negatives out of the total amount of true negatives:

$$TNR_k := \frac{TN_k}{FP_k + TN_k}. \tag{4.36}$$

For the multi-classification problem, it is useful to have an overall performance measure. Therefore the single class scores can be combined by averaging them. Two common approaches are: micro and macro averages. The macro average computes the desired metric for each class and then calculates the unweighted mean. The macro-average of the recall, for example, is shown in Equation 4.37 [53]. This metric is favoured, when the data set is imbalanced [54] and will be used as an overall-performance measure in this thesis.

$$\text{recall}_\text{macro} := \frac{\sum_{k=0}^{K-1} \text{recall}_k}{K} \tag{4.37}$$

When it is desired to prefer the class with the most instances, then the micro-average is computed [54]. It takes the individual classification results into account.

A good visualisation of the performance of a binary classifier represents the Receiver Operating Characteristic (ROC) curve [31]. The ROC curve plots the recall versus the false positive rate (FPR), which equals to $1 - TPR$. The corresponding performance measure is the ROC AUC score. It computes the Area Under the Curve (AUC) ROC curve and gives information about how separable classes are. The AUC score ranges between 0 and 1. The following applies: The greater the ROC AUC score the better the model. A perfect model has a ROC AUC equal to 1.0, whereas a classifier, which makes random guesses, outputs a ROC AUC of 0.5.

Those evaluation metrics can be computed for the training as well as for the test set. For the test set, predictions are directly made and compared with the actual labels. The models' performance regarding the training set is obtained via the k-fold cross-validation [55]. The k-fold cross-validation splits the training set into k random subsets called folds. The model is trained using $k - 1$ folds, the remaining fold is used to make predictions. Those predictions are then compared with the actual labels. This process is repeated until every fold is once used for making predictions. The result is usually summarised with the mean of the evaluation scores. The strategy of k-fold cross-validation is also used for hyperparameter tuning [31] (Chapter 4.1.6).

4.5. Dimensionality Reduction

Having many features makes it harder to find a good solution due to the *curse of dimensionality*: The classification accuracy suffers significantly in high dimensions since additional features in the training set increase the possible solutions exponentially [44]. Reducing the number of dimensions does not only help to mitigate the *curse of dimensionality*, but it also speeds up training and is a useful visualisation tool. Another object is that the act of dimensionality reduction can be seen as a feature extraction method (Chapter 4.1.2) in means of projecting the original feature space onto a lower-dimensional one without much information loss [51].

Two common dimensionality reduction algorithms correspond to

- **Principal Component Analysis (PCA)** and
- **Linear Discriminant Analysis (LDA)**

Both represent a linear transformation technique [56]. Hence, the main objective is to find a linear transformation matrix $\boldsymbol{W}_{d \times k}$, which maps the original d-dimensional feature vector $\boldsymbol{x}$ onto a k-dimensional feature subspace, where $k \ll d$. The vector $\boldsymbol{x}$ is represented by d features. Considering m independent realisations of the vector $\boldsymbol{x} \in \mathbb{R}^d$, the new reduced feature vector $\boldsymbol{z} \in \mathbb{R}^k$ can be computed via

$$\mathbf{z} = \mathbf{W}^T \mathbf{x}. \tag{4.38}$$

4.5.1. Principal Component Analysis

PCA is an unsupervised dimensionality reduction algorithm with the aim of finding a hyperplane that preserves the maximum variance of the original data. In PCA, correlated features are transformed in linearly uncorrelated features, called *principal components (PC)* [51] (Figure 4.12).

The linear transformation matrix $\boldsymbol{W}$ is obtained by eigenvalue decomposition of the covariance matrix $\boldsymbol{S}$ [56]:

$$\mathbf{W} \leftarrow \text{eig\_decomposition} \left(\mathbf{S} = \sum_{i=1}^{m} \left(\mathbf{x}^{(i)} - \bar{\mathbf{x}} \right) \left(\mathbf{x}^{(i)} - \bar{\mathbf{x}} \right)^T \right), \tag{4.39}$$

where $\boldsymbol{x}^{(i)}$denotes the i^{th} instance feature vector and m is the number of instances. The vector $\bar{\boldsymbol{x}}$ represents the mean vector

$$\bar{\mathrm{x}} = \frac{1}{m} \sum_{i=1}^{m} \mathrm{x}^{(i)}. \tag{4.40}$$

In other words, the eigenvalues ($\boldsymbol{S} : \lambda_1 > \lambda_2 > \cdots > \lambda_d$) and eigenvectors ($\boldsymbol{S} : \boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d$) are computed by decomposing the covariance matrix $\boldsymbol{S}$. The next step is to select k eigenvectors according to the k largest eigenvalues. The linear transformation matrix now consists of the selected eigenvectors.

The number k of PCs can be set as a predefined number or in terms of percentage of variance explained. For example, a threshold can be set to preserve 95% of the variance of the original data set [57]:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} > 0.95. \tag{4.41}$$

PCs are sensitive to the unit and range of the initial input [57]. Therefore it is needed to perform *standardisation* (Chapter 4.1.3). One drawback of PCA is that it gives high weights to features with high variability, irrespective of any discriminating character [56].

4.5.2. Linear Discriminant Analysis

LDA is a classification algorithm, which tries to find a projection that keeps the classes as far away as possible. However, LDA can also be seen as a supervised dimensionality reduction algorithm. In LDA, a linear combination of the original features is built so that the mean difference between the classes is maximised. The new features are called *linear discriminants (LD)* (Figure 4.12).

The transformation matrix $\boldsymbol{W}$ is achieved by eigenvalue decomposition of the so-called *scatter-matrices* [56], [58]: One matrix is called *within-class* scatter matrix $\boldsymbol{S}_W$:

$$\mathbf{S}_W = \sum_{j=1}^{K} n_j \sum_{i=1}^{n_j} \left(\mathbf{x}_j^{(i)} - \bar{\mathbf{x}}_j\right) \left(\mathbf{x}_j^{(i)} - \bar{\mathbf{x}}_j\right)^T, \tag{4.42}$$

where K and n_j is the number of classes and the number of samples in class j, respectively. $\boldsymbol{x}_j^{(i)}$ is the i^{th} instance's feature vector of class j. The mean vector of the class j is denoted as $\bar{\mathbf{x}}_j$. The second scatter matrix is known as *between-class* scatter matrix $\boldsymbol{S}_B$:

$$\mathbf{S}_B = \sum_{j=1}^{K} n_j \left(\bar{\mathbf{x}}_j - \bar{\mathbf{x}}\right) \left(\bar{\mathbf{x}}_j - \bar{\mathbf{x}}\right)^T, \tag{4.43}$$

where $\bar{\mathbf{x}}$ is the mean vector of the whole input data. The corresponding algorithm is summarised in Equation 4.44 [56]. The dimension of the new feature subspace k is restricted by $k \leq K-1$, since the rank of S_B is at most $K-1$.

$$\begin{aligned} \mathbf{W}_{S_W} &\leftarrow \text{eig\_decomposition}\left(S_W\right), \quad \mathbf{x}_{S_W} = \mathbf{W}_{S_W}\mathbf{x} \\ \mathbf{W}_{S_B} &\leftarrow \text{eig\_decomposition}\ \left(S_B \mid \mathbf{x}_{S_W}\right) \\ \mathbf{W} &= \mathbf{W}_{S_W}\mathbf{W}_{S_B} \end{aligned} \tag{4.44}$$

The assumption of a normal distribution is applied to LDA as a classifier. However, for the purpose of reducing dimensions, the algorithm gives reasonable solutions even though this assumption is violated [59].

A comparison of PCA and LDA is schematically shown in Figure 4.12.

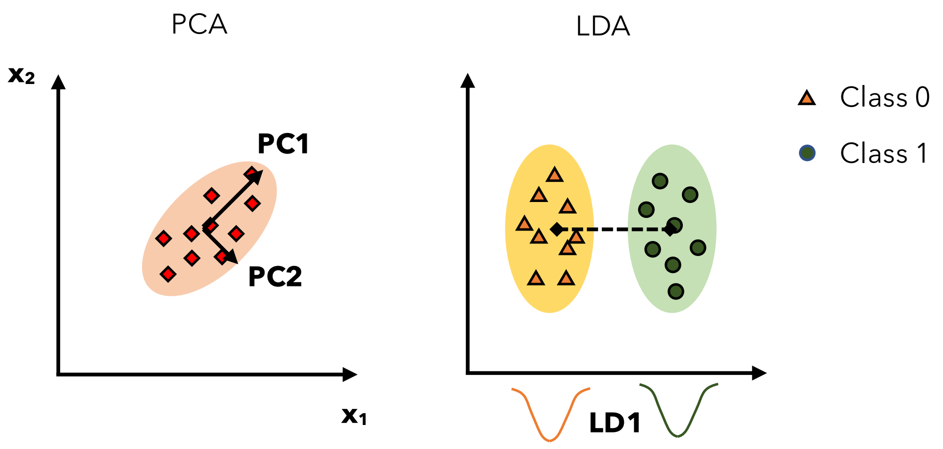


Figure 4.12.: PCA vs. LDA based on Raschka [59]. PCA: preserving maximum amount of variance. LDA: maximises distance between classes.

Part II
Development

5. Introduction

This chapter gives a short overview of the defined activities and the embedded sensors. In general, two MEMS IMUs, each attached to one shin guard of a soccer player, are used to perform Activity Recognition (AR). The accelerometer data is utilised for this task. All calculations were performed in *Python* 3.7. The used Python modules for Machine Learning are *scikit-learn* and *TensorFlow*2.

5.1. Activities

In the course of this thesis, five various soccer-specific activities are studied. They are listed in Table 5.1 and subsequently shortly described.

Table 5.1.: List of selected activities and their corresponding abbreviations.

Activity	Abbreviation
Standing	ST
Walking	WA
Running	RU
Pass	PA
Shot	SH

- **Standing (ST)**: Standing is defined as being in an upright position with both feet on the ground. No or little movement with feet noticeable.

- **Walking (WA)**: Moving on foot at a moderate speed ($\approx$ 5.5 km/h).

- **Running (RU)**: Moving on foot at an advanced speed. The activity "Running" includes movements on foot with a speed higher than 8 km/h.

- **Pass (PA)**: The soccer player tries to kick the ball to a region where a teammate is located. Passes can be short or long. There exist different types of soccer passes, such as inside foot and outside foot. The pass can be done with the right or the left leg.

- **Shot (SH)**: Defines a more intense kick of the ball. Basic techniques regarding shooting a ball are inside foot, outside foot and full instep kick. The shot can be done with the right or the left leg.

According to LEVANON and DAPENA [60] the kicking motion consists of three phases and four key moments (Figure 5.1). The leg is pulled back to gain momentum. The first phase, "Back-swing", starts with the toe-off of the shooting leg until maximum hip extension. The knee bends until the maximal flexion is achieved ("Leg-cocking"). The last phase ("Leg-acceleration") describes the process between maximal knee flexion and ball impact.

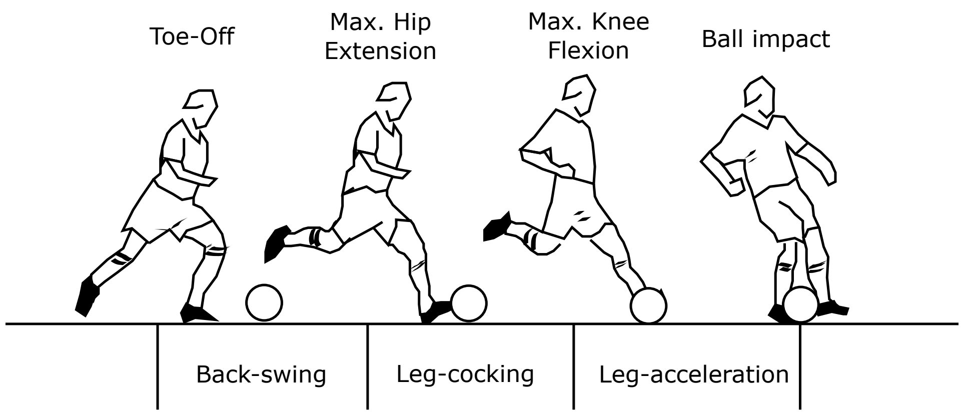


Figure 5.1.: Kicking motion according to LEVANON and DAPENA [60]

5.2. Inertial Measurement Unit

The activities are recorded via a MPU-9250 MotionTracking device manufactured by *InvenSense Inc.* and is available in $3 \times 3 \times 1$ mm Quad Flat No-lead package (QFN). It is a multi-chip module that comprises a triaxial gyroscope, a triaxial accelerometer and a triaxial magnetometer. A 16-bit Analog-to-Digital Converter (ADC) converts the output for each axis and sensor. The gyroscope can output angular rates in a user-programmable range of minimal $\pm 250°/\text{sec}$ and maximal $\pm 2000°/\text{sec}$. The accelerometer can supply data in a range of ± 2 g, ± 4 g, ± 8 g or ± 16 g, respectively. The displacement of the proof mass is detected by capacitive sensors (Chapter 2.1.1). The magnetometer shows a full range of ± 4800 μT. The device also features an integrated temperature sensor. A detailed explanation of the sensor specifications can be found in the datasheet of InvenSense Inc. [61].

In this thesis, two MPU-9250 sensors, one sensor assigned to each leg, are utilised. Following, to distinguish the sensors in the evaluation process, they are denoted as “Sensor 1” and “Sensor 2”. They are not dependent on any leg, in terms of left and right.

6. Data Collection and Preparation

The accelerometer of MPU-9250 was used to acquire data to study the five selected activities. The following sections give a detailed overview of the test setup, including the realisation of data-acquisition, the sensor's orientation and calibration as well as the data collection and the following preprocessing steps.

6.1. Test Setup

To acquire data from the measurement device it is needed to interface the IMU to a Microcontroller Unit (MCU) via an Inter-Integrated Circuit (I2C) communication protocol. The chosen MCU is the ESP32, designed by *Espressif*. The ESP32 features integrated Wi-Fi and Bluetooth connectivity as well as a robust design and ultra-low power consumption [62]. Finally, a Printed Circuit Board (PCB), developed by Christoph Schmied, Dipl.-Ing. (Institute of Geodesy, Working Group Navigation, TU Graz), represents the necessary hardware components like an SD card reader, where the recorded data is stored on a memory card for further post-processing. The used library to communicate with the MPU-9250 is a modified version from Bolder Flight Systems [15]. The PCB is mounted to a plastic plate, which can be attached to the leg using straps. The whole test setup is shown in Figure 6.1. The power is supplied by a rechargeable battery.

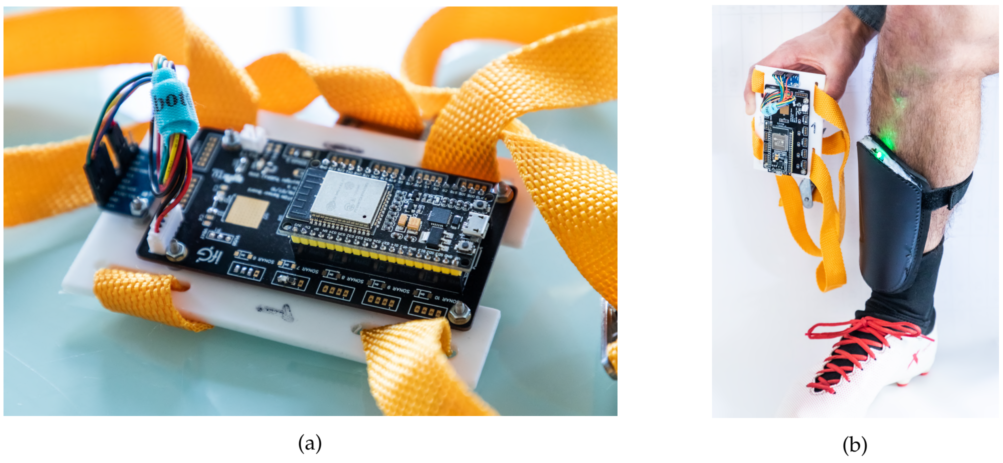


Figure 6.1.: Shin guard with mounted sensor (photo: Uni Graz/Konstantinov): (a) Hardware setup, (b) Hardware setup in comparison with future shin guard.

6.1.1. Settings

The output rate is set to 100 Hz. Additionally, an anti-aliasing filter is used to prevent aliasing or to satisfy the Nyquist theorem, respectively. An anti-aliasing filter is a low pass filter that removes spectral content, which is above the defined bandwidth. The used library supports several programmable digital low pass filter bandwidths: 5 Hz, 10 Hz, 20 Hz, 41 Hz, 92 Hz and 184 Hz. Since the output rate is 100 Hz, the selected digital low pass filter bandwidth is 41 Hz. The accelerometer range is set to its maximal range of ± 16 g. This corresponds to approximately ± 157 m/s^2 at latitude coordinate for Graz (1 g $\approx$ 9.807 m/s^2).

