Konzeption und prototypische Umsetzung einer Trainings-App für Kickboxer

unter Verwendung von inertialen Messeinheiten, maschinellem Lernen und dem Flutter-Framework

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Tobias Saladauski

[redacted]

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr. Torsten Edeler

Zweitprüfer: Prof. Dr. Jan Mietzner

Hamburg, 26. 4. 2022

Inhaltsverzeichnis

Keywords

Human Activity Recognition, Gesture Recognititon, Inertial Measurement Unit, Movesense, Python, Jupyter, Machine Learning, Supervised Learning, Multiclass Classification, MINIROCKET, Dart, Flutter, C

Abstract

People like to compete in Sports, including martial arts, which more and more amateur athletes discover for themselves, some of whom might one day go from amateur to pro. The lack of technical sports equipment for kickboxers and muay thai athletes is a problem for athletes that would like to have the ability to somehow quantize their training progress. Out of personal engagement with the sport as well as an interest in the field of machine learning does this thesis attempt to investigate what's possible when hardware, software and humans work together. This thesis covers the basics of movement sensing and machine learning, especially classification, formulates a concept, and creates and evaluates a prototype of a kickbox-app.

Stichworte

Gesten-Erkennung, Inertiale Messeinheit, Movesense, Python, Jupyter, Maschinelles Lernen, Überwachtes Lernen, Multiklassen-Klassifikation, MINIROCKET, Dart, Flutter, C

Zusammenfassung

Menschen machen gerne Sport, auch dem Kampfsport widmen sich jedes Jahr mehr Amateursportler, von denen manche eventuell eines Tages vom Amateur zum Profi werden. Der Mangel an technischem Trainingsequipment für Kick- und Thaiboxer stellt Sportler die ihre Trainingsfortschritte gerne quantifizieren vor ein Problem. Aus persönlichem Engagement im Sport sowie Interesse an dem Feld des Maschinellen Lernens heraus versucht diese Arbeit zu erforschen was möglich ist wenn Hardware, Software und Mensch zusammenarbeiten. Diese Arbeit geht auf die Grundlagen der Bewegungsmessung sowie des Maschinellen Lernens insbesondere Klassifikation ein, verfasst ein Konzept, und kreiert und evaluiert einen Prototyp einer Kickbox-App.

1 Einleitung

1.1 Motivation

Die steigende Anzahl an sportlich aktiven Menschen in Deutschland beschreibt einen Trend der automatisch auch von einem Anstieg in Herstellung und Nutzung diversen Sportequipments begleitet wird, vor allem sei hier auf die stetige Weiterentwicklung der Integrierung von Sportsensorik in den Trainingsprozess hinzuweisen. Von Freizeit- bis Profisport können Athleten von der Quantisierung ihres Trainings profitieren, nicht zuletzt um Zeitpunkt und Ausmaß der sportlichen Aktivität unkompliziert abzuspeichern. Während Leistungssportler die Analyse der geleisteten Trainingseinheiten nutzen können um die eigene Leistung zu optimieren ermöglichet es Freizeitsportlern die erzielten Trainingsfortschritte zu veröffentlichen und mit anderen zu vergleichen.
Im Jahr 2021 waren in Deutschland 25.807 Menschen Mitglied in insgesamt 342 Kickbox-Sportvereinen (DOSB 2021), hinzu kommen eine undefinierte Anzahl an Sportlern welche Kickboxen außerhalb eingetragener Vereine, z.B. in Fitness- und Kampfsportstudios oder privat, betreiben. Doch während auf den Boxsport zugeschnittene Trainingssensoren erhältlich sind gibt es ein solches Produkt für den Kickbox-Sport nicht. Aus diesem offensichtlichen Mangel sowie persönlichem Interesse an der Materie[1] ergab sich folglich die Entscheidung ein entsprechendes Produkt zu konzipieren und einen Prototypen zu entwickeln.

1.2 Struktur der Arbeit

Inhalt dieser Arbeit ist die Konzeption und prototypische Entwicklung einer Android-App welche, in Verbindung mit 4 Bewegungssensoren sowie einem mit machinellem Lernen trainierten Algorithmus, von Kickboxern ausgeführte Tritt- und Schlagtechniken beim Schattenboxen[2] erkennen soll.
Zunächst werden in Kapitel 2 einige theoretische Grundlagen der Bewegungsmessung sowie des Maschinellen Lernens erklärt welche für die Entwicklung des Algorithmus sowie der App von Belang sind.
Dann werden in Kapitel 3 die beiden Algorithmen vorgestellt welche in dieser Arbeit verwendet wurden.

[1] Der Autor dieser Arbeit trainiert seit 2018 selbst Muay Thai (*Thaiboxen*) bzw. K1-Kickboxen

[2] *Schattenboxen* bezeichnet die Ausführung von Techniken ohne einen Trainingspartner oder Equipment zu treffen

In Kapitel 4 wird ein grober Entwurf der gewünschten Funktionalität vorgestellt und das Thema der Trainingsdaten und deren Validierung behandelt.
In Kapitel 5 werden das verwendete Equipment, Tools und Programmiersprachen vorgestellt.
In Kapitel 6 wird dann der Entwicklungs-/Trainingsprozess dargelegt.
Im folgenden Kapitel 7 werden der implementierte Algorithmus evaluiert.
Schlussendlich wird in Kapitel 8 ein Fazit gezogen und ein Ausblick auf mögliche Weiterentwicklungen gewagt.