6.1.2. Time Synchronisation

Since the Activity Recognition (AR) is based on two IMUs, each attached to one shin guard, they must be synchronised with each other. The sensors are synchronised according to the Precision Time Protocol (PTP) [63]. This approach allows to synchronise precisely multiple devices in a network. The synchronisation accuracy is in the nanosecond range. The basic principle consists of defining synchronisation between a Master and a Slave clock.

The Master defines the time to which the Slave synchronises. Therefore, a sequence of four messages between the Master and Slave are sent. Those messages lead to four timestamps, denoted as $T1$, $T2$, $T3$ and $T4$. The first message is the initial sync message from Master to Slave. This results in the first two timestamps: $T1$ (transmission time) and $T2$ (reception time). This message is followed by a sync follow-up message from the Master. Now the Slave sends a delay request message to the Master that again results in two timestamps: $T3$ (transmission time) and $T4$ (reception time). The last message is a delay response from the Master. Based on the four timestamps, the Slave determines the latency between Master and Slave. The Slave clock adds the necessary offset to coincide with the Master clock.

One shin guard is defined as Master and one as Slave. The Master uses its internal clock as the reference time. The transmission of the synchronisation messages takes place via Wi-Fi.

6.1.3. Sensor Orientation and Placement on Shin Guard

The output data refers to a right-handed coordinate system with z-axis positive down. The coordinate system of the MPU-9250 is shown in Figure 6.2. The sensor, together with the hardware components, is located at the lower end of the shin guard. The placement of the MPU-9250 on the shin guard with labelled sensor axes is also illustrated in Figure 6.2. Note that the sensor is rotated relative to the shin guard.

6.2. Sensor Calibration

Since MEMS sensors are usually uncalibrated, it is necessary to calibrate them. This section deals with the calibration of the accelerometer from the MPU-9250 based on the approach represented in Chapter 2.4. In the course of the calibration process, the biases and scale factor error of each sensor axis are determined. Therefore, several calibration sets took place on different days from November 2019 until March 2020. They are listed in Table 6.1. For the determination of the sensor errors, the mean of 50

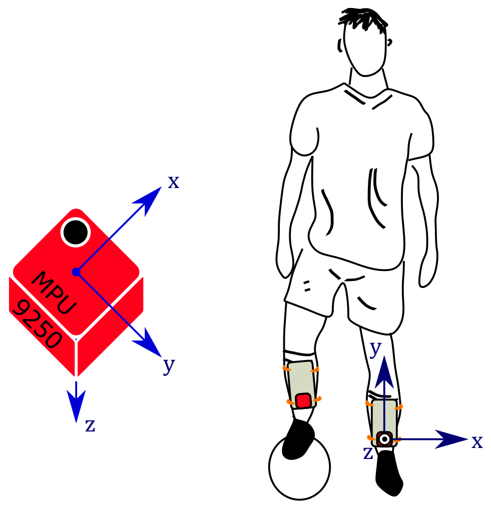


Figure 6.2.: Sensor orientation: (left) Orientation of MPU-9250. The black point identifies pin number 1. The pin out diagram diagram for MPU-9250 can be found in [61], (right) Placement of sensor on shin guard with labelled sensor axes.

samples per orientation are taken into account. The measurements until and inclusive 11. December 2019 are used to compute the biases and scale factors that are applied to the raw sensor data for the subsequent measurement campaigns (Chapter 6.3). The last two data sets are used to observe the long-time behaviour of the sensor errors.

Table 6.1.: List of calibration sets.

Data set	**Date [dd.mm.yy]**	**Sensor 1**	**Sensor 2**
1	12.11.19	✓	✓
2	19.11.19	✓	✓
3	16.11.19	✗	✓
4	11.12.19	✓	✓
5	29.01.20	✗	✓
6	05.03.20	✓	✓

The determined biases of all calibration sets are summarised in Figure 6.3. As it can be seen, Sensor 2 shows significantly larger bias offsets than Sensor 1. In particular, the y-component of Sensor 2 has a relatively large bias. Its mean value is 4.46 m/s^2. The largest bias variation in terms of standard deviation (one sigma) shows the z-axis of Sensor 2, namely ± 0.049 m/s^2, followed by the z-axis of Sensor 1 (± 0.035 m/s^2). The other axes show a variation between ± 0.005 m/s^2 up to ± 0.024 m/s^2. The determined scale factor errors vary from 0.992 to 1.000 for all calibration sets. The variations of the scale factor error are around 0.001. Thus, the bias represents the main error source.

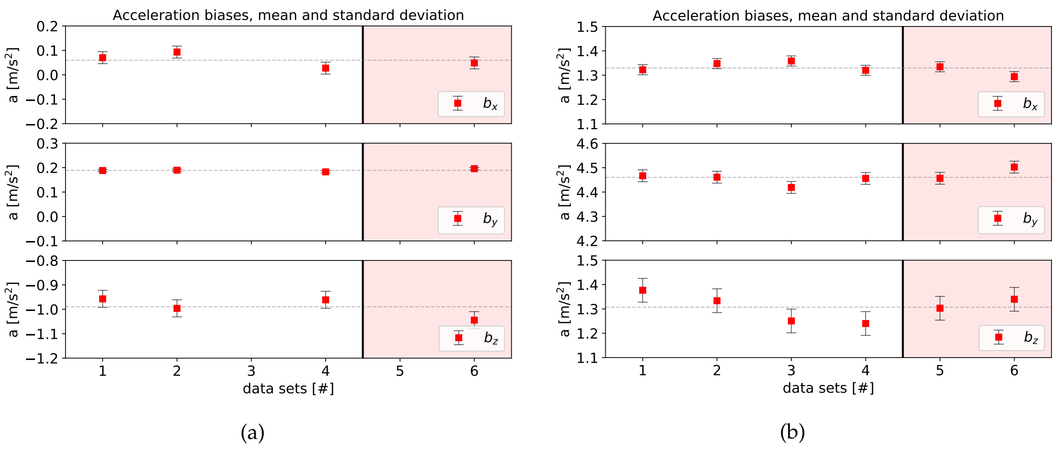


Figure 6.3.: Accelerometer biases inspired by Moder *et al.* [7]: (a) Sensor 1, (b) Sensor 2. The measurements until and inclusive data set 4 are taken into account to compute the biases and scale factors for the measurement campaigns.

As already mentioned, the final biases and scale factor errors are obtained using the mean bias and the mean scale factor errors from the first three or four calibration sets, respectively. Those values are listed in Table 6.2.

Table 6.2.: Biases and scale factor errors used for measurement campaigns.

		bias [m/s^2]	**scale factor error [-]**
Sensor 1	x	0.06	0.999
	y	0.19	0.998
	z	-0.97	0.994
Sensor 2	x	1.34	0.997
	y	4.45	0.999
	z	1.30	0.994

Figure 6.4 shows the total acceleration, once computed from the raw sensor data and once from the calibrated data. The data shown originate from the calibration set 4. The six different sensor orientations are well noticeable. The first two sensor orientations correspond to the z-pair, the next two to the x-pair and the last two to the y-pair. In the different static phases, the total acceleration should equal the gravitational acceleration ($\approx 9.807\ \mathrm{m/s^2}$). The deviations from the gravitational acceleration are significant. The standard deviations of the total acceleration, obtained from raw sensor data, are $\pm 0.565\ \mathrm{m/s^2}$ for Sensor 1 and $\pm 4.824\ \mathrm{m/s^2}$ for Sensor 2. After calibrating the sensor data, the variation is reduced to $\pm 0.013\ \mathrm{m/s^2}$ for both sensors.

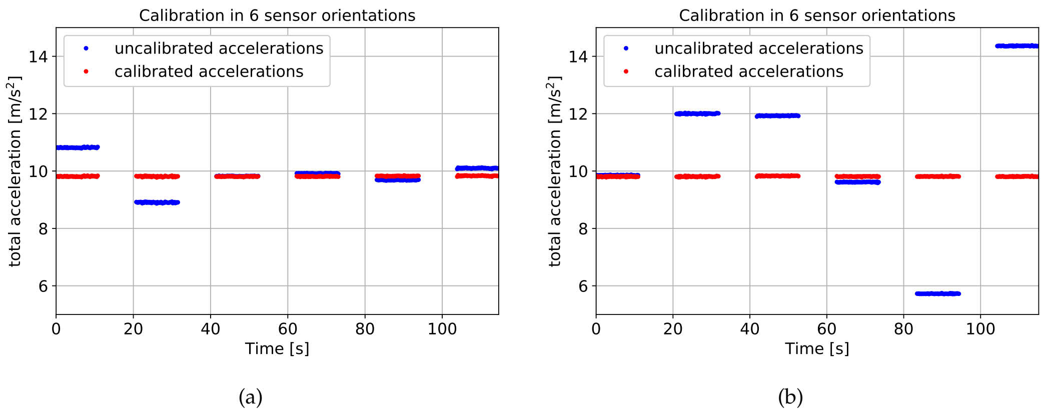


Figure 6.4.: Calibration measurements from calibration set 4: (a) Sensor 1, (b) Sensor 2. 50 samples per orientation.

6.3. Measurement Campaigns

The measurement campaigns took place on February 4 and March 11 in the year 2020 at the University Sports Centre Rosenhain (Graz). The activities, as defined in Chapter 5.1, were performed by Philipp Birnbaumer, MSc (Institute of Sports Science, Exercise Physiology, Training & Training Therapy Research Group, University of Graz). The author itself and Stefan Laller, Dipl.-Ing. (Institute of Geodesy, Working Group Navigation, TU Graz), were responsible for technical support and documentation.

The Institute of Sports Science developed standardised patterns. Each activity pattern was stored in a different log-file on the SD card. The average speed of walking is defined as 5.5 km/h. For running different speeds are recorded, namely 8 km/h, 12 km/h and 17 km/h, as well as one shuttle run. A shuttle run consists of running between marked lines and continually increasing the speed. In this case, the shuttle starts at 8 km/h and ends up at 14 km/h. The shuttle run also includes deceleration movements of the test person. Single passes are performed with different techniques (inside foot and outside foot). Two different distances (10 m and 20 m) are used to get different intensities of a pass. Passes are also recorded during running with different speeds (8 km/h and 12 km/h) to obtain more variation in the data. Shots are recorded using inside foot, outside foot and full instep. Passes and shots are done using the right as well as the left leg.

6.4. Segmentation

To obtain training and test data that can be fed into a ML algorithm, the inertial sensor data need to be segmented. The activities standing, walking and running are segmented using an overlapping sliding window of size W. The overlap is 50%. Concerning passes and shots, the window is set at the centre of the detected peaks and cut out accordingly. The corresponding labels are manually assigned. This results in a data set containing several time series of activity patterns of length W. In this thesis, two different window sizes W are investigated, namely 128 measuring points ($\hat{=}$1.28 s) and 256 measuring points ($\hat{=}$2.56 s). The distribution of the data set based on a window size of 256 measuring points is shown in Figure 6.5.

6.5. Data Smoothing

The next preprocessing step consists of smoothing the inertial data. The data are smoothed by fitting quintic spline functions [60] to each coordinate axis. Using quintic spline functions make it possible to reduce high-frequency noise while preserving the activity pattern. However, the smoothing process

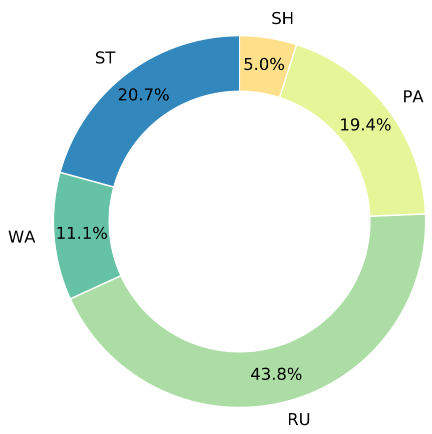


Figure 6.5.: Distribution of data set with a window size of 256 measuring points. The data set has a size of 1109 samples. The used abbreviations are listed in Table 5.1.

can also distort the activity pattern since sudden impacts can be mistaken as noise.

7. Wavelet Analysis

In this chapter, the acquired accelerometer data are investigated in more detail. A multiresolution wavelet analysis is applied to selected sensor data. Therefore, the Discrete Wavelet Transform (DWT) is used as a filter bank, as introduced in Chapter 3.6, which results in a *discrete transform plot* (Figure 3.7). However, the wavelet analysis includes the selection of a suitable wavelet. The chosen wavelet for this analysis is the *reverse biorthogonal 3.1* (rbio*3.1*) wavelet. It is symmetric, not orthogonal, but biorthogonal. The corresponding scaling function and wavelet function are shown in Figure 7.1.

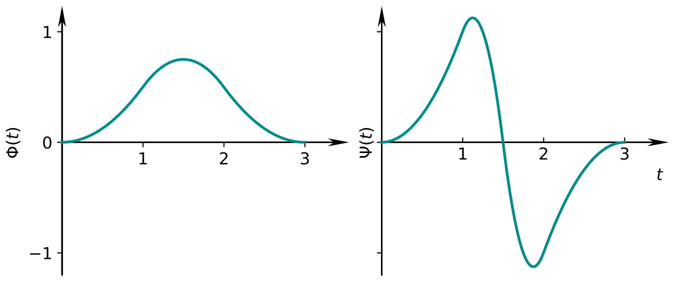


Figure 7.1.: Reverse biorthogonal 3.1: (right) scaling function, (left) wavelet function

The approximation and detail coefficients describe the similarity between the signal and the wavelet at a certain scale. Each scale corresponds to one decomposition level. The multilevel decomposition is done via the open-source wavelet transform software *PyWavelets 1.1.1* for *Python* [64]. An 1D DWT is applied to the data. The output is a list of coefficients arrays, containing the approximation coefficients array (cA_M) and the detail coefficients arrays ($cD_M, \ldots, cD_1$). The letter $M \geq 0$ indicates the level of decomposition. The level of decomposition depends on the desired frequencies to analyse. Another aspect is the filter length since the maximum level of decomposition is reached when the filter length of the wavelet becomes shorter than the input data length [64] (Equation 7.1). In this study, the chosen maximum decomposition level is 7. The corresponding frequency bands are listed in Table 7.1.

$$\max_{\text{level}} = \left\lfloor \log_2 \left(\frac{\text{length(data)}}{\text{length(filter)} - 1} \right) \right\rfloor \tag{7.1}$$

Table 7.1.: Frequency ranges corresponding to different decomposition levels. The sampling rate is 100 Hz.

		Frequency range [Hz]	
Coefficients arrays	**Scale indexing**	**to**	**from**
cD_1	$m = 1$	50	25
cD_2	$m = 2$	25	12.5
cD_3	$m = 3$	12.5	6.3
cD_4	$m = 4$	6.3	3.1
cD_5	$m = 5$	3.1	1.6
cD_6	$m = 6$	1.6	0.8
cD_7	$m = 7$	0.8	0.4
cA_7	-	0.4	0

Run and Shot Records

Running takes mainly place in the sagittal plane. Regarding the sensor orientation (Figure 6.2), the sagittal plane is defined by the y- and z-axes. Figure 7.2 shows the *discrete transform plot* of the difference between the vertical acceleration (y-component) of the left and right leg in the event of running. In total, the test persons runs a distance of 60 m three times. Between the running phases, the test person is at rest. Large positive coefficient values mean a strong similarity between the signal and the wavelet at a certain scale index. Thus, it is apparent that the decomposition level 5 and 6 mainly constitute running-phases.

Figure 7.3 shows the wavelet analysis of the total acceleration of two shot-records. The computation of the total acceleration is described in Formula 8.1. Only the shooting foot is considered. Between the shots, the test person walks. It is noticeable that the event of a shot is present in nearly all frequency bands. The walking phases are barely visible.

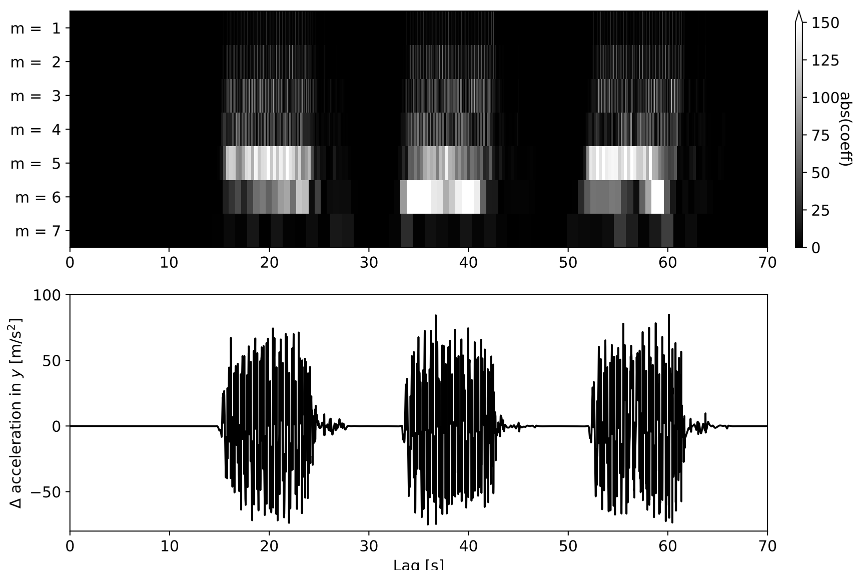


Figure 7.2.: Detail coefficients arrays of acceleration in y. Signal from the right leg is subtracted from the signal of the left leg. Test person runs with an average speed of 17 km/h.

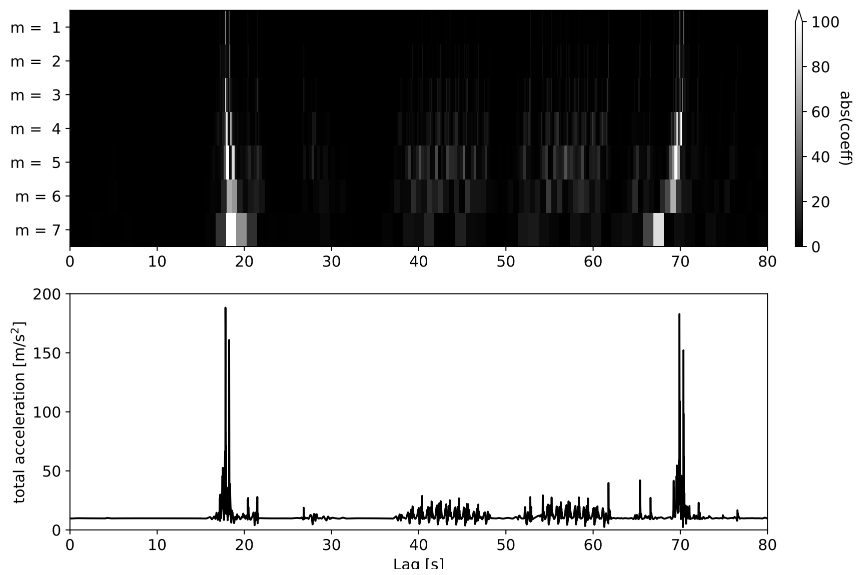


Figure 7.3.: Detail coefficients arrays of total acceleration of right leg during two shots.

8. Activity Recognition Strategy

This chapter deals with the activity recognition strategy. First of all, an overview of the different strategies of training and testing the selected Machine Learning (ML) algorithms are given. Second, the approach towards recognition of soccer specified activities is introduced as well as the chosen features. Third, the hyperparameters of the models are given.

8.1. Training and Testing

In total, four different models are compared with each other, namely Logistic Regression, Support Vector Machine (SVM), Random Forest Classifier and an Artificial Neural Network (ANN). A detailed description about the ML algorithms is given in Chapter 4.3. Figure 8.1 shows the flowchart of the training and testing process. Assuming that the data are already labelled and smoothed (Chapter 6.5), the data set is split into a training and a test set, as explained in Chapter 4.2. The next step is to compute the features for the training set as well as for the test set. Consequently, several transformation steps for the training data are performed, indicated by the box with the blue dotted lines. From the sequences of feature scaling and different feature selection methods, transformation values are generated and applied to the test data. This guarantees that the test data are not biased. For example, the features need to be scaled using *standardisation* (Formula 4.1). The mean and variance for standardisation is obtained from the training data and those values are used for the feature scaling process of the test data. However, that task also includes the selection of an appropriate feature subset. Three different approaches are taken into account:

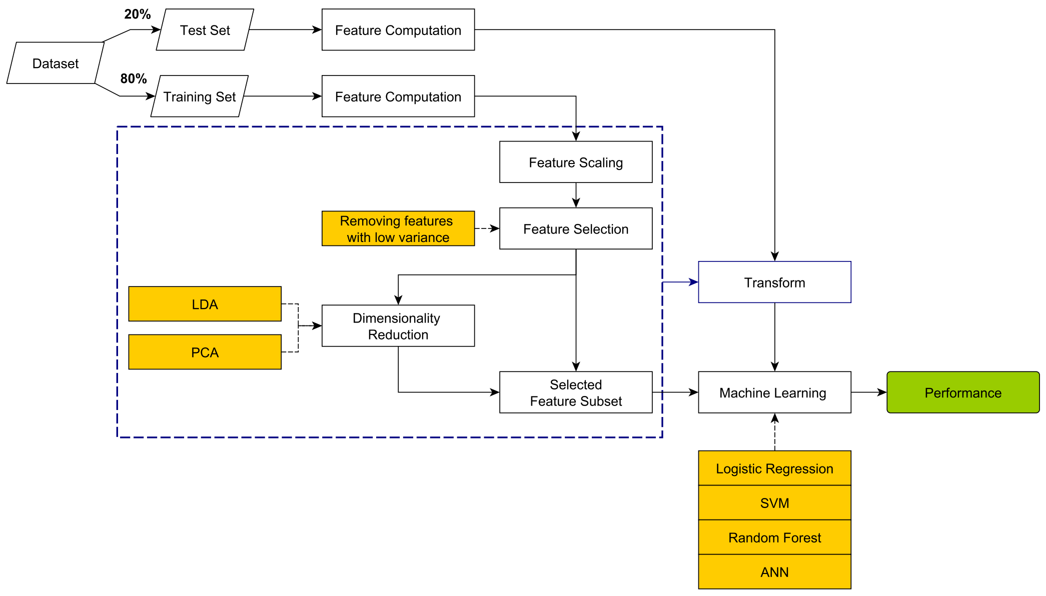


Figure 8.1.: Flowchart of Training and Testing the ML algorithm.

- **Variant 1: Removing features with low variance**
  When the variance of a feature is low or close to zero, respectively, it is likely that the feature contains approximately constant values. This means that this feature does not contribute to the separation of the classes. This step removes all features that are lower than a given threshold. In this case, the variance threshold is set to 0.24.

- **Variant 2: Removing features with low variance followed by PCA**:
  After removing features with low variance, an unsupervised dimensionality reduction algorithm is applied, namely PCA (Chapter 4.5.1). The number of components is chosen so that the total amount of variance explained is larger than 97.5% (Equation 4.41).

- **Variant 3: Removing features with low variance followed by LDA**
  After removing features with low variance, a supervised dimensionality reduction algorithm is applied, namely LDA (Chapter 4.5.2).

The ML block consists of training different algorithms with different feature subsets. The training step comprises hyperparameter tuning (Chapter 4.1.6) and the determination of the model parameters. The algorithm is trained using the training set. Additionally, since the input data is imbalanced (Figure 6.5), classes are weighted. The performance is measured by applying the trained model to the test set. The model outputs predicted labels. The comparison between the predicted labels and the actual labels gives an insight about the quality of the model.

8.2. Recognition of Soccer Activities

For a robust pattern recognition model of soccer activities, a classification construction scheme is introduced. Figure 8.2 gives an idea of it. Accelerometer data are acquired from the left and right leg. Subsequently, the data is preprocessed. This step includes segmentation and data smoothing by fitting a quintic spline. The construction scheme consists of two phases [65]: pre-classifier construction and the construction of the shot/pass-classifier as well as the standing/walking/running-classifier. The first phase (pre-classifier) aims to separate activities with a ball impact like passes and shots from activities with no ball possession, such as standing, walking and running. The second phase is specialised to determine the activity that the soccer player performs (shot/pass-classifier and standing/walking/running-classifier).

Due to separation into different classifiers, features can be customised more individually for the relevant activities. The features are obtained from the calibrated three-dimensional accelerometer measurements. All measurements refer to the *body-frame* (Chapter 2.5). The features are extracted from the segmented data set. Features are given in the time as well as in the frequency domain. However, the model scheme and the selected features for the individual classifiers will be explained in the following subsections. In general, the process of feature extraction includes the transformation processes like standardisation, removing features with low variance and applying the different dimensionality reduction algorithms (Chapter 8.1). Note that the transformation values are obtained from the training data and are applied to the new input data. The different classifiers are trained individually based on the corresponding features.

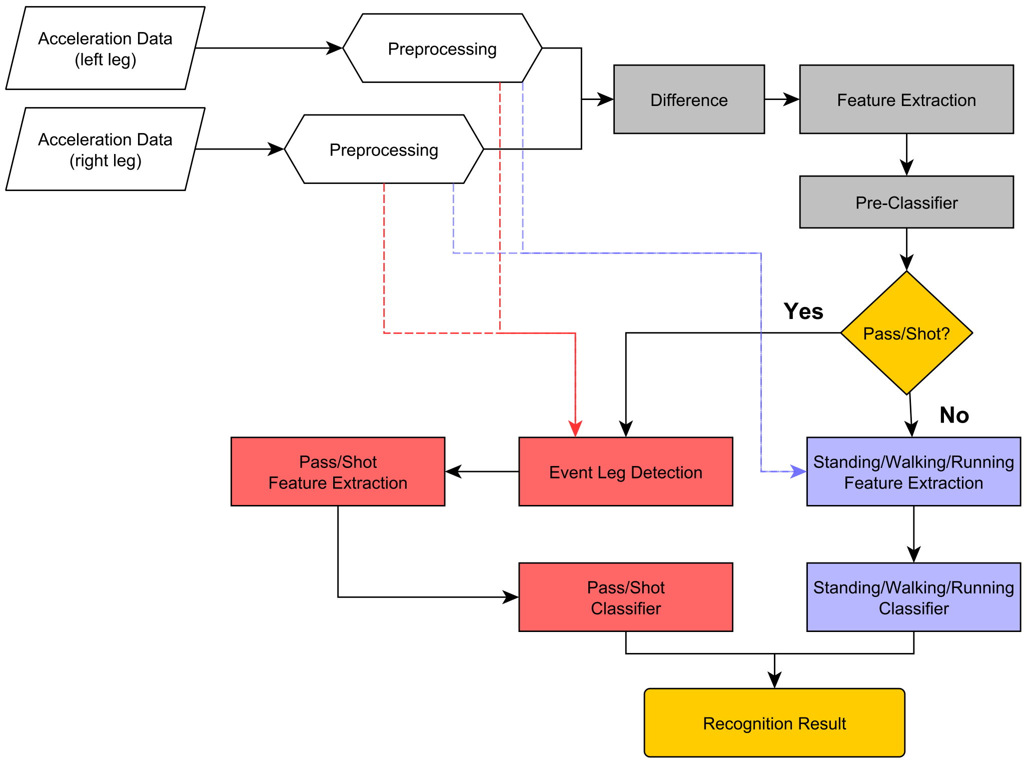

Acceleration Data (left leg)
Acceleration Data (right leg)
Preprocessing
Preprocessing
Difference
Feature Extraction
Pre-Classifier
Pass/Shot?
Yes
No
Event Leg Detection
Pass/Shot Feature Extraction
Pass/Shot Classifier
Standing/Walking/Running Feature Extraction
Standing/Walking/Running Classifier
Recognition Result

Figure 8.2.: Flowchart of Soccer Activity Recognition.

8.2.1. Pre-Classifier Construction

The pre-classifier separates passes and shots from the remaining activities. The whole pre-classification process is indicated with the grey boxes in Figure 8.2. Therefore, the preprocessed data of the right leg is subtracted from the preprocessed data of the left leg [3]. One the one hand, in event of a kick, the shooting leg produces a high peak in the data compared to the supporting leg. That peak should be still visible after subtraction. On the other hand, in event of standing, walking or running, the accelerometer data show a smooth motion of both feet. As a result, no extraordinary peaks are visible after subtraction. The pre-classifier makes a binary decision: if the signal refers to a pass or shot, then it is tagged as “1” (positive class), otherwise as “0” (negative class).

Feature Extraction

Features are extracted in the time and frequency domain. Each activity is represented by a signal of length W. From this window statistical values are selected, which should characterise each activity. The features in time domain are:

- **Largest and second largest value of total acceleration:** The total acceleration is computed as follows:

  $$a_{\text{total},i} = \sqrt{x_i^2 + y_i^2 + z_i^2}. \tag{8.1}$$

  The subscript i indicates the i^{th} element of the vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$. Those contain the accelerometer data of the corresponding sensitive axis. The length of the vector is equal to the window size W. From $\boldsymbol{a}_{\text{total}}$ the two largest values are extracted. These features consider the signal intensity of a kick and the pre- and post-impacts resulting from sharp decelerations of the kicking process (Figure 5.1). Peripheral regions of the signal (10% from the beginning and the end of the windowed time series) are neglected to prevent that stopping movements are misclassified as kick.

- **Signal Magnitude Area (SMA) of the transverse plane:** Only using the x and z component for the computation of the SMA should favour passes and shots since they mainly take place in the transverse plane. The SMA is introduced by Yang *et al.* [65]:

  $$\text{SMA} = \frac{1}{W}\left(\sum_{i=1}^{W} |x_i| + |z_i|\right). \tag{8.2}$$

  The subscript i indicates the i^{th} element of a vector.

In the frequency domain, wavelet-based filter banks are utilised (Chapter 3.6). The signal is decomposed into approximation and detail coefficients that correspond to different decomposition levels as shown in Chapter 7. Thus, they give a compact representation of the energy distribution in frequency and time. Depending on the chosen wavelet, the maximum level of decomposition can be obtained by Formula 7.1. For the pre-classifier the *reverse biorthogonal 3.1* (rbio*3.1*) wavelet is utilised. For a window length W of 256 data points, the maximum level of decomposition M is 6. The corresponding frequency bands are listed in Table 7.1. Note that the approximation coefficients array cover the remaining frequency range. In case of $W = 256$, the range is from 0 Hz to 0.8 Hz.

The following wavelet-based features are used for each of the three coordinate axes:

- **Maximum of absolute detail coefficient arrays in each subband (exclusive cD_1)**: The maximum value of the absolute detail coefficients array give information about the intensity of the activity.

- **Root Mean Square (RMS) of detail coefficients arrays in each subband (exclusive cD_1)**: These features provide insight into the variability of the signal in each frequency band. The RMS for the detail coefficients arrays cD_j, where $j \in \{2, \ldots, M\}$ and M denotes the maximum level of decomposition, is given as follows:

$$\mathrm{RMS}(cD_j) = \sqrt{\frac{1}{N_j} \sum_{i=1}^{N_j} cD_{j,i}^2}, \tag{8.3}$$

  where $cD_{j,i}$ is associated with the i^{th} component of the detail coefficients array. N_j is the length of the coefficients array at level j.

- **Mean and maximum of absolute approximation coefficients array and RMS**: The approximation coefficients contain information about low frequency components. Thus, the mean of the approximation coefficients array cA_M is defined as

$$\mathrm{MEAN}(cA_M) = \frac{1}{N} \sum_{i=1}^{N} |cA_{M,i}|, \tag{8.4}$$

  where N is the length of the approximation coefficients array. The maximum and RMS are computed analogous to the detail coefficients.

8.2.2. Standing/Walking/Running-Classifier Construction

When the pre-classifier decides that the signal is not an activity with ball contact, the model tries to assign the activity to standing, walking or running. The process of the standing/walking/running-classifier is represented by the blue boxes in Figure 8.2. The relevant features are computed separately for the sensor data of the left and the right leg. An appropriate feature subset is selected and fed into the trained model. Based on the input features, a decision is made whether the activity is recognised as standing, walking or running. So, the standing/walking/running-classifier represents a multiclass classifier.

Standing/Walking/Running Feature Extraction

In time domain, the maximum, the mean and the Interquartile Range (IQR) are computed for each sensor axis. The general formula for the mean of an arbitrary vector $\boldsymbol{a}$ of length W is

$$\text{MEAN}(\boldsymbol{a}) = \frac{1}{W}\sum_{i=1}^{W} a_i, \tag{8.5}$$

where a_i is the i^{th} element of the vector $\boldsymbol{a}$. The IQR is defined as

$$\text{IQR}(\boldsymbol{a}) = Q_3(\boldsymbol{a}) - Q_1(\boldsymbol{a}), \tag{8.6}$$

where $Q_1(\boldsymbol{a})$ is the first quantile and $Q_3(\boldsymbol{a})$ the third quantile of the vector $\boldsymbol{a}$.

Additionally, the SMA, based on all three sensor axis, is extracted:

$$\text{SMA} = \frac{1}{W}\left(\sum_{i=1}^{W} |x_i| + |y_i| + |z_i|\right). \tag{8.7}$$

The *daubechies* 2 *(db2)* wavelet is used to detect standing, walking and running. The RMS of the detail coefficients arrays of level $j \in \{5, 6, 7\}$ are computed and used as a feature. The chosen detail coefficients arrays cover a frequency range of 0.4 Hz to 3.1 Hz. Activities such as walking and running are mainly performed in the sagittal plane. Hence, the features are extracted from the y and z-component.