2 Theoretische Grundlagen

2.1 Inertialsensoren und inertiale Messeinheiten

2.1.1 Definition und Funktionsweise

Der Begriff „Inertialsensor“ bezeichnet Sensoren (typischerweise Beschleunigungs- oder Drehratensensoren) welche zur Messung der eigenen Bewegung im Raum (und somit der Bewegung des Körpers/Gegenstands an welchem diese befestigt sind) das sog. Trägheitsprinzip verwenden: das Bestreben von physikalischen Körpern in ihrem Bewegungszustand zu verharren solange keine externen Kräfte auf sie einwirken. Demnach kann die Bewegung anhand der Trägheit (lateinisch *inertia*) einer beweglichen Masse innerhalb des Sensors gemessen werden wenn dieser durch externe Kräfte bewegt/rotiert wird. Dank technologischer Fortschritte in der Mikrosystemtechnik ist es heute möglich diese Sensoren in sehr kleinen Gehäusen unterzubringen.
Die räumliche Kombination von drei Beschleunigungs- und drei Drehratensensoren, jeweils orthogonal zueinander motiert, in einem Gehäuse wird allgemein als „inertiale Messeinheit“ (IMU, englisch *Inertial Measurement Unit*, umgangssprachlich „Bewegungssensor“) bezeichnet. Durch die Kombination der Messwerte der verbauten Sensoren kann ein IMU Bewegungen entlang aller sechs kinematischen Freiheitsgrade[1] messen.
Sensor Fusion
In manchen IMUs werden zusätzlich drei orthogonale Magnetometer (Magnetfeldsensoren) eingebaut welche die Messung von Magnetfeldern erlauben, dies kann als Kompass zur Kalibrierung/Referenz dienen und Drift (siehe Abschnitt 2.1.3) entgegenwirken.
Im Handel erhältliche IMUs sind oft klein, leicht und kabellos, wodurch sie sich hervorragend für Anwendungen eignen in welchen die natürliche Bewegung eines physikalischen Körpers untersucht aber möglichst wenig gestört werden soll (z.B. Menschliche oder Tierische Bewegungsanalyse). Solche IMUs verfügen um die gemessenen Daten zu speichern typischerweise entweder über einen Transmitter (z.B. Bluetooth oder Funk) oder einen SD-Karten-Slot (Iosa et al. 2016).

[1] Die sechs kinematischen Freiheitsgrade beschreiben die voneinander unabhängigen Bewegungen welche ein starrer Körpers im dreidimensionalen Raum ausführen kann: die Bewegung entlang der drei Achsen sowie die Rotation um diese

2.1.2 Anwendungsbereiche

IMUs werden heutzutage in einer Vielzahl von industriellen Bereichen sowie Konsumgütern eingesetzt:

- in inertialen Navigationssystemen („Trägheitsnavigationssysteme“) von Fahr- und Flugzeugen sowie Schiffen werden IMUs verwendet um basierend auf der Beschleunigungs- und Drehrate kontinuierlich die aktuelle Position zu bestimmen. Um den Messfehler, welcher durch Drift (siehe Abschnitt 2.1.3) entsteht, zu minimieren bzw. auszugleichen kann ein solches System mit externen Referenzsignalen (z.B. GPS) zusammenarbeiten , z.B. kann ein Navigationssystem einen temporären Ausfall eines GPS-Systems (z.B. in einem Tunnel oder auf hoher See) ausgleichen indem es basierend auf der letzten bekannten Position und Beschleunigung sowie den seitdem registrierten Inertialmessungen den aktuellen Standort schätzt.
- fast alle Smartphones und Tablets nutzen IMUs um den Bildschirm darauf anzupassen ob das Gerät horizontal oder vertikal gehalten wird.
- Im Gaming erlaubt die Verwendung von IMUs dem Nutzer das beeinflussen eines Videospiels durch Bewegung des Controllers, z.B. in der Nintendo Wii/Switch.
- Fitnesstracker, Smartwatches und andere sog. *Wearables* verwenden IMUs um bestimmte Bewegungsmuster (z.b. laufen) zu erkennen oder Schritte zu zählen.
- in der Medizin und Physiotherapie existieren mehrere Anwendungen für Menschliche Bewegungsanalyse durch IMUs, z.B. Ganganalyse, Stabilometrie oder Tremor-Gutachten. (Iosa et al. 2016)
- Lenkflugkörper, unbemannte Luftfahrzeuge sowie im Handel erhältliche Multikopter (umgangsprachlich *Drohnen*) stabilisieren sich im Flug mithilfe von eingebauten IMUs.
- IMUs können für Motion Capture Verfahren verwendet werden und stellen damit eine Alternative zur verbreiteten visuellen Methode dar (Xsens o.J.).

2.1.3 Nachteile und Schwächen

Ein großer Nachteil von inertialen Messeinheiten in der Navigation ist dass sie mit der Zeit immer ungenauer werden, d.h. die geschätzte Position und Orientierung entfernt sich mit der Zeit immer weiter von der tatsächlichen. Dieses Phänomen, welches als *Drift* bezeichnet wird, hat zwei Ursachen:

1. Die Akkumulation winziger Messfehler der Sensoren selbst.
2. Die ein- bzw. zweifache mathematische Integration der Messwerte, welche für die Umwandlung von gemessenen Beschleunigungswerten in Positionsschätzungen notwendig ist, dadurch jedoch den akkumulierten Messfehler potenziert (Dudek & Jenkin 2008).

2.2 Machinelles Lernen

2.2.1 Definition

Machinelles Lernen (ML, englisch *Machine Learning*) steht als Oberbegriff für den Prozess in welchem sich künstliche Systeme, typischerweise Computer-Algorithmen, automatisch durch Erfahrung und die Verarbeitung von Daten verbessern können als auch für das Feld welches sich mit solchen Algorithmen beschäftigt. Es ist kategorisch dem Forschungsgebiet der künstlichen Intelligenz zuzuordnen. ML-Algorithmen nutzen Beispieldaten (sog. Trainingsdaten) um ein internes statistisches Modell aufzubauen mithilfe dessen sie Entscheidungen treffen, ohne für ein beliebiges Problem spezifisch „vorprogrammiert“ worden zu sein. ML-Algorithmen funktionieren auf Grundlage der Annahme dass Vorgehensweisen und Schlussfolgerungen, welche in der Vergangenheit zum korrekten Ergebnis geführt haben auch zukünftig zum richtigen Ergebnis führen. ML-Algorithmen können Probleme lösen, für die sie nicht spezifisch programmiert wurden, indem sie von vorhandenen Daten lernen. Für simple Aufgaben ist es möglich einem Algorithmus die notwendigen Schritte zur Lösung exakt vorzuschreiben, sodass kein maschinelles Lernen erforderlich ist. Für komplexere/kompliziertere Probleme ist dies aber oft schwer realisierbar oder sogar unmöglich weshalb es effektiver und oft effizienter ist ein Programm seinen eigenen Algorithmus entwickeln zu lassen. (Alpaydin 2020: 1-4)

2.2.2 Anwendung

ML-Algorithmen finden in einer Vielzahl von Anwendungen Verwendung in welchen es schwer bis unmöglich ist Probleme durch traditionelle Algorithmen zu lösen:

- Medizin
- Spam-Filter
- Spracherkennung
- Maschinelles Sehen (englisch *Machine Vision*), auch Computer Vision genannt
- Das sog. Data-Mining (englisch *Data* „Daten“ und *mine* „abbauen“, „fördern“) verwendet unüberwachtes Lernen (siehe Abschnitt 2.2.3) um in großen Datensätzen („Big Data“) nach Mustern zu suchen.

2.2.3 Algorithmen

Abhängig von der Art des Feedbacks welches einem ML-Algorithmus in seiner Lernphase zur Verfügung steht werden die Herangehensweisen an ML-Probleme traditionell in drei Kategorien aufgeteilt:

- Unüberwachtes Lernen (englisch *Unsupervised Learning*, UL):
  dem Algorithmus werden keine Informationen über die erwünschten Ausgabewerte übergeben, sodass dieser die Trainingsdaten analysieren muss ohne auf ein spezifiziertes Ergebnis hinzuarbeiten. Hierbei ist das Ziel oft das Finden von latenten Strukturen in den Trainingsdaten, wie z.B. Cluster (Datenpunkte welche in einer oder mehreren Eigenschaften Ähnlichkeiten aufweisen). Anstelle dass UL-Algorithmen auf Feedback reagieren (wie SL-Algorithmen es tun) finden sie Ähnlichkeiten in den Trainingsdaten und reagieren auf einen neuen Datensatz basierend auf der Gegenwart oder dem Fehlen dieser Ähnlichkeiten. Bei der Clusteranalyse werden eine Menge von Datenpunkten in Teilmengen, sogenannte Cluster, aufgeteilt, sodass Datenpunkte desselben Clusters in einer oder mehreren Kriterien ähnlich sind und Datenpunkte unterschiedlicher Cluster sich unähnlich sind. Verschiedene Cluster-Techniken unterscheiden sich darin dass sie für die Sortierung unterschiedliche Ähnlichkeitsmetriken verwenden. Evaluiert werden diese z.B. anhand der Ähnlichkeit der Datenpunkte eines Clusters oder der Unähnlichkeit unterschiedlicher Cluster.