8.2.3. Pass/Shot-Classifier Construction

The pass/shot-classification is marked with the red boxes in Figure 8.2. The accelerometer data from the left and right leg are used. A preprocessing step consists of detecting the shooting leg or the event leg, respectively. This step is important since the differentiation of the shooting and supporting leg plays a crucial role in the classification process. This is achieved by the assumption that the peak of the total acceleration of the transverse plane (x and z-component) of the shooting leg is higher than the peak produced by the supporting leg. The shooting leg can be determined via:

$$R = \frac{\max \boldsymbol{a}_{\text{trans}}^{\text{left}}}{\max \boldsymbol{a}_{\text{trans}}^{\text{right}}} \begin{cases} \text{left leg}\,, & \text{if } R > 1 \\ \text{right leg}\,, & \text{otherwise} \end{cases}, \tag{8.8}$$

where

$$\begin{aligned} a_{\text{trans},i}^{\text{left}} &= \sqrt{x_i^{(\text{left})^2} + z_i^{(\text{left})^2}} \\ a_{\text{trans},i}^{\text{right}} &= \sqrt{x_i^{(\text{right})^2} + z_i^{(\text{right})^2}}. \end{aligned}$$

The y-component (vertical component) is neglected since the pre-impacts of the supporting leg, due to running-up to a shot, can falsify the results. Once the shooting leg is determined, the features for the pass/shot-classifier are computed and fed into the pass/shot-classifier. The pass/shot-classifier is a binary classifier and decides, whether the signal is associated with a pass or a shot.

Pass/Shot Feature Extraction

In time domain, the two largest values of the total acceleration of the transverse plane are extracted. Additionally, the Pearson correlation coefficient is computed from the total acceleration (Equation 8.1) of the event and supporting leg:

$$r = \frac{\sum_{i=1}^{W} \left(a_{\text{total},i}^{(\text{support})} - \bar{a}_{\text{total}}^{(\text{support})} \right) \left(a_{\text{total},i}^{(\text{event})} - \bar{a}_{\text{total}}^{(\text{event})} \right)}{\sqrt{\sum_{i=1}^{W} \left(a_{\text{total},i}^{(\text{support})} - \bar{a}_{\text{total}}^{(\text{support})} \right)^2} \sqrt{\sum_{i=1}^{W} \left(a_{\text{total},i}^{(\text{event})} - \bar{a}_{\text{total}}^{(\text{event})} \right)^2}}, \tag{8.9}$$

where $\bar{a}_{\text{total}}^{(\text{event})}$ and $\bar{a}_{\text{total}}^{(\text{support})}$ represent the mean values of event leg and the supporting leg, respectively. The Pearson's correlation gives information about the linear relationship of two variables. This feature is based on the fact that the behaviour of the supporting leg differs in case of a shot or in case of a pass, respectively. The intensity of the supporting leg during a shot is higher than during a pass. Thus, the correlation coefficient is also higher. In the frequency domain, the *rbio3.1* wavelet and the *discrete Meyer (FIR Approximation) (dmey)* wavelet are utilised for separating passes and shots. The maximum values of the absolute wavelet coefficients are extracted. The input data is the total acceleration (Equation 8.1). Thus, the distinction between these two classes takes place over the intensity of the kick.

8.2.4. Number of Features

Table 8.1 summarises the total number of features based on each feature selection variant, as introduced in Chapter 8.1. It can be seen that the number of features is significantly reduced after applying the dimensionality reduction algorithms (variants 2 and 3). Variant 3 applies the LDA as dimensionality reduction algorithm. Hence, the number of features is restricted by $K-1$ (K is the total number of classes). For example, the pass/shot classifier has to separate passes from shots. This means that after applying LDA to the feature space, only one feature remains.

Table 8.1.: Total number of features.

Variant	**1**	**2**	**3**
Pre-Classifier	42	12	1
Standing/Walking/Running-Classifier	32	10	2
Pass/Shot-Classifier	25	13	1

8.3. Model Hyperparameters

As already mentioned, in total four different ML models are investigated: Logistic Regression, SVM, Random Forest Classifier and an ANN. In the following section, the used hyperparameters are listed for each algorithm and each sub-classifier (pre-classifier, standing/walking/running-classifier, pass/shot-classifier). Additionally, the different feature selection methods (Chapter 8.1) result in a different number of features and can lead to different hyperparameters.

Table 8.2 shows the hyperparameters for Logistic Regression (Chapter 4.3.1). The hyperparameter C in Logistic Regression defines the inverse of the regularisation strength. Smaller values refer to stronger regularisation. The L_2-norm is used as a penalty. The hyperparameters are valid for all feature selection methods.

Table 8.2.: Model hyperparameters for Logistic Regression.

	C	**Penalty**
Pre-Classifier	100	L_2
Standing/Walking/Running-Classifier	1	L_2
Pass/Shot-Classifier	10	L_2

In Table 8.3 the hyperparameters for SVM are listed. The parameter C describes the regularisation parameter as introduced in Chapter 4.3.2. The parameter γ represents the kernel coefficient and refers to the RBF kernel.

Table 8.3.: Model hyperparameters for SVM.

Variant	**1**	**2**	**3**		
		kernel		**C**	γ
Pre-Classifier	linear	linear	linear	100	-
Standing/Walking/ Running-Classifier	linear	linear	linear	1	-
Pass/Shot-Classifier	RBF	RBF	linear	10	0.01

Table 8.4 summarised the used hyperparameters for the Random Forest Classifier. The number of estimators/trees depends on the number of features. Using one tree as an estimator, as it is the case of variant 3, is equivalent to the CART training algorithm (Chapter 4.3.3). As already mentioned, variant 3 uses the LDA for dimensionality reduction and the number of features is restricted by one less than the total number of classes. When only

one feature is available (Table8.1), then only one tree is necessary to make a decision.

Table 8.4.: Model hyperparameters for Random Forest.

Variant	**1**	**2**	**3**	**1**	**2**	**3**
	trees [#]			**maximum depth [#]**		
Pre-Classifier	30	10	1	4	4	2
Standing/Walking/ Running-Classifier	50	50	10	3	3	3
Pass/Shot-Classifier	50	50	1	3	3	2

The hyperparameters for the ANN (Chapter 4.3.4) are listed in Table 8.5. It describes the number of hidden layers, including the number of neurons per layer. All layers are fully connected. SGD with [Nesterov] momentum is chosen as the optimizer. Momentum can help the algorithm to converge faster. By usage of momentum technique, information from previous steps are included in the next estimation steps. The hyperparameter momentum also dampens fluctuations in the signal [66]. All models are trained with a momentum of 0.9. The corresponding learning rate η of the SGD optimizer is also given in Table 8.5. The activation function for all hidden layers is represented by the SELU function (Equation 4.31). *Early Stopping* is used to avoid overfitting of the neural net. The chosen ANN architectures are made as simple as possible since the size of the training data set is relatively small.

Table 8.5.: Model hyperparameters for ANN.

Variant	**1**	**2**	**3**	**1**	**2**	**3**	
	hidden layers [#]			**neurons per hidden layer [#]**			η **[-]**
Pre-Classifier	2	2	1	$(5,3)$	$(2,1)$	(4)	0.02
Standing/Walking/ Running-Classifier	4	4	4	$(3,3,2,2)$	$(1,1,1,1)$	$(3,3,2,2)$	0.01
Pass/Shot-Classifier	2	2	1	$(4,2)$	$(4,2)$	(4)	0.02

Part III
Model Evaluation and Results

9. Training and Testing

The performance of the different models is evaluated on the basis of the training and test sets. In the following sections, the evaluation scores

- Accuracy
- Precision
- Recall
- Area Under the Receiver Operating Characteristic Curve (ROC AUC)

of the trained models are discussed in detail (see Chapter 4.4). The performance on the training set is evaluated on each classifier individually. For selected classifiers, the decision boundaries are visualised. The display of the decision boundaries is only possible for the feature extraction variants that are based on dimensionality reduction algorithms. The test set is tested on the final construction scheme that is illustrated in Figure 8.2. The analysis is valid for a window size of 256 samples. However, the last section deals with analogous investigations on basis of a smaller window size.

In total, four different ML algorithms are investigated: the Logistic Regression classifier, the Support Vector Machine (SVM) classifier, the Random Forest classifier and the Artificial Neural Network (ANN). The different feature extraction variants are stated below:

- **Variant 1: Removing features with low variance**

- **Variant 2: Removing features with low variance followed by PCA**
  Principal Component Analysis (PCA) is an unsupervised dimensionality reduction algorithm.

- **Variant 3: Removing features with low variance followed by LDA**
  Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction algorithm.

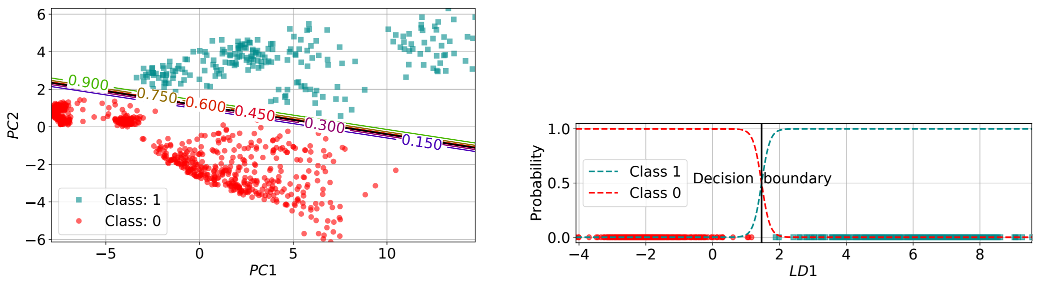
9.1. Training Models

The performance metrics for the training data of Logistic Regression, Support Vector Machine (SVM) and Random Forest are computed by using 10-fold cross-validation. The evaluation of the Artificial Neural Network (ANN) model takes place during the iterative training process. The performance of the ANN is monitored during the training process. The desired performance metrics can be displayed at each epoch until the algorithm converges to a solution. This results in so-called *learning curve* and gives information about possible problems of the model. An example of a learning curve is given in the following section.

9.1.1. Pre-Classifier

To evaluate the performance of the pre-classifier, several metrics are calculated: accuracy, precision, recall and the ROC AUC score. The Logistic Regression classifier and the SVM show perfect scores of 100% for all feature extraction methods. However, the Random Forest classifier and the ANN also show a high percentage in all performance metrics ($\geq$ 98.6%). That implies each classifier has a high ability to separate classes correctly. Figure 9.1 gives an insight about the operating mode of the Logistic Regression classifier. Class 1 corresponds to activities, which are associated with a pass or a shot. Class 0 refers to activities with no ball contact. The figure shows the training set, the decision boundaries and the model's estimated probabilities that an instance belongs to the positive class (Class 1): once represented by the first two Principal Component (PC) and once represented by the Linear Discriminant (LD). The black line indicates the positions where a fifty-fifty chance is given. The final models that bases on variant 2 are trained on all PCs and not only on the first two. The utilisation of the first two PCs only aims as a visualisation tool. The first two PCs constitute 83% of variance explained. Note that after applying LDA to the feature subspace, only one feature remains. As is can be seen, the two classes are well separable.

Figure 9.2 shows an example of the learning curves of the ANN where the features of variant 1 are fed into. The loss function computes the error in prediction. Thus, decreasing values of the loss function and increasing



(a) (b)

Figure 9.1.: Decision boundaries for Logistic Regression with probabilities: (a) using first two PCs as features, (b) using LD as feature.

values of the performance metrics imply that the model works well on the training instances. The learning curves for the other two variants also converge well after some iterations.

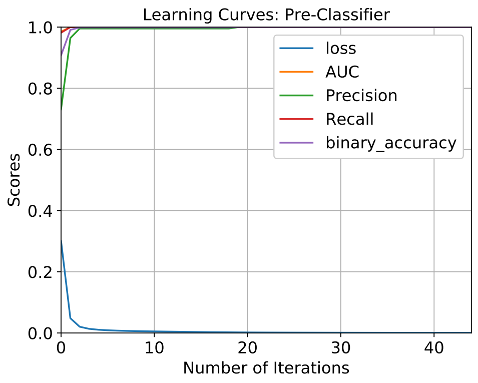


Figure 9.2.: Learning curves for pre-classifier (variant 1): performance metrics measured over each epoch. The binary accuracy counts how often predictions correspond to binary labels.

9.1.2. Standing/Walking/Running-Classifier

The evaluation metrics (accuracy, precision, recall, ROC AUC) for all classifiers and variants are all above 99%. Consequently, some computed decision boundaries are shown. The decision boundaries based on the LDs can be seen from the example of the SVM classifier and the Random Forest Classifier (Figure 9.3). The decision boundaries are indicated with the black solid lines. However, the discriminative character of the features is clearly visible.

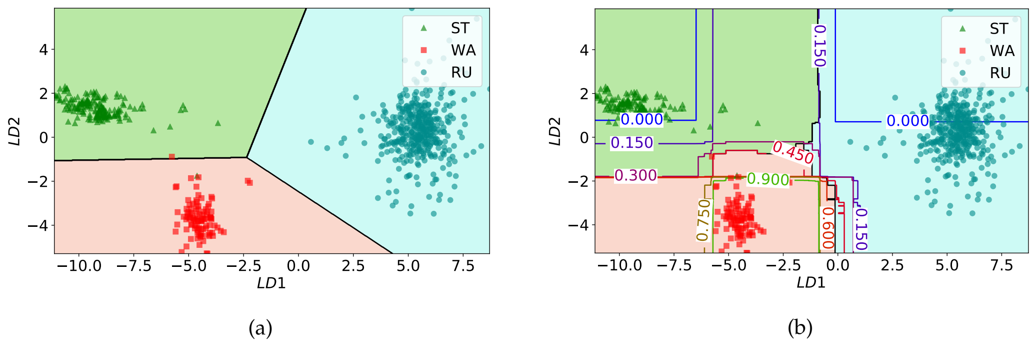


Figure 9.3.: Decision boundaries for standing/walking/running-classifier based on LDA: (a) SVM, (b) Random Forest.

Figure 9.4 contrasts the approaches, which are based on PCA. The probabilities for the class walking are marked. Since SVM does not provide estimated probabilities, only the decision boundary is highlighted. The decision boundaries show quite different growth patterns. The Random Forest classifier seems to slightly overfit the walking instances.

9.1.3. Pass/Shot-Classifier

For the given training set, the separation between passes and shots work well, although the training size is relatively small. The training set contains 172 passes and 44 shots. Logistic Regression, SVM and ANN show perfect scores in all metrics. Random Forest performs well for variant 2 and variant 3. Random Forest based on variant 1 performs worst with a recall score of 90%.

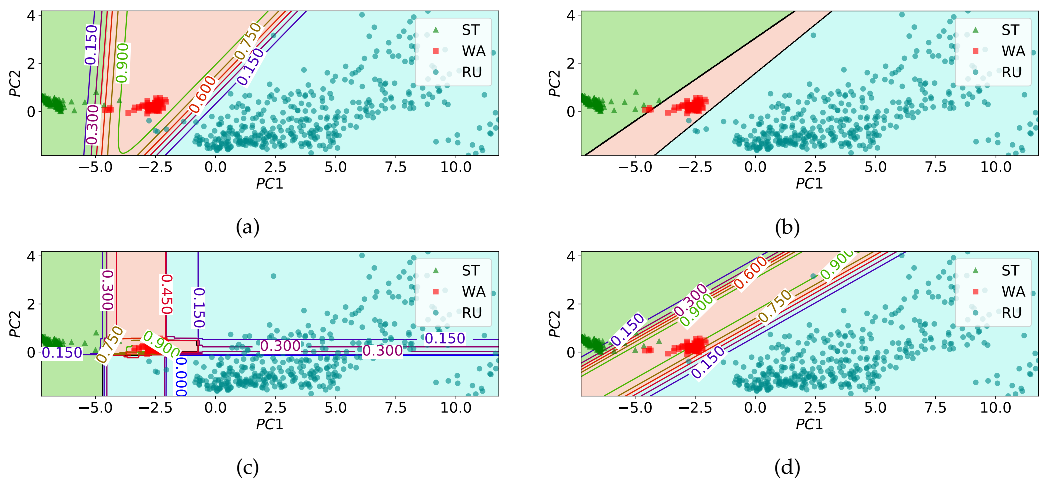
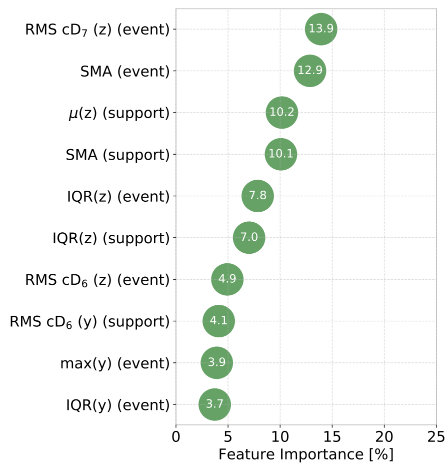


Figure 9.4.: Decision boundaries for standing/walking/running-classifier based on first two PCs (correspond to 91% of variance explained): (a) Logistic Regression, (b) SVM, (c) Random Forest Classifier, (d) ANN.