- Bestärkendes Lernen (englisch *Reinforcement Learning*, RL):
  der Algorithmus muss mit einem dynamischen System interagieren in welchem er ein bestimmtes Ziel erreichen muss, z.B. ein Fahrzeug steuern oder ein Videospiel spielen. Während diesem Prozess erhält der Algorithmus vom System Feedback (z.B. mehr oder weniger „Belohnungen“), und versucht anhand dieses Feedbacks die eigene Leistung zu optimieren. RL-Algorithmen werden z.B. in selbstfahrenden Autos verwendet, oder wenn ein Computer lernen soll ein Spiel gegen einen menschlichen Gegner zu spielen.

- Überwachtes Lernen (englisch *Supervised Learning*, SL):
  wird im folgenden Abschnitt erläutert.

Supervised Learning

Überwachtes Lernen, oder der gebräuchlichere englische Begriff Supervised Learning (SL), bezeichnet die für Klassifizierungsprobleme (wie das dieser App zugrundeliegende) am weitesten verbreitete Herangehensweise. Einem SL-Algorithmus werden Trainingsdaten, beispielhafte Eingabewerte sowie die erwünschten „richtigen“ Ausgabewerte, zur Verfügung gestellt, mit denen eine Systematik erlernt werden soll um Eingaben korrekt auf Ausgaben abzubilden. SL-Algorithmen erstellen ein mathematisches Modell aus den Trainingsdaten welches sowohl die Eingaben als auch die erwünschten Ausgaben enthält (Russell & Norvig 2010: 706-710). In dem mathematischen Modell besteht jede Eingabe aus einer Liste (auch Array oder Vektor genannt) von Werten, dies wird auch als Feature Vector (englisch *Feature*: Eigenschaft) bezeichnet, sodass die gesamte Menge an Trainingsdaten in Form einer Matrix repräsentiert werden kann. Durch iterative Optimierung einer sogenannten Zielfunktion (englisch *Objective Function*) können SL-Algorithmen eine Funktion approximieren („lernen“) welche genutzt werden kann um für neue, unbekannte Eingaben eine Vorhersage bzgl. der wahrscheinlichsten korrekten Ausgabe zu treffen (Mohri

et al. 2012). Eine optimale Funktion erlaubt dem Algorithmus für Eingaben welche nicht Teil der Trainingsdaten waren die korrekte Ausgabe zu bestimmen. Arten von überwachtem Lernen sind z.B. Klassifizierung und Regressionsanalyse (Alpaydin 2020). Klassifizierungsalgorithmen werden verwendet wenn die möglichen Ausgaben diskrete Elemente einer begrenzten Menge sind, z.B. ein Algorithmus welcher Emails in verschiedene Ordner einsortiert. Regressionsalgorithmen dagegen erzeugen numerische Ausgaben innerhalb eines definierten Intervalls, z.B. ein Algorithmus welcher anhand von Eigenschaften eines Hauses den Martkwert einer Immobilie schätzt.

2.2.4 Modelle

Bei der Anwendung von maschinellem Lernen wird ein sogenanntes statistisches Modell erstellt, eine Repräsentation aller Informationen oder Annahmen über die möglichen Eingaben und Ausgaben. Durch Trainingsdaten wird dieses Modell „trainiert“ und ist dann in der Lage auf Basis neuer Eingaben Vorhersagen bzgl. der wahrscheinlichsten korrekten Ausgabe zu treffen. Es existieren eine Vielzahl an erforschten und verwendeten Modellen, von denen hier nur eine Auswahl kurz vorgestellt werden sollen:

- Künstliche neuronale Netze (englisch *Artifical Neural Networks*, ANN):
  Künstliche neuronale Netze imitieren in ihrer Funktionsweise natürliche/biologische neuronale Netze wie sie im Gehirn von Menschen und anderen Lebewesen zu finden sind. Ein ANN-Modell besteht aus einer Anzahl von miteinander verbundenen Einheiten/Kotenpunkten, welche als „künstliche Neuronen“ bezeichnet werden. Jede Verbindung zwischen diesen künstlichen Neuronen kann, ähnlich den Synapsen eines biologischen Gehirns, eine Information (ein sog. „Signal“) an ein anderes künstliches Neuron übertragen. Das Empfänger-Neuron wiederum kann dieses Signal verarbeiten und dann Signale an weitere Neuronen schicken mit denen es verbunden ist. Typischerweise ist das Signal welches zwischen Neuronen verschickt wird eine reelle Zahl die durch eine nichtlineare Funktion von der Summe der Eingangssignale eines Neurons errechnet wurde. Künstliche Neuronen sowie die Verbindungen zwischen ihnen verfügen oft über eine sogenannte Gewichtung (englisch *Weight*), welche ein hindurch gesendetes Signal verstärkt oder abschwächt. Diese Gewichtung wird während des Trainingsprozesses des Modells angepasst. Künstliche Neuronen können außerdem einen Schwellenwert (englisch *Threshold*) besitzen, welcher von den empfangenen Signalen bzw. dem daraus berechneten Wert überschritten werden muss bevor das Neuron ein Signal sendet. Typischerweise werden künstliche Neuronen in Schichten (englisch *Layers*) angeordnet, welche sich in der Art und Weise der Signaltransformation unterscheiden können. Die Eingaben, in Form eines Eingabevektors, stellen das Eingangssignal der ersten Schicht (Eingabeschicht, englisch *Input Layer*) dar und werden von dort durch jede weitere Schicht weitergeleitet bis die Ausgabewerte bei der letzten Schicht (Ausgabeschicht, englisch *Output Layer*) ankommen. Es ist auch möglich dass die Signale mehrfach von den Schichten verarbeitet werden bevor das finale Ergebnis ausgegeben wird. Das sogenannte *Deep lear-*

ning (deutsch: vielschichtiges/tiefes lernen) bezeichnet künstliche neuronale Netze mit zahlreichen Schichten zwischen Ein- und Ausgabeschicht, Modelle dieses Typs haben z.B. in den Bereichen des maschinellen Sehens (englisch *Computer Vision*) und Spracherkennung (englisch *Speech Recognition*) Erfolge erzielt.

- Entscheidungsbäume (englisch *Decision Trees*):
  Entscheidungsbäume lassen sich als Baumdiagramme verstehen in welchen nacheinander getroffene Entscheidungen über Eigenschaften der Eingabeinformation, repräsentiert in den Verbindungen (sogenannten Kanten) zwischen inneren Knoten, bis zu der Vorhersage der Ausgabe, repräsentiert als Endknoten (sogenannte Blätter), führen. Entscheidungsbäume eignen sich gut dafür in Entscheidungsanalysen das Finden und Treffen von Entscheidungen visuell darzustellen.

- Support Vector Machine (SVM, deutsch *Stützvektormaschine/-methode*, nicht gebräuchlich):
  SVMs stellen eine Gruppe von Methoden des Überwachten Lernens (siehe Abschnitt 2.2.3) dar, welche sowohl bei Klassifizierungs- als auch Regressionsproblemen angewendet werden können. Traditionell haben SVMs bei Klassifizierungsproblemen die folgenden Eigenschaften:

  – Nicht probabilistisch:
    Ein nicht probabilistisches Modell gibt nur eine Vorhersage bzgl. dem Wert (Regression) bzw. der Klassenzugehörigkeit (Klassifizierung) aus, nicht jedoch eine Wahrscheinlichkeitsverteilung bzgl. mit welcher Wahrscheinlichkeit die Ausgabe als korrekt eingeschätzt wird (und mit welchen kleineren Wahrscheinlichkeiten andere Ausgaben korrekt sein könnten).

  – Binär:
    Ein binärer Klassifizirer kann nur Vorhersagen über die Klassenzugehörigkeit bei Zielmengen von zwei Klassen treffen.

  – Linear:
    Ein linearer Klassifizierer trifft eine Entscheidung durch eine Linearkombination der Eingabewerte.

  Es existieren jedoch Methoden um eine oder mehrere dieser Eigenschaften zu ändern, z.B.:

  – Platt Scaling (deutsch *Platt Skalierung*, nicht gebräuchlich):
    Platt Scaling erlaubt es aus eigentlich nicht-probabilistischen Klassifizierungsmodellen probabilistische Ausgaben (Wahrscheinlichkeitsverteilungen) zu erhalten.

  – Kernel-Trick:
    Der sogenannte Kernel-Trick erlaubt es einem eigentlich linearen Klassifizierungsmodell eine nicht-lineare Klassifizierung durchzuführen, hierfür werden die Eingaben in einen höherdimensionalen Raum transformiert in welchem sie dann einfacher separiert werden können.

- Regressionsanalyse:
  Regressionsanalyse beinhaltet eine Vielzahl an statistischen Methoden welche die Beziehung zwischen Eingabedaten (sogenannte unabhängige Variablen) und von diesen abhängigen Ausgabedaten (sogenannte abhängige Variablen) schätzen bzw. Vorhersagen bzgl. dieser treffen soll. Auf Regressionsanalyse, insbesondere die sogenannte Gratregression, wird in Abschnitt 3.2 eingegangen.

3 Algorithmen

3.1 Faltungsmatrix-Transformation mit MINIROCKET

In diesem Abschnitt sollen die Funktionsweise des in dieser Arbeit verwendeten Faltungsmatrix-Transformer (Convolutional Kernel Transform) MINIROCKET sowie Grundlagen auf welchen dieser basiert erläutert werden.