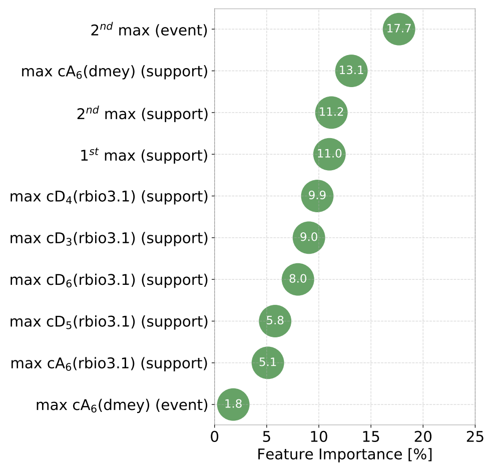
9.2. Feature Importance

Random Forest classifier also computes the relative importance of the input features. It measures how useful a feature is, based on the impurity on average. The computed feature importance scores of all features together sum up to 100%. Figure 9.5a illustrates the ten most important features according to the Random Forest classifier for the standing/walking/running classification process. Only features based on the y and z-component are present among the ten best features. This is the case since the the forward movements mainly affect the coordinates in the sagittal plane. The most important feature is the RMS of the detail coefficients array at decomposition level 7 [0.4-0.8 Hz] with almost 14%.

Figure 9.5b shows the measure of the then most important features corresponding to the pass/shot classification. It seems that actions of post- and pre-impacts (represented by the second largest values) are higher weighted than the actual ball impact. Another point is, that the features based on the supporting leg are more important than the features based on the event leg.



(a)



(b)

Figure 9.5.: Feature importance: (a) standing/walking/running-classifier. "*Left*" refers to data from the left leg. "*Right*" refers to data from the right leg. (b) pass/shot-classifier. "*Event*" refers to data from the event/shooting leg. "*Support*" refers to data from the supporting leg.

Regarding the feature importance of PCs, it can be said that features with a higher variance are tendentiously higher weighted than PCs with a lower one. The feature importance concerning the LDs is promptly explained. Only one feature is fed into the pass/shot-classifier. Thus, the one LD contributes 100% of the decision process. The LDA, applied on the feature subset related to the standing/walking/running-classifier, results in two LDs. The relevance of the first and second LD is almost equally distributed: $LD1$ makes up 56% and $LD2$ 44%.

9.3. Testing Models

The evaluation scores of the test set are listed in Table 9.1. All metrics, except for the accuracy score, are based on the computation of the macro-average. The accuracy score refers to the overall accuracy. In general, all models work reasonably well on the training as well as on the test set. Thus, no clearly recognisable link would indicate that a model outperforms.

Table 9.1.: Evaluation scores of test set using a window size of 256 samples.

	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	**1**	100.0	100.0	100.0	100.0
	2	99.5	99.5	98.2	99.0
	3	99.5	98.3	99.5	99.7
SVM	**1**	100.0	100.0	100.0	100.0
	2	99.5	99.5	98.2	99.0
	3	100.0	100.0	100.0	100.0
Random Forest	**1**	98.6	97.3	97.5	98.6
	2	99.1	99.1	97.7	98.8
	3	100.0	100.0	100.0	100.0
ANN	**1**	99.1	96.9	99.1	99.4
	2	99.5	98.3	99.5	99.7
	3	99.5	98.3	99.5	99.7

9.4. Different Window Size

In this section, the influence of a smaller window size is investigated. The chosen window size comprises 128 samples. That complies with 1.28 s. A smaller window size results in a different distribution of the features. One reason is that the maximum level of decomposition regarding the wavelet decomposition is lower due to the lower number of samples in the data. However, the classifier trained on the smaller window size performs at the same level as using a classifier trained with the window size of 256 samples. The corresponding evaluation scores are given in Table 9.2. The vast majority of the models show evaluation scores above 99%.

Table 9.2.: Evaluation scores of test set using a window size of 128 samples.

	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	**1**	99.8	99.6	99.9	99.9
	2	99.5	99.1	99.8	99.9
	3	99.8	99.5	98.2	99.1
SVM	**1**	99.8	99.6	99.9	99.9
	2	99.5	99.1	99.8	99.9
	3	99.5	99.1	98.1	99.0
Random Forest	**1**	98.7	96.4	99.2	99.4
	2	97.3	96.1	96.6	97.9
	3	100.0	100.0	100.0	100.0
ANN	**1**	99.8	99.6	99.7	99.9
	2	99.3	97.5	99.4	99.6
	3	100.0	100.0	100.0	100.0

10. Recognition of Soccer Activities

This chapter deals with the final evaluation of the different models with different window sizes. As is had been shown in Chapter 9, all models and variants have the potential to detect standing, walking, running, passing and shooting. It has been proven that the chosen features and hyperparameters work. So, the final model is trained on the whole data set (training set plus test set) and is validated on new data to analyse the performance in real applications. The new data is declared as *validation set*. The validation set consists of five sequences, which contain various soccer-specific elements.

10.1. Validation Set

The validation set chains together five sequences. Figure 10.1 shows the corresponding time series that is represented by the total accelerations of the sensor data from the left and right leg. The different sequences are divided by the grey dotted lines. One sequence consists of:

- Standing ($\approx$ 1 min)
- Running with increasing speed continually (8 km/h up to 12 km/h). During the running phase two passes and one shot are performed:
  - Pass during running with 8 km/h
  - Pass during running with 10 km/h
  - Shot during running with 12 km/h

  The running phase lasts for about 45 s.
- Short pause of approximately 12 s.

- Walking with an average speed of 4 km/h. During the walking phase, one pass is done. This phase lasts for about 36 s.

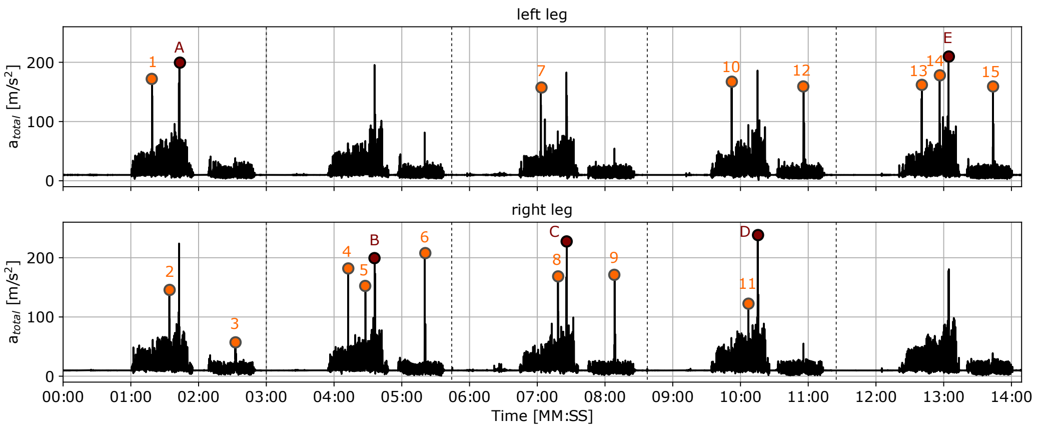


Figure 10.1.: Validation set.

Each sequence consists of different types of passes and shots, executed with the left or the right leg. Shots are indicated by red dots and letters (Figure 10.1) and characterised in Table 10.1. The location of the dots in the subplots gives information about the leg that performed the kick. In total, 5 shots had been performed.

Table 10.1.: Description of shots. In each sequence one shot is present.

Letter	**A**	**B**	**C**	**D**	**F**
Leg	right	right	left	right	left
Type	full instep	full instep	full instep	outside foot	full instep

Passes are also done in two different distances, namely 10 m and 20 m. The passes are marked by orange dots, numbered from 1 to 15 and described in Table 10.2. From Figure 10.1, it is already noticeable that the main difference between passes and shots is the activity of the supporting leg.

Table 10.2.: Description of passes.

Sequence 1	**Point number**	**1**	**2**	**3**
	Leg	left	right	right
	Type	insight foot	insight foot	insight foot
	Distance [m]	10	10	10
Sequence 2	**Point number**	4	5	6
	Leg	right	right	right
	Type	outside foot	insight foot	insight foot
	Distance [m]	10	10	20
Sequence 3	**Point number**	**7**	**8**	**9**
	Leg	left	right	right
	Type	insight foot	outside foot	insight foot
	Distance [m]	10	10	20
Sequence 4	**Point number**	**10**	**11**	**12**
	Leg	right	right	left
	Type	outside foot	insight foot	insight foot
	Distance [m]	10	10	20
Sequence 5	**Point number**	**13**	**14**	**15**
	Leg	left	left	left
	Type	insight foot	insight foot	insight foot
	Distance [m]	10	10	20

10.2. Comparison of Classifiers and Feature Extraction Methods

The data from the validation set is fed into the classification construction scheme. The corresponding flowchart is illustrated in Figure 8.2. The data are segmented using a window size W of 256 samples ($\hat{=}$2.56 s) and an overlap of 154 samples that equals to exactly 60.15625%. In this section, the different models and feature extraction methods are compared with each other.

The segmented validation set has been manually labelled to allow a comparison between the predicted and the actual labels. Due to the overlapping windows, passes or shots can be present in consecutive windows. Figure 10.2 shows, as an example, the Shot C that is visible in three consecutive windows. The definition of a shot includes pre- and post-impacts. Therefore, all three time-series are labelled as shot.

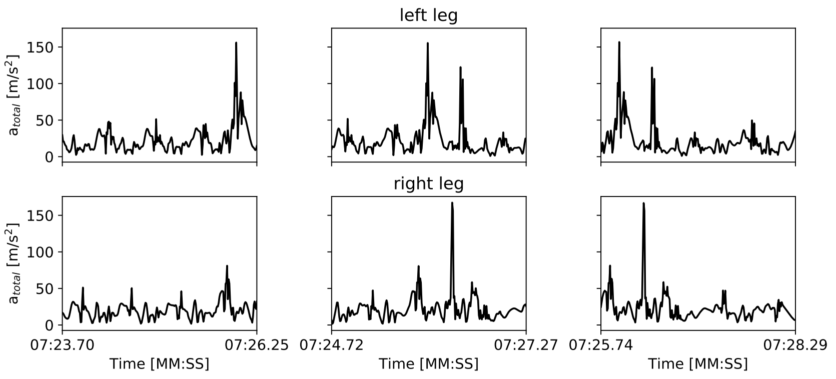
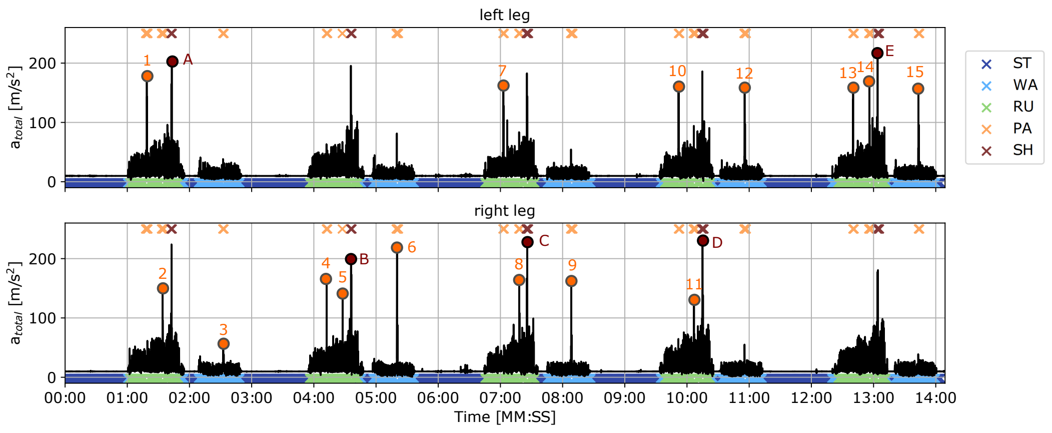


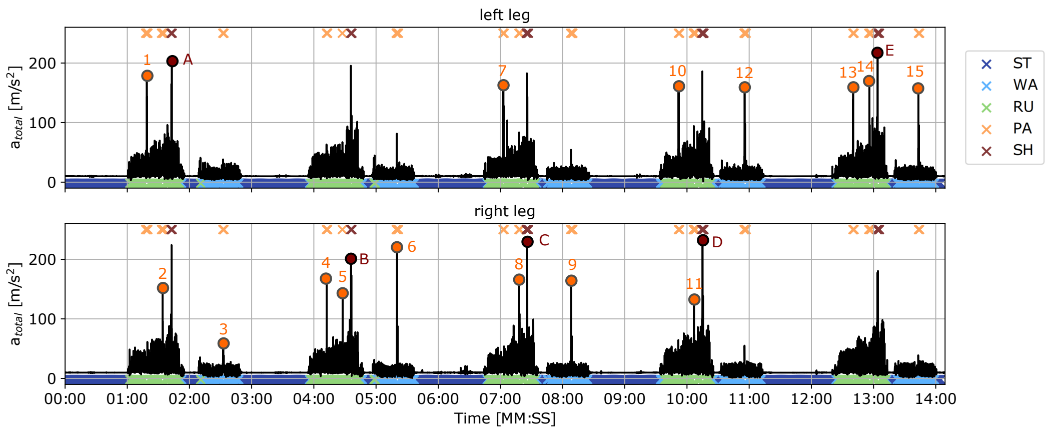
Figure 10.2.: Shot in overlapping windows: shown by the example of Shot C.

10.2.1. Variant 3

The classification scheme is applied to the validation set using the different feature extraction variants. The analysis has shown that the ML algorithms combined with the Linear Discriminant Analysis (LDA) as dimensionality reduction algorithm (variant 3) outperforms the other feature extraction methods. However, in Figure 10.3a, the reference solution is visualised. The crosses refer to the actual labels. The output from the standing/walking/running-classifier is visualised "under" the time series, the output from the pass/shot-classifier "above" of it. Figure 10.3b shows the recognition result of Logistic Regression based on variant 3. Now, the crosses refer to the predicted labels. From a purely visual point of view, all passes and shots are correctly detected. Upon closer inspection of the illustration, it is noticeable that the transition areas between standing, walking and running have faults. However, the confusion matrix provides a more detailed analysis of the model's performance. The corresponding confusion matrix is given in Figure 10.8c (Chapter 10.2.4). The confusion matrix confirms that some misclassification between standing, walking and running took place as well as that passes and shots are perfectly detected.



(a)



(b)

Figure 10.3.: Recognition result: (a) reference solution (b) Logistic Regression (variant 3). The abbreviations used in the legend are listed in Table 5.1.

From the confusion matrix several evaluation metrics, like accuracy, precision, recall and ROC AUC are derived, which are summarised in Table 10.3. In general, perfect scores are indicated with a green background. Scores between 81% and 61% are filled with an orange background. Every score under 61% is marked with a red background. In addition, the algorithm with the best overall-performance is also highlighted with green. The overall performance represents the overall accuracy and the macro-averages of the precision, recall and ROC AUC scores.

Table 10.3.: Evaluation scores for variant 3 with window size $W = 256$ samples. The overall performance for accuracy score ≙ overall-accuracy. The overall performances of precision, recall and ROC AUC score ≙ macro-average.

	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	**ST**	95.7	91.8	99.2	96.0
	WA	95.3	97.8	83.4	91.4
	RU	98.9	98.5	98.0	98.6
	PA	100.0	100.0	100.0	100.0
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		94.9	97.4	96.1	97.2
SVM	**ST**	94.1	85.5	99.7	94.6
	WA	93.8	100.0	75.8	87.9
	RU	98.8	96.5	98.5	98.7
	PA	100.0	100.0	100.0	100.0
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		93.4	97.0	94.8	96.2
Random Forest	**ST**	95.7	91.8	92.2	96.0
	WA	95.4	98.3	83.4	91.5
	RU	98.8	97.0	98.0	98.5
	PA	100.00	100.0	100.0	100.0
	SH	100.00	100.0	100.0	100.0
Overall Performance [%]:		94.9	97.4	96.1	97.2
ANN	**ST**	95.5	91.6	99.2	95.9
	WA	95.5	97.3	84.8	92.0
	RU	98.7	99.5	94.9	97.4
	PA	99.4	87.8	100.0	99.7
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		94.6	95.2	95.8	97.0

The table shows that nearly all algorithms, except for ANN, could recognise all passes and shots. However, the main difference between the different algorithms is the separation of standing, walking and running. The main faults take place in the transition areas, as seen in Figure 10.3. The lowest recall score has the SVM classifier for the class walking, but shows a perfect precision score. That means that the ability of correctly assigning a walking instance to the class walking is approximately 3 : 4. The accuracy of the positive predictions is, however, 100%. The best overall performance metrics is shown by the Logistic Regression classifier and the Random Forest classifier. Both classifiers provide the same result. In general, all misclassification regarding standing, walking and running happened in the transition areas of those activities. For example, the first steps of a running movement or deceleration movements show errors. Therefore, they are not considered as serious errors since the switching between standing, walking or running is a fuzzy process.

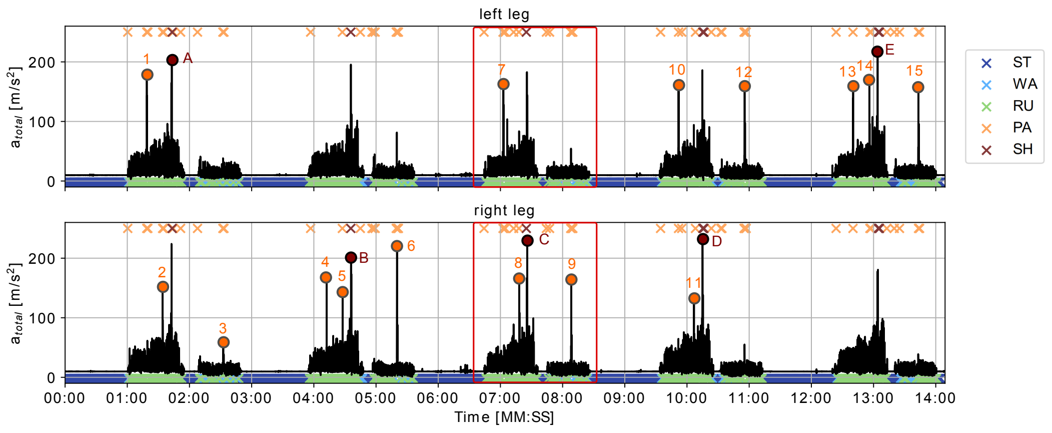
10.2.2. Variant 2

Variant 2 deals with the feature extraction method that bases on Principal Component Analysis (PCA). In Table 10.4, the corresponding evaluation metrics are listed. It is noticeable that the Random classifier has the lowest scores. The recall-score of walking and shot as well as the precision of passing are under 54%. That means that the algorithm's predictions are rather random. The SVM classifier and ANN classifier have as well troubles in detecting walking. The best performance shows the Logistic Regression classifier. However, compared to the solution that bases on a supervised LDA dimensionality reduction algorithm, the overall performance has become worse.

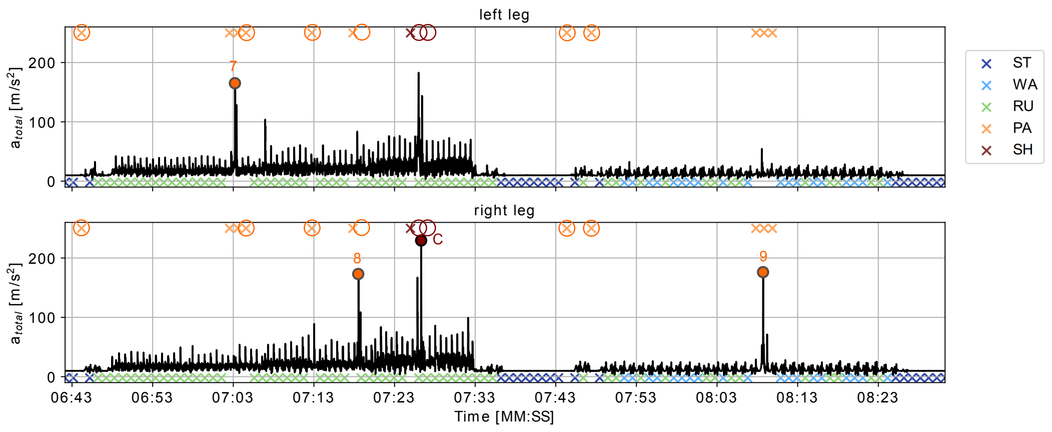
Figure 10.4a illustrated the computed recognition result of the Random Forest classifier. It is visible that walking is mixed up with running as well as that there are too many detected passes in the transition zones. Another issue is that passes and shots are not recognised clearly in means of detecting a pass or shot in consecutive overlapping windows, as shown in Figure 10.4b. The red and orange circles refer to false classifications of shots and passes, respectively.

Table 10.4.: Evaluation scores for variant 2 with window size $W = 256$ samples

	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	**ST**	96.7	93.9	99.2	97.0
	WA	95.9	95.4	88.2	93.3
	RU	98.3	96.9	95.9	97.5
	PA	99.4	97.0	88.9	94.4
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		95.2	96.6	94.4	96.4
SVM	**ST**	92.5	88.9	95.2	92.8
	WA	91.9	88.3	78.7	87.6
	RU	97.6	94.4	95.4	96.8
	PA	99.2	96.8	83.3	91.6
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		90.6	93.7	90.5	93.7
Random Forest	**ST**	95.5	93.3	97.0	95.7
	WA	84.7	97.7	40.8	70.2
	RU	86.4	64.0	96.9	90.0
	PA	96.0	53.1	72.2	84.7
	SH	99.3	100.0	53.8	76.9
Overall Performance [%]:		80.9	81.6	72.2	83.5
ANN	**ST**	88.2	79.3	99.7	89.2
	WA	87.3	93.4	54.0	76.4
	RU	98.1	97.4	94.4	96.8
	PA	99.4	94.3	91.7	95.7
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		86.5	92.9	88.0	91.6



(a)



(b)

Figure 10.4.: Recognition result: Random Forest (variant 2): (a) Recognition result, (b) Zoom of Figure 10.4a. Errors regarding pass/shot classification are highlighted.

The inability of the Random Forest classifier in detecting walking can be explained in the characteristics of its decision boundary. Figure 10.5 shows the decision boundary of the trained model based on the first two Principal Components (PC). The plotted points represent the transformed points of the validation set, thus, the reference solution. Looking closer at the encircled area, it is noticeable that the walking instances are partly present in the decision regions of walking and running. The most misclassifications took place in this area. The reason is that the model is trained on walking instances that are performed on an average speed of 5.5 km/h and the walking phase of the validation phase is performed at an average speed of 4 km/h.

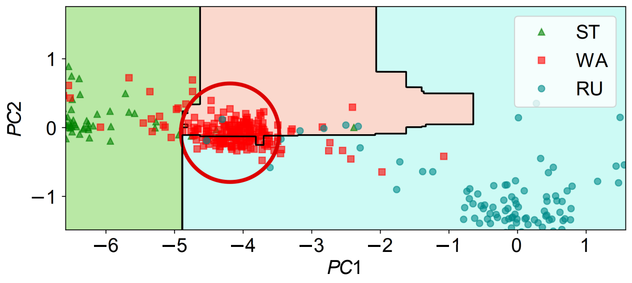


Figure 10.5.: Decision boundary of Random Forest (variant 2) with plotted validation set. Encircled area highlights the problem area in walk detection.

In comparison to the solution of the Random Forest classifier, the Logistic Regression classifier (Figure 10.6) outperforms, although Pass 8 has not been detected.

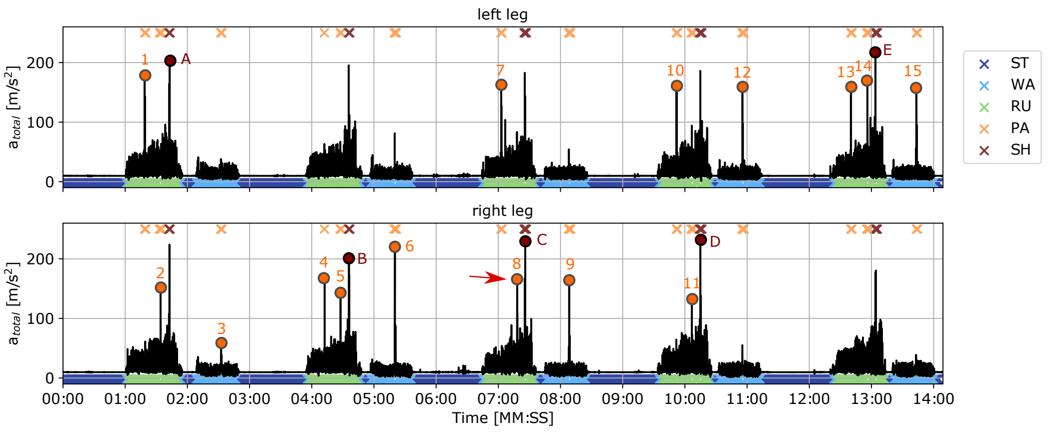
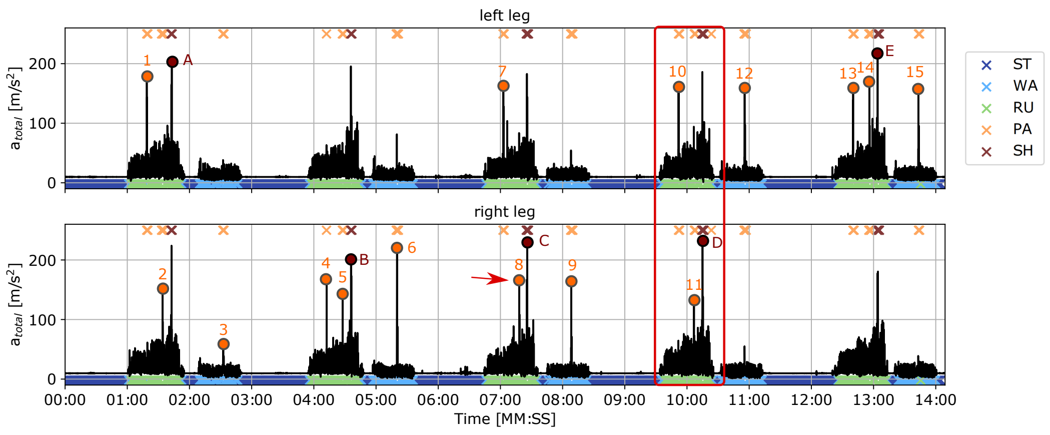


Figure 10.6.: Recognition result: Logistic Regression (variant 2).

10.2.3. Variant 1

The classification scheme is applied to the validation set using the first feature extraction variant. Variant 1 implies that features with a low variance are excluded. No dimensionality reduction algorithm is applied. Figure 10.7a shows the recognition result of the ANN, based on variant 1. As it can be seen, Pass 8 has not been detected. Furthermore, a stopping movement has been misclassified as pass in the running phase of the fourth sequence (indicated by the red box). Another issue is that passes are not recognised clearly in means of detecting a pass in consecutive overlapping windows. The corresponding confusion matrix is given in Figure 10.8j. None of the algorithms have recognised all passes. Logistic Regression, SVM and ANN had overlooked Pass 8. Random Classifier detected Pass 8, but missed Pass 5. The ANN classifier is the only classifier that mistook a stopping movement for a pass.



(a)

Figure 10.7.: Recognition result of ANN (variant 1).

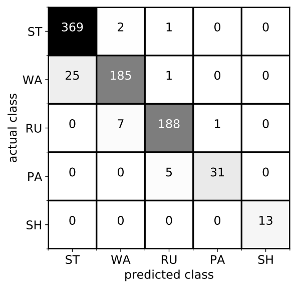
Table 10.5 gives information about the performance of the different models. Every algorithm, except for the Random Forest, can detect shots including pre- and post-impacts. Logistic Regression provides the best overall performance, although the assumption of no or little collinearity of the features is violated. Compared to variant 3 (Table 10.3), however, the overall-performance has deteriorated.

Table 10.5.: Evaluation scores for variant 1 with window size $W = 256$ samples

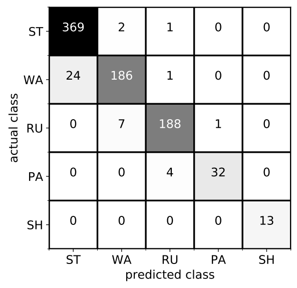
	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	**ST**	96.6	93.7	99.2	96.9
	WA	95.8	95.4	87.7	93.1
	RU	98.2	96.6	95.9	97.4
	PA	99.3	96.9	86.1	93.0
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		94.9	96.5	93.8	96.1
SVM	**ST**	95.2	90.9	99.2	95.5
	WA	94.4	97.7	80.1	89.7
	RU	97.7	92.8	98.0	97.8
	PA	99.2	100.0	80.6	90.3
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		93.2	96.3	91.6	94.7
Random Forest	**ST**	97.1	94.4	99.5	97.3
	WA	96.1	95.9	88.6	93.7
	RU	97.6	94.9	94.9	96.7
	PA	98.8	88.2	83.3	91.4
	SH	99.8	100.0	84.6	92.3
Overall Performance [%]:		94.7	94.7	90.2	94.3
ANN	**ST**	95.5	91.4	99.5	95.9
	WA	94.4	95.1	82.5	90.5
	RU	98.3	96.9	95.9	97.5
	PA	99.2	93.9	86.1	92.9
	SH	100.0	100.0	100.0	100.0
Overall Performance [%]:		93.7	95.5	92.8	95.4

10.2.4. Confusion Matrices

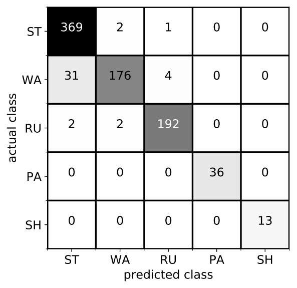
This section compares the evaluated models based on the confusion matrices. The confusion matrices create more transparency regarding the errors made in the classification process. All evaluation metrics from the Table 10.3 to 10.5 are derived from those matrices.



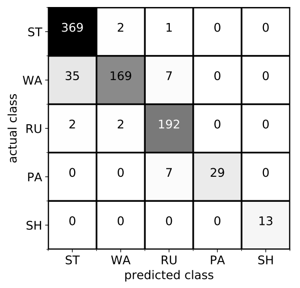
(a) Logistic Regression: variant 1



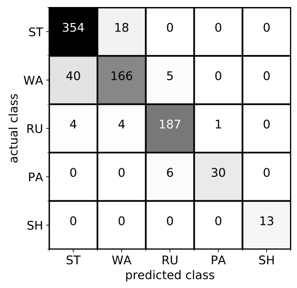
(b) Logistic Regression: variant 2



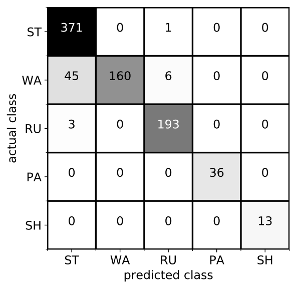
(c) Logistic Regression: variant 3



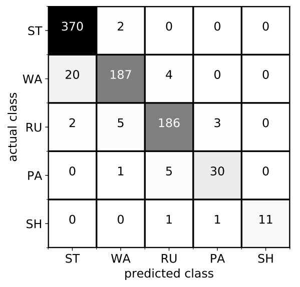
(d) SVM: variant 1



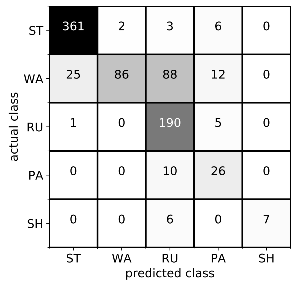
(e) SVM: variant 2



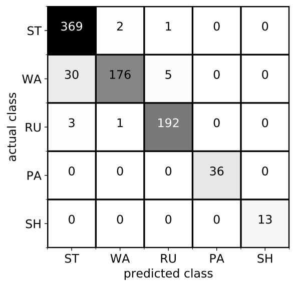
(f) SVM: variant 3



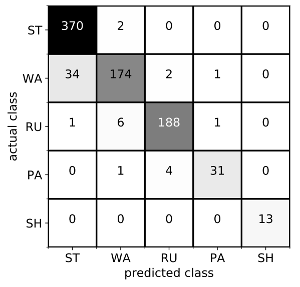
(g) Random Forest: variant 1



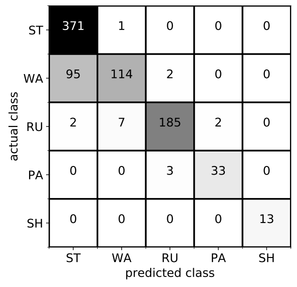
(h) Random Forest: variant 2



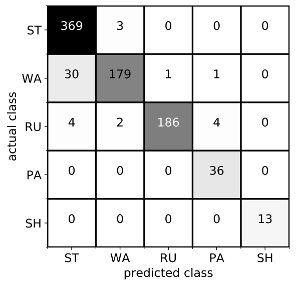
(i) Random Forest: variant 3



(j) ANN: variant 1



(k) ANN: variant 2



(l) ANN: variant 3

Figure 10.8.: Confusion matrices for validation set with a window size of 256 samples. In total, 12 different models are evaluated. The rows compare the feature extraction variants for the same algorithm. The columns compare the algorithms with the same feature extraction variant.

10.3. Different Window Size

This section deals with the effect on real applications with a window size half it's former size. That corresponds to $W = 128$ samples or 1.28 s. The analysis is analogous to the previous chapter. The evaluation has shown that a smaller window size results in an insufficient detection of shots. Table 10.6 shows the precision and recall scores for the shot-recognition. Although the precision values are relatively high, the recall-scores are not satisfying. Due to a smaller window size, pre- and post-impacts of a shot are often misclassified as pass or not even identified as a kick. So, a clean detection of a shot is not possible. A complete list of the evaluation scores can be found in Appendix A. The corresponding confusion matrices are summarised in Appendix B.

Table 10.6.: Precision and recall scores of validation set for class shot using a window size of 128 samples.