3.1.1 Faltungsmatrizen

Eine Faltungmatrix (englisch *(Convolution) Kernel*) ist ein Werkzeug welches aus einer Eingabematrix bestimmte Eigenschaften (Features) extrahieren kann. Dies geschieht indem die Faltungsmatrix über die Eingabematrix gleitet und durch Multiplikation der Werte beider Matrizen eine neue Matrix erzeugt in welcher eine bestimmte Eigenschaft stärker ausgeprägt ist. Traditionell werden Faltungsmatrizen bei der digitalen Bildbearbeitung verwendet, die folgende Abbildung veranschaulicht zum Beispiel die Anwendung einer Schärfungsfilter-Faltmatrix auf zwei Pixel eines Bildes:

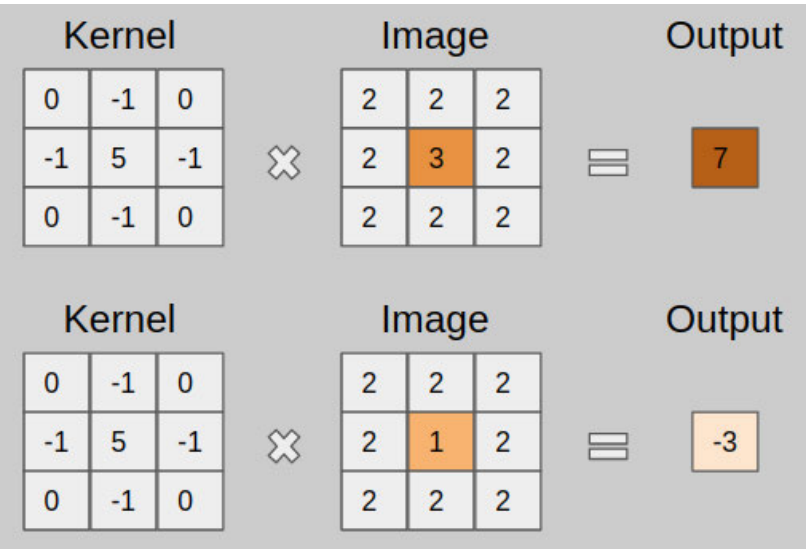


Abbildung 3.1: Schärfungsfilter-Faltungsmatrix, Quelle: Ganesh (2019)

Hierbei wird jeder Wert der Faltungsmatrix mit dem entsprechenden Pixelwert des Bildausschnitts multipliziert und die Summe aller Ergebnisse wird zum neuen Wert des mittleren Pixels. Dadurch ergeben sich die neuen Werte:

$$\begin{aligned} 3*5 \quad &+ \quad 2*-1 \quad + \quad 2*-1 \quad + \quad 2*-1 \quad + \quad 2*-1 \quad = 7 \\ 1*5 \quad &+ \quad 2*-1 \quad + \quad 2*-1 \quad + \quad 2*-1 \quad + \quad 2*-1 \quad = -3 \end{aligned} \tag{3.1}$$

Der obere Pixel mit dem Wert 3 wird durch die Kernel-Multiplikation zu 7 während der untere Pixel mit dem Wert 1 zu -3 wird, durch diese Vergrößerung des Kontrasts von 3 und 1 zu 7 und -3 wirkt das Bild schärfer als vorher.
Auch im Bereich des Maschinellen Lernens finden Faltungsmatrizen Verwendung. Anstatt manuell definierte Kernel zu verwenden lernen Convolutional Neural Networks (CNN), eine Art von Künstlichen Neuronalen Netzen (siehe Abschnitt 2.2.4), während ihres Anpassungsprozesses welche Kernel in der Lage sind latente Eigenschaften (Features) zu extrahieren.
Bei der Anwendung von Faltungsmatrizen auf Zeitreihenanalyse fällt oft die zweite Dimension weg, in diesem Fall spricht man von 1-Dimensionaler Faltungsmatrix-Multiplikation (1D Convolution), dies ist in Abbildung 3.2 zu sehen. (Ganesh 2019)

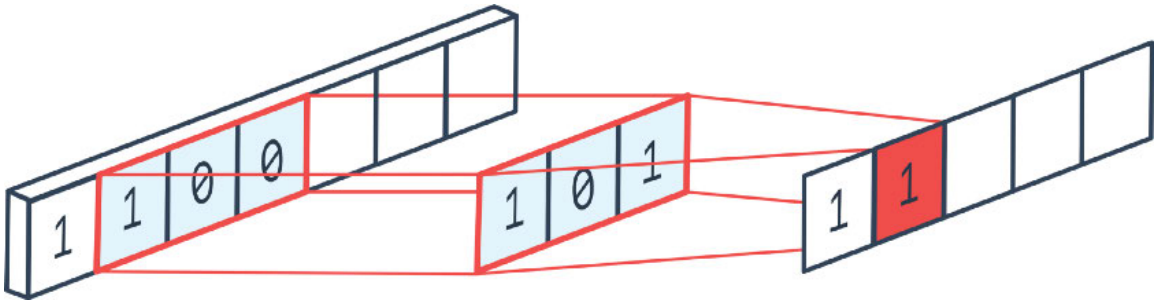


Abbildung 3.2: 1-Dimensionale Faltungsmatrix-Multiplikation, Quelle: Ganesh (2019)

Es geht jedoch sogar noch simpler, Dempster et al. (2020) bemerken nämlich, mit Verweis auf Forschungsarbeiten aus den Jahren 2009 und 2011: *„The weights of convolutional kernels are typically learned. However, it is well established that random convolutional kernels can be effective [...].“*[1]
Und aus dem Konzept mit zufällig generierten Faltungsmatrizen aus Daten Features zu extrahieren entstand 2019 ROCKET und 2020 MINIROCKET.