	Variant	Precision [%]	Recall [%]
Logistic Regression	**1**	100.0	80.0
	2	92.3	80.0
	3	100.0	60.0
SVM	**1**	100.0	66.7
	2	100.0	73.3
	3	100.0	60.0
Random Forest	**1**	100.0	66.7
	2	85.7	40.0
	3	100.0	60.0
ANN	**1**	92.3	80.0
	2	92.3	80.0
	3	83.3	66.7

10.4. Real-Time Capability

The following section investigates, whether the presented AR-scheme for soccer-specific activities can be carried out in real-time, or not. Therefore, the real-time capability is tested based on two instances. One instance belongs to the class running and the other to the class shot. The duration of time is measured that is needed to get a recognition result: time needed to preprocess data and time needed to get the final prediction. The test is carried out with the Logistic Regression classifier as ML algorithm based on variant 3 (removing features with low variance followed by LDA) with a window size of 2.56 s since this combination has shown one of the most reasonable solutions. The final computation time was determined by using the average value of five times repeating the AR process[1]. The results are listed in Table 10.7. It can be seen that the classification process only represents a small part of the whole classification process. The main part is the data smoothing by fitting quintic spline functions. The duration of the data smoothing depends on the complexity of the signal. Thus, a signal that contains a shot takes longer to be smoothed than a running instance, for example.

Table 10.7.: Computation time (averaged values from five repetitions).

	Running	**Shot**
Data Smoothing [ms]	18.7	36.6
Classification Process [ms]	2.1	2.2
Total [ms]	20.8	38.8

The classification process itself also takes longer for an instance that belongs to a shot. A detailed breakdown of the classification process is shown in Figure 10.9. The pre-classification is the same for both activities. It comprises the computation of the features, followed by the transformation, and the prediction of the label. The feature transformation includes feature scaling, removing features with low-variance and applying the transformation matrix that is determined from the LDA. After the recognition result of the pre-classifier is available, the classification process splits. The running instance undergoes the standing/walking/running-classification scheme. The shot instance goes through the pass/shot-classification scheme. Both

[1]Notebook: ASUS A55V, CPU: Intel Core i7-3610QM (2.3 GHz), Memory: 8 GB, OS: WIN10 Pro

also include the feature computation, feature transformation and the prediction of the class. However, for the pass/shot detection, the determination of the shooting leg has to be done besides. The most time-consuming task represents the different transformation steps in each sub-classifier.

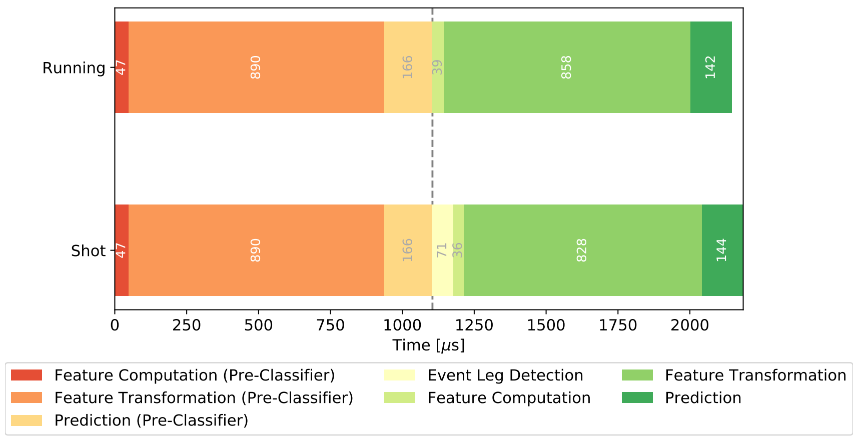


Figure 10.9.: Detailed view: computation time of classification process (averaged values from five repetitions).

The proposed classification scheme uses overlapping sliding windows to segment the data. The used overlap is about 60% and the used window size is 2.56 s. Thus, about every second a new instance is available for which the classification process has to be performed. However, the final prediction of an instance is made under 40 ms. Hence, predictions can be done under one second. Based on these investigations, it can be assumed that the presented classification scheme is real-time capable.

11. Conclusion and Outlook

This thesis aimed to develop a suitable classification scheme based on Machine Learning (ML) and MEMS accelerometer data to detect soccer-specific activities, such as passing, shooting and running. Therefore, different ML algorithms, as well as various feature extraction methods, were analysed and compared with one another. The trained models were tested on real data sets to find out which algorithm suits best for this particular application.

First of all, it has been shown that the Discrete Wavelet Transform (DWT) is a useful and compact method to characterise the temporal and spectral components of a signal. Due to the usage of wavelets, complex signals can be better analysed. The implementation of DWT as a filter bank and, thus, decomposing the signal into approximation and detail coefficients, allows a better understanding of the energy distribution of a signal. The filter bank divides the signal into two or more sub-frequency-bands. Extracting statistical values from the desired sub-frequency-band provides an efficient and compact way to characterise a signal.

For distinguishing passes and shots, investigations of the feature importance indicated that features based on the supporting leg are more useful compared to features based on the shooting leg. Thus, pre- and post-impacts of the kicking movement play an essential role in the classification process.

A classification scheme, which consists of a pre-classifier, a shot/pass-classifier and a standing/walking/running-classifier, has been introduced. The different sub-classifiers have been trained individually. Two different window sizes were investigated: 2.56 s and 1.28 s. The training and testing process of the classification scheme has shown that all classifiers based on the various feature extraction methods can correctly classify standing, walking, running, passing and shooting. At that time, it was not clear yet, which classifier and which feature extraction method is significantly better than another.

More information about the suitability of the various models was provided through tests on real data. The real data contains of simple sequences of soccer-specific activities, such as pass during running. The investigation were carried out with a degree of overlap of approximately 60%. A window size of 2.56 s results in a better solution than a window size of 1.28 s. A time span of 1.28 s is too short to cover a full process of a shot. Pre- and post-impacts were often mistaken for other activities. The topped recall score of the class shot was 80%. Such a recall score is judged as insufficient for this specific applications.

It has been shown that applying the Linear Discriminant Analysis (LDA) to the feature subspace significantly outperforms the other two feature extraction methods in real applications. Logistic Regression, Support Vector Machine (SVM) and Random Forest were able to perfectly detect passing and shooting. That means that the accuracy, precision, recall and ROC AUC scores received 100%. The only misclassifications took place in the transition areas of standing, walking and running and are therefore not considered as critical errors. The best overall performance accomplished the Logistic Regression classifier and the Random Forest classifier with an accuracy score of 94.6%, a precision score of 97.4%, a recall score of 96.1% and a ROC AUC score of 97.2%. The ability to precisely detect the defined activities is given.

The other two feature extraction methods were not able to detect kicks in the sense of detecting a shot including pre- and post-impacts. Another issue was that not all passes were recognised. The performance has deteriorated compared with the variant based on LDA. The worst performance was reached using the Random Forest classifier based on Principal Component Analysis (PCA). The recall-score for shots was only about 54% and for walking only around 40%. A possible explanation is that the Decision Tree classifiers built many linear and orthogonal decision boundaries. Therefore, it is likely that they tend to overfit. That implies that new instances that are slightly different than the training instances are misclassified. Using the PCA for dimensionality reduction does not lead to a gain in precision compared to using the raw feature subset. In general, the Artificial Neural

Network (ANN) showed a mediocre performance. That can be due to several factors, such as the small-sized data set since neural networks are designed to deal with large data sets. Another error source could be the chosen hyperparameters.

To summarise the result of this thesis, with the selection of an appropriate feature subset (feature engineering), it is possible to detect simple, soccer-specific activities using low-cost IMUs. Using the Logistic Regression classifier or the Random Forest classifier in combination with LDA has lead to the most reasonable solutions. It has also been proven, that the presented classification scheme is real-time capable. And since the sensor exceeds the range of ± 16 g during the performance of a shot, it was sufficient to distinguish between passes and shots.

However, this topic provides a favourable outlook for further research. On the one hand, additional movements such as dribbling, tackling or other gestures could be included in the classification process. On the other hand, the gyroscope measurements could be used together with the accelerometer data for the feature computation.

Furthermore, the analysis is only based on data of one test person. Hence, the user dependency should be investigated since different persons show different kick-patterns with different intensities. For example, top-level soccer players exhibit more intensive kicks than hobby soccer players. By the way, the shin guard with embedded, low-cost IMUs could be a possible application for hobby teams. It represents a cost-effective performance analysis (number of passes/shots/sprints) of individual soccer players in comparison to video-based approaches that are present in professional soccer teams. Another interesting point would be the determination of the current running speed of the soccer player to gain more information about the player's performance. Therefore, a regression task could be integrated in the ML process.

Appendices

A. Evaluation Scores

The evaluation scores are based on the analysis of the validation set (Chapter 10.1) using a window size of 128 samples. Scores between 81% and 61% are filled with an orange background. Every score under 61% is marked with a red background. The overall performance of the accuracy score refers to the overall-accuracy. For the precision, recall and ROC AUC score, the overall performance is computed using the macro-average.

Table A.1.: Evaluation scores for variant 1 with window size $W = 128$ samples.

	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	**ST**	94.0	88.5	99.9	94.5
	WA	93.5	94.4	76.2	88.0
	RU	99.9	96.7	99.5	92.2
	PA	99.5	90.3	84.8	92.3
	SH	99.8	100.0	80.0	90.0
Overall-Performance [%]:		93.3	95.5	88.1	92.8
SVM	**ST**	94.6	89.5	99.7	95.0
	WA	93.2	97.4	76.7	88.0
	RU	98.1	94.4	98.3	98.2
	PA	95.5	90.0	81.8	90.8
	SH	99.7	100.0	66.7	83.3
Overall Performance [%]:		92.5	94.3	84.6	91.1
Random Forest	**ST**	97.2	94.2	99.9	97.4
	WA	96.2	99.0	86.8	93.2
	RU	98.7	95.6	99.3	98.9
	PA	99.2	82.8	72.7	86.2
	SH	99.9	100.0	86.7	93.3
Overall Performance [%]:		95.5	94.3	89.1	93.8
ANN	**ST**	95.1	90.5	99.7	95.5
	WA	94.3	98.9	79.8	89.7
	RU	98.7	95.6	99.3	98.9
	PA	99.6	96.4	81.8	90.9
	SH	99.8	92.3	80.0	90.0
Overall Performance [%]:		93.7	94.7	88.1	93.0

Table A.2.: Evaluation scores for variant 2 with window size $W = 128$ samples.

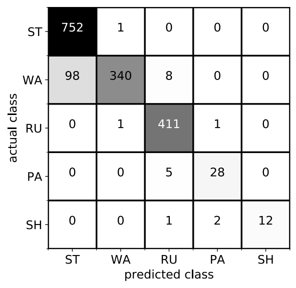
	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	**ST**	95.1	90.5	99.7	94.3
	WA	94.3	98.9	79.8	87.5
	RU	98.7	95.6	99.3	99.0
	PA	99.6	96.4	81.8	95.4
	SH	99.8	92.3	80.0	90.0
Overall Performance [%]:		93.7	94.7	88.1	93.0
SVM	**ST**	88.7	80.2	99.7	89.7
	WA	87.7	97.3	55.8	77.6
	RU	98.5	95.5	98.5	98.5
	PA	99.3	83.9	78.8	89.2
	SH	98.8	100.0	73.3	86.7
Overall Performance [%]:		87.0	91.4	81.2	88.3
Random Forest	**ST**	97.0	94.1	99.7	97.3
	WA	85.4	98.6	46.2	73.0
	RU	86.9	65.9	98.1	90.6
	PA	99.2	80.6	75.8	87.7
	SH	99.4	85.7	40.0	70.0
Overall Performance [%]:		83.9	85.0	71.9	83.7
ANN	**ST**	95.1	90.5	99.7	95.5
	WA	94.3	98.9	79.8	89.7
	RU	98.7	95.6	99.3	98.9
	PA	99.6	96.4	81.8	90.9
	SH	99.8	92.3	80.0	90.0
Overall Performance [%]:		93.7	94.7	88.1	93.0

Table A.3.: Evaluation scores for variant 3 with window size $W = 128$ samples.

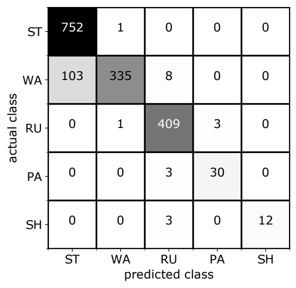
	Class	Accuracy [%]	Precision [%]	Recall [%]	ROC AUC [%]
Logistic Regression	ST	98.2	96.8	99.3	98.3
	WA	96.4	98.0	88.6	94.0
	RU	98.0	93.8	98.5	94.0
	PA	99.3	75.6	93.9	98.2
	SH	99.6	100.0	60.0	96.7
Overall Performance [%]:		95.8	92.8	88.1	93.4
SVM	ST	95.2	90.7	99.7	95.6
	WA	92.3	97.9	72.9	86.1
	RU	96.8	89.6	98.5	97.4
	PA	99.4	81.1	90.9	95.2
	SH	99.6	100.0	60.0	80.0
Overall Performance [%]:		91.7	91.9	84.4	90.9
Random Forest	ST	97.8	95.9	99.5	98.0
	WA	96.0	98.7	86.3	93.0
	RU	97.8	92.5	99.0	98.2
	PA	99.3	78.9	90.9	95.2
	SH	99.6	100.0	60.0	80.0
Overall Performance [%]:		95.3	96.0	97.8	92.9
ANN	ST	98.4	97.3	99.3	98.5
	WA	96.9	98.8	89.7	94.6
	RU	98.0	93.8	98.5	98.2
	PA	99.2	75.0	90.9	95.1
	SH	99.6	83.3	66.7	83.3
Overall Performance [%]:		96.1	89.6	89.0	94.0

B. Confusion Matrices

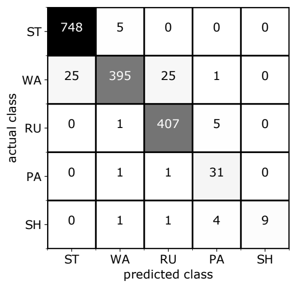
The confusion matrices are a result of the evaluation of the validation set (Chapter 10.1) using a window size of 128 samples ($\widehat{=}$1.28 s). In total, 12 different models are evaluated. The rows compare the feature extraction methods using the same ML algorithm. The columns compare the different ML algorithms based on the same feature extraction method.



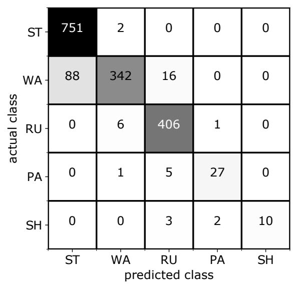
(a) Logistic Regression: variant 1



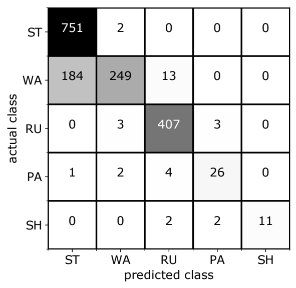
(b) Logistic Regression: variant 2



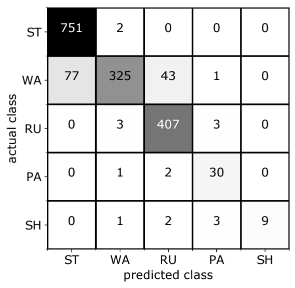
(c) Logistic Regression: variant 3



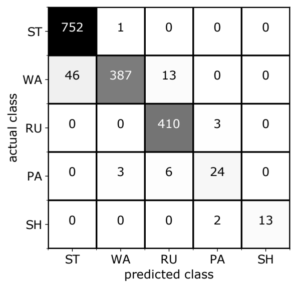
(d) SVM: variant 1



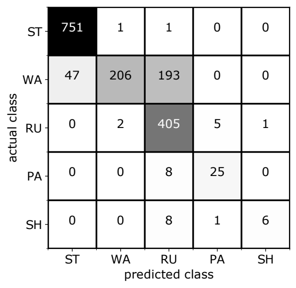
(e) SVM: variant 2



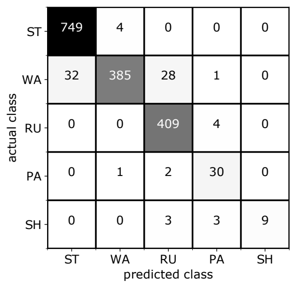
(f) SVM: variant 3



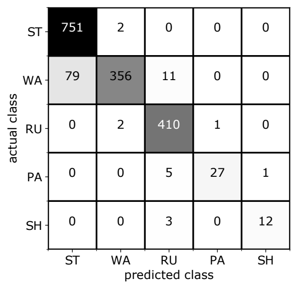
(g) Random Forest: variant 1



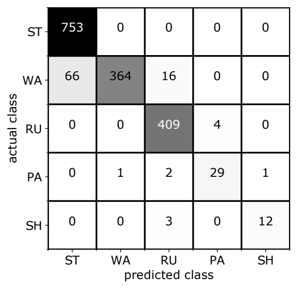
(h) Random Forest: variant 2



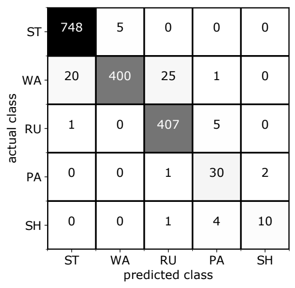
(i) Random Forest: variant 3



(j) ANN: variant 1



(k) ANN: variant 2



(l) ANN: variant 3

Figure B.1.: Confusion matrices for validation set with a window size of 128 samples.