[1] *„Die Gewichtungen von Faltungsmatrizen werden normalerweise erlernt. Es ist jedoch etabliert das zufällige Faltungsmatrizen effektiv sein können [...].“*

3.1.2 ROCKET/MINIROCKET

> *„Most methods for time series classification that attain state-of-the-art accuracy have high computational complexity, requiring significant training time even for smaller datasets, and are intractable for larger datasets. Additionally, many existing methods focus on a single type of feature such as shape or frequency.*
> *[...] we show that simple linear classifiers using random convolutional kernels achieve state-of-the-art accuracy with a fraction of the computational expense of existing methods.“*[2]
>
> Abstract der ROCKET-Forschungsarbeit, (Dempster et al. 2020)

Der in dieser Arbeit verwendete Algorithmus MINIROCKET (**MINI**mally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform: Minimal zufälliger Faltungsmatrix-Transformer) basiert auf dem 2019 veröffentlichten ROCKET-Algorithmus (**R**and**O**m **C**onvolutional **KE**rnel **T**ransform: zufälliger Faltungsmatrix-Transformer), beide sind in ihrer Funktion der sogenannten Feature Extraction (Eigenschafts-Extrahierung) zuzuordnen, d.h. sie extrahieren in den Zeitreihendaten vorliegende Informationen bzgl. der Klassenzugehörigkeit eines Datensatzes und transformieren diese in Listen von Eigenschaften (Feature Vektor) die ein linearer Klassifizierer leichter approximieren kann als die originale Zeitreihe. Die Autoren argumentieren dass sich andere Methoden typischerweise nur auf eine Eigenschaft von Zeitreihen fokussieren, Faltungsmatrizen seien jedoch in der Lage viele der Eigenschaften zu extrahieren für welche vorher jeweils spezialisierte Techniken notwendig waren. Es sei sogar effektiv eine große Anzahl an zufälligen Faltungsmatrizen zu generieren welche in Kombination in der Lage sind Eigenschaften die für Zeitreihen-Klassifizierung relevant sind zu extrahieren, auch wenn eine einzelne Faltungsmatrix allein eine relevante Eigenschaft vielleicht nur mäßig erfasst. Im Gegensatz zu ROCKET entfernt MINIROCKET jedoch den Großteil der tatsächlichen Zufälligkeit aus dem Algorithmus und nutzt diese Änderungen geschickt aus um die Transform-Funktion dramatisch zu beschleunigen. Das macht MINIROCKET 75 mal so schnell wie ROCKET bei großen Datensätzen, bei ähnlicher Genauigkeit, und während ROCKET für die 108 Datensätze des UCR Archivs 2 Stunden brauchte schafft MINIROCKET es in 8 Minuten. Zum Vergleich: der nächstschnellste Algorithmus brauchte ungefähr 14 Stunden.

ROCKET's Transform-Funktion

Die Transform-Funktion des ROCKET-Algorithmus generiert eine sehr hohe Anzahl (Standard ist 10 000) an zufälligen Faltungsmatrizen, d.h. die Eigenschaften jeder Faltungsmatrix werden wie folgt ausgesucht:

[2] *„Die meisten Methoden der Zeitreihen-Klassifizierung dessen Genauigkeit auf dem neusten Stand der Technik ist sind hoch komplex, benötigen sogar für kleine Datensätze viel Trainingszeit und bewältigen größere Datensätze nur schwer. Außerdem konzentrieren sich viele existierende Methoden auf nur eine einzige Eigenschaft, wie Form oder Frequenz.*
[...] wir zeigen dass simple lineare Klassifizierer unter Verwendung von zufälligen Faltungsmatrizen mit einem Bruchteil des Rechenaufwands existierender Methoden den neusten Stand der Genauigkeit erreichen.“

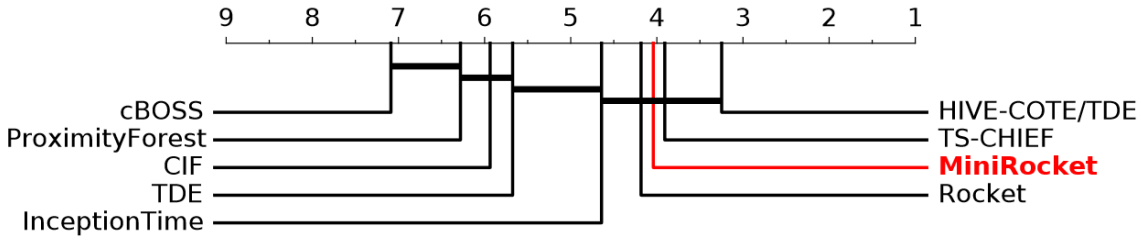


Abbildung 3.3: Durchschnittlicher Rang von MINIROCKET im Vergleich zu anderen aktuellen Methoden, gemessen über 30 Stichprobenwiederholungen von 109 Datensätzen des UCR Archivs,
Quelle: (Dempster et al. 2021)