Bibliography

[1] A. Paar. (2017). "Anton paar sportstec: Eröffnung des skills.lab." http://www.skills-lab.com/, [Online]. Available: https://www.anton-paar.com/at-de/ueber-uns/news/news/detail/anton-paar-sportstec-eroeffnung-des-skillslab/ (visited on 08/03/2020).

[2] F. Tenzer. (16.06.2020). "Prognose zum absatz von wearables weltweit von 2014 bis 2024," [Online]. Available: https://de.statista.com/statistik/daten/studie/417580/umfrage/prognose-zum-absatz-von-wearables/ (visited on 08/03/2020).

[3] D. Schuldhaus, C. Zwick, Körger Harald, Dorschky Eva, R. Kirk, Eskofier, and Bjoern M., "Inertial sensor-based approach for shot/pass classification during a soccer match," Digital Sports Group, Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg, 4 pp.

[4] B. Hofmann-Wellenhof, K. Legat, and M. Wieser, *Navigation, Principles of Positioning and Guidance*, eng. Vienna and s.l.: Springer Vienna, 2003, 427 pp., Hofmann-Wellenhof, Bernhard (author.) Legat, Klaus (author.) Wieser, Manfred (author.), ISBN: 978-3-7091-6078-7. DOI: 10.1007/978-3-7091-6078-7.

[5] D. Uttamchandani, *Wireless MEMS Networks and Applications*, eng, ser. Woodhead Publishing Series in Electronic and Optical Materials. Kent: Elsevier Science, 2016, 290 pp., ISBN: 978-0-08-100450-0. [Online]. Available: http://www.sciencedirect.com/science/book/9780081004494.

[6] PCB Group. (2020). "Introduction to mems accelerometers," [Online]. Available: https://www.pcb.com/resources/technical-information/mems-accelerometers (visited on 07/03/2020).

[7] T. Moder, C. Reitbauer, M. Dorn, and M. Wieser, *Calibration of Smartphone Sensor Data Usable for Pedestrian Dead Reckoning*, eng. Piscataway, NJ: IEEE, 2017, 8 pp., Makino, Hideo (VeranstalterIn) Ureña, Jesús (VeranstalterIn), ISBN: 9781509063000. [Online]. Available: `http://ieeexplore.ieee.org/servlet/opac?punumber=8106926`.

[8] A. Noureldin, T. B. Karamat, and J. Georgy, *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration*, eng. Berlin and Heidelberg: Springer, 2013, 313 pp., ISBN: 978-3-642-30466-8. DOI: `10.1007/978-3-642-30466-8`. [Online]. Available: `http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=528445`.

[9] G. Krishnan, C. Kshirsagar, G. Ananthasuresh, and N. Bhat, "Micromachined high-resolution accelerometers," *Journal of the Indian Institute of Science*, vol. 87, 2007.

[10] I. Amerini, P. Bestagini, L. Bondi, R. Caldelli, M. Casini, and S. Tubaro, "Robust smartphone fingerprint by mixing device sensors features for mobile strong authentication," *Electronic Imaging*, vol. 2016, no. 8, pp. 1–8, 2016, ISSN: 2470-1173. DOI: `10.2352/ISSN.2470-1173.2016.8.MWSF-088`.

[11] Henry Martin, Paul Groves, Mark Newman, "The limits of in-run calibration of mems inertial sensors and sensor arrays," vol. 2016,

[12] T. Moder, "Wesen und nutzen inertialer mems sensoren in der fahrzeugnavigation," Master's Thesis, TU Graz, 2011, 120 pp.

[13] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global positioning systems, inertial navigation and integration*, eng, [Elektronische Ressource], 2. ed. Hoboken, NJ: Wiley, 2007, 525 pp., ISBN: 0-470-04190-0. DOI: `10.1002/0470099720`. [Online]. Available: `http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10278857`.

[14] S. Stančin and S. Tomažič, "Time- and computation-efficient calibration of mems 3d accelerometers and gyroscopes," eng, *Sensors (Basel, Switzerland)*, vol. 14, no. 8, pp. 14 885–14 915, 2014, Journal Article Research Support, Non-U.S. Gov't. DOI: `10.3390/s140814885`. eprint: `25123469`.

[15] Bolder Flight Systems. (2007). "Mpu9250, Readme.md," [Online]. Available: https://github.com/bolderflight/MPU9250 (visited on 07/01/2020).

[16] P. S. Addison, *The illustrated wavelet transform handbook, Introductory theory and applications in science, engineering, medicine and finance*, Second edition. Boca Raton FL: CRC Press Taylor & Francis Group, 2017, xvii, 446 pages, ISBN: 1482251329.

[17] Lee A. Barford, R. Shane Fazzio, and David R. Smith, *An introduction to wavelets*, Hewlett-Packard Company 1992, Ed., Instruments and Photonics Laboratory, 1992.

[18] C.-L. Liu, *A Tutorial of the Wavelet Transform*. 2010, 72 pp.

[19] F. S. Ayachi, H. P. Nguyen, C. Lavigne-Pelletier, E. Goubault, P. Boissy, and C. Duval, "Wavelet-based algorithm for auto-detection of daily living activities of older adults captured by multiple inertial measurement units (imus)," eng, *Physiological measurement*, vol. 37, no. 3, pp. 442–461, 2016, Journal Article. DOI: 10.1088/0967-3334/37/3/442. eprint: 26914432.

[20] P. Barralon, N. Vuillerme, and N. Noury, "Walk detection with a kinematic sensor: Frequency and wavelet comparisonenvironment," eng, *Annals of biomedical engineering*, vol. 34, no. 4, pp. 547–563, 2006, Journal Article Research Support, Non-U.S. Gov't Review, ISSN: 0090-6964. DOI: 10.1007/s10439-005-9068-2. eprint: 16550450.

[21] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle, *GNSS - Global Navigation Satellite Systems, GPS, GLONASS, Galileo, and more*, eng. Vienna: Springer-Verlag Wien, 2008, ISBN: 978-3-211-73012-6. DOI: 10.1007/978-3-211-73017-1. [Online]. Available: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10490534.

[22] G. Tzanetakis, G. Essl, and P. Cook, "Audio analysis using the discrete wavelet transform," *Proceedings of the Conference in Acoustics and Music Theory Applications*, pp. 318–323, 2001.

[23] Y. Zhang, Z. Guo, W. Wang, S. He, T. Lee, and M. Loew, "A comparison of the wavelet and short-time fourier transforms for doppler spectral analysis," *Medical Engineering & Physics*, vol. 25, no. 7, pp. 547–557,

2003, PII: S1350453303000523, ISSN: 13504533. DOI: 10.1016/S1350-4533(03)00052-3.

[24] S. Vandeput, "Heart rate variability : Linear and nonlinear analysis with applications in human physiology," Faculty of Electrical Engineering, ISBN 978-94-6018-262-4, Katholieke Universiteit Leuven, 2020.

[25] M. Chaudhary and A. Dhamija, "A brief study of various wavelet families and compression techniques," *Journal of Global Research in Computer Science*, 2013, ISSN-2229-371X. [Online]. Available: www.jgrcs.info.

[26] D. Komorowski and S. Pietraszek, "The use of continuous wavelet transform based on the fast fourier transform in the analysis of multichannel electrogastrography recordings," eng, *Journal of medical systems*, vol. 40, no. 1, p. 10, 2016, Journal Article. DOI: 10.1007/s10916-015-0358-4. eprint: 26573647.

[27] D. T. Lee and A. Yamamoto, "Wavelet analysis, Theory and applications," *Hewlett-Packard Journal*, vol. 1994, [Online]. Available: https://www.hpl.hp.com/hpjournal/94dec/dec94a6a.pdf.

[28] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989, ISSN: 01628828. DOI: 10.1109/34.192463.

[29] A. Jensen and A. Cour-Harbo, *Ripples in Mathematics, The Discrete Wavelet Transform*, eng. Berlin and Heidelberg: Springer, 2001, 246 pp., ISBN: 978-3-642-56702-5. DOI: 10.1007/978-3-642-56702-5.

[30] L. Wanhammar and Y. J. Yu, "Digital filter structures and their implementation," in *Academic Press Library in Signal Processing: Volume 1 - Signal Processing Theory and Machine Learning*, ser. Academic Press Library in Signal Processing, vol. 1, Elsevier, 2014, pp. 245–338, ISBN: 9780123965028. DOI: 10.1016/B978-0-12-396502-8.00006-1.

[31] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow, Concepts, tools, and techniques to build intelligent systems*, Second edition. 2019, 819 pp., ISBN: 978-1-492-03264-9.

[32] P. Bhavsar, I. Safro, N. Bouaynaya, R. Polikar, and D. Dera, "Machine learning in transportation data analytics," in *Data Analytics for Intelligent Transportation Systems*, Elsevier, 2017, pp. 283–307, ISBN: 9780128097151. DOI: 10.1016/B978-0-12-809715-1.00012-2.

[33] M. H. C. Law, M. A. T. Figueiredo, and A. K. Jain, "Simultaneous feature selection and clustering using mixture models," eng, *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1154–1166, 2004, Evaluation Study Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, Non-P.H.S., ISSN: 0162-8828. DOI: 10.1109/TPAMI.2004.71. eprint: 15742891.

[34] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014, PII: S0045790613003066, ISSN: 00457906. DOI: 10.1016/j.compeleceng.2013.11.024.

[35] S. Raschka. (2014). "About feature scaling and normalization, And the effect of standardization for machine learning algorithms," [Online]. Available: https://sebastianraschka.com/Articles/2014_about_feature_scaling.html.

[36] A. Zheng. (2020). "Evaluating machine learning models by alice zheng, Chapter 4. hyperparameter tuning." O'Reilly online learning, Ed., [Online]. Available: https://www.oreilly.com/library/view/evaluating-machine-learning/9781492048756/ch04.html (visited on 05/15/2020).

[37] C. Piech, *Logistic regression*, Stanford University, Ed., Lecture Handouts, 2017. [Online]. Available: https://web.stanford.edu/class/archive/cs/cs109/cs109.1176/lectureHandouts/ (visited on 06/26/2020).

[38] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006, xx, 738, ISBN: 9780387310732.

[39] J. G. Scott, *Statistical modeling*, University of Texas at Austin, 2010. [Online]. Available: http://jgscott.github.com/.

[40] C. Shalizi, *Chapter 12, Logistic regression*, Course Title: STATISTICS 154, University of California, 2013.

[41] S. Ruder, *An overview of gradient descent optimization algorithms*, Insight Centre for Data Analytics, NUI Galway, 15.09.2016. [Online]. Available: `http://arxiv.org/pdf/1609.04747v2`.

[42] J. C. Stoltzfus, "Logistic regression: A brief primer," eng, *Academic emergency medicine : official journal of the Society for Academic Emergency Medicine*, vol. 18, no. 10, pp. 1099–1104, 2011, Journal Article. DOI: `10.1111/j.1553-2712.2011.01185.x`. eprint: `21996075`.

[43] K. S. Parikh and T. P. Shah, "Support vector machine – a large margin classifier to diagnose skin illnesses," *Procedia Technology*, vol. 23, pp. 369–375, 2016, PII: S2212017316300408, ISSN: 22120173. DOI: `10.1016/j.protcy.2016.03.039`.

[44] W. S. Noble, "What is a support vector machine?" eng, *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006, Journal Article Research Support, U.S. Gov't, Non-P.H.S. Review, ISSN: 1087-0156. DOI: `10.1038/nbt1206-1565`. eprint: `17160063`.

[45] J. A. S. Sá, A. C. Almeida, B. R. P. Rocha, M. A. S. Mota, J. R. S. Souza, and L. M. Dentel, "Lightning forecast using data mining techniques on hourly evolution of the convective available potential energy," in *Anais do 10. Congresso Brasileiro de Inteligência Computacional*, (Fortaleza, Ceará,), G. d. A. Barreto and J. A. F. Costa, Eds., SBIC, 8-11/11/2011, pp. 1–5. DOI: `10.21528/CBIC2011-27.1`.

[46] D. Poojari. (2019). "Machine learning basics: Decision tree from scratch, Theoretical framework." Towards Data Science, Ed., [Online]. Available: `https://towardsdatascience.com/machine-learning-basics-descision-tree-from-scratch-part-i-4251bfa1b45c` (visited on 05/11/2020).

[47] W. Koehrsen. (2018). "An implementation and explanation of the random forest in python, A guide for using and understanding the random forest by building up from a single decision tree." Towards Data Science, Ed., [Online]. Available: `https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76` (visited on 05/11/2020).

[48] S. Löwel and W. Singer, "Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity," eng, *Science (New York, N.Y.)*, vol. 255, no. 5041, pp. 209–212, 1992, Journal Article, ISSN: 0036-8075. DOI: 10.1126/science.1372754. eprint: 1372754.

[49] D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS, "Learning internal representations by error propagation," in *Readings in Cognitive Science*, Elsevier, 1988, pp. 399–421, ISBN: 9781483214467. DOI: 10.1016/B978-1-4832-1446-7.50035-2.

[50] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," vol. 2017, Advances in Neural Information Processing Systems 30 (NIPS 2017) 9 pages (+ 93 pages appendix). [Online]. Available: http://arxiv.org/pdf/1706.02515v5.

[51] J. S. Sánchez, V. García, and R. A. Mollineda, "Exploring synergetic effects of dimensionality reduction and resampling tools on hyperspectral imagery data classification," in *Machine Learning and Data Mining in Pattern Recognition*, ser. Lecture Notes in Computer Science, P. Perner, Ed., vol. 6871, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 511–523, ISBN: 978-3-642-23198-8. DOI: 10.1007/978-3-642-23199-5_38.

[52] F. Krüger, "Activity, context, and plan recognition with computational causal behaviour models," e Faculty of Computer Science and Electrical Engineering, Dissertation, University of Rostock, 2016.

[53] S. Abraham, C. Huynh, and H. Vu, "Classification of soils into hydrologic groups using machine learning," *Data*, vol. 5, no. 1, p. 2, 2020, PII: data5010002. DOI: 10.3390/data5010002.

[54] G. Lanaro. (2016). "Evaluation measures for multiclass problems," [Online]. Available: http://gabrielelanaro.github.io/blog/2016/02/03/multiclass-evaluation-measures.html.

[55] K. Wisiol, "Human-activity recognition," Institute of Navigation, Master's Thesis, TU Graz, 2014, 98 pp.

[56] M. Pechenizkiy, A. Tsymbal, and S. Puuronen, "On combining principal components with fisher's linear discriminants for supervised learning," 2006.

[57] L. Fei-Fei, *Principal component analysis (pca)*, Princeton University, Ed., COS 429 - Computer Vision, 2008. [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall08/cos429/CourseMaterials/lecture2/PCA_handout.pdf.

[58] A. M. Martinez and A. C. Kak, "Pca versus lda," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 2, pp. 228–233, 2001, ISSN: 01628828. DOI: 10.1109/34.908974.

[59] S. Raschka, *Linear discriminant analysis, Bit by bit*, 2014. [Online]. Available: https://sebastianraschka.com/ (visited on 06/28/2020).

[60] J. LEVANON and J. DAPENA, "Comparison of the kinematics of the full-instep and pass kicks in soccer," *Medicine & Science in Sports & Exercise*, vol. 30, no. 6, 1998, ISSN: 0195-9131. [Online]. Available: https://journals.lww.com/acsm-msse/Fulltext/1998/06000/Comparison_of_the_kinematics_of_the_full_instep.22.aspx.

[61] InvenSense Inc., Ed., *Mpu-9250, Product specification revision 1.1*.

[62] ESPRESSIF SYSTEMS. (2019). "Esp32," [Online]. Available: https://www.espressif.com/en/products/socs/esp32/overview (visited on 07/07/2020).

[63] EndRun Technologies, "Ptp-1588 [white paper]," n.d. [Online]. Available: https://endruntechnologies.com/pdf/PTP-1588.pdf (visited on 07/08/2020).

[64] The PyWavelets Developers. (18.04.2020). "Discrete wavelet transform (dwt)," [Online]. Available: https://pywavelets.readthedocs.io/en/latest/ref/dwt-discrete-wavelet-transform.html (visited on 07/16/2020).

[65] J.-Y. Yang, J.-S. Wang, and Y.-P. Chen, "Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers," *Pattern Recognition Letters*, vol. 29, no. 16, pp. 2213–2220, 2008, PII: S0167865508002560, ISSN: 01678655. DOI: 10.1016/j.patrec.2008.08.002.

[66] Keras. (20.07.2020). "Sgd," [Online]. Available: https://keras.io/api/optimizers/sgd/.