- Länge:
  Wird zufällig aus $\{7, 9, 11\}$ ausgewählt, mit gleichmäßiger Wahrscheinlichkeit
- Gewichtungen:
  Werden aus einer Normalverteilung gezogen und um den Mittelwert zentriert
- Verzerrung (Bias):
  Wird aus einer Gleichverteilung auf dem Intervall $[-1, 1]$ gezogen
- Ausdehnung (Dilation):
  Wird aus einer exponentialen Skala gezogen (d = Ausdehnung):

$$d = \lfloor 2^x \rfloor, x \sim \mathscr{U}(0, A)$$
$$A = log_2 \frac{l_{\text{input}} - 1}{l_{\text{kernel}} - 1} \tag{3.2}$$

- Polsterung (Padding):
  50% Wahrscheinlichkeit dass bei Anwendung der generierten Faltungsmatrix Null-Polsterung (Zero Padding) an den Enden der Zeitreihe hinzugefügt wird, falls ja wird so viel hinzugefügt dass das mittlere Element der Faltungsmatrix am Anfang am ersten Wert der Zeitreihe anliegt.

Jede Zeitreihe wird nun mit allen 10 000 Faltungsmatrizen multipliziert, und aus dem Ergebnis jeder Faltungsmatrix werden mit den zwei Diskretisierungsprozessen *Global Max Pooling* (Maximalwert) und *Proportion of Positive Values* (Anteil positiver Werte) zwei Werte extrahiert die in die finale Eigenschaftsliste der Zeitreihe gelangen. Dies produziert 20 000 Eigenschaften pro Zeitreihe. (Dempster et al. 2020)

Proportion of Positive Values

Der PPV-Diskretisierungsprozess berechnet seinen den Anteil an positiven (> 0) Werten aus den Ergebnissen der Faltmatrix-Multiplikation z_i wie folgt:

$$ppv = \frac{1}{n}\sum_{i=0}^{n-1}[z_i > 0] \tag{3.3}$$

MINIROCKET's Transform-Funktion

Anstelle der Faltmatrizen mit sehr unbeschränkten Hyperparametern von ROCKET verwendet MINIROCKET eine kleinere, begrenzte Menge an Faltmatrizen mit einer sehr viel begrenzteren Auswahl an Hyperparametern, was zu seiner recheneffizienz beiträgt:

- Länge: 9
- Gewichtungen:
  Wird zufällig aus $\{-1, 2\}$ ausgewählt, mit ungleicher Verteilung (immer sechs mal -1 und drei mal 2)[3]
  (Dies ergibt 84 verschiedene Gewichtungs-Permutationen)
- Verzerrung (Bias):
  Wird für jede Faltmatrix/Ausdehnung-Kombination aus den Quantilen $[0.25, 0.5, 0.75]$ des Ergebnisses der Faltmatrix-Multiplikation mit einem zufälligen Trainingsdatensatz gezogen
- Ausdehnung (Dilation):
  Es gibt eine identische festgelegte Menge D an Ausdehnungen, angepasst an die Länge der Eingabe-Zeitreihe

$$\begin{aligned} D &= \{\lfloor 2^0 \rfloor, \ldots, \lfloor 2^{\max} \rfloor\} \\ \max &= \log_2(l_{\text{input}} - 1)/(l_{\text{kernel}} - 1) \\ l_{\text{input}} &= \text{Eingabe-Länge} \\ l_{\text{kernel}} &= \text{Faltungsmatrix-Länge} \end{aligned} \tag{3.4}$$

- Polsterung (Padding):
  Wird zufällig aufgeteilt, sodass 50% der Faltmatrix/Ausdehnung-Kombinationen Polsterung haben

Anstelle von Global Max Pooling und PPV verwendet MINIROCKET lediglich PPV, das führt dazu dass MINIROCKET nur halb so viele Eigenschaften pro Zeitreihe generiert während die Genauigkeit jedoch gleich bleibt.
(Dempster et al. 2021)

[3] Es ist wichtig dass die Summe der Gewichtungen 0 ist

MiniRocket's Transformations-Optimierungen

Die Beschränkung der möglichen Gewichtungen auf Permutationen von
$\{-1, -1, -1, -1, -1, -1, 2, 2, 2\}$ sowie die PPV-Methode erlaubt die dramatische Beschleunigung der Transform-Funktion durch vier Optimierungen:

1. PPV für W und -W gleichzeitig berechnen:
   Diese Optimierung wird durch die mathematischen Eigenschaften der begrenzten Gewichtungs-Varianten und PPV ermöglicht.
   Es gilt folgendes:

   $$\text{PPV} = 1 - \text{PNV (Proportion of Negative Values, Anteil negativer Werte)} \quad (3.5)$$

   D.h. die Berechnung von PNV kostet keine zusätzliche Rechenleistung. Da PNV jedoch für die invertierte Faltungsmatrix PPV darstellt erlaubt dies ohne eine zusätzliche Faltmatrix-Multiplikation einen weiteren Wert für die Eigenschaftsliste zu bekommen.

2. Das Ergebnis einer Faltmatrix-Multiplikation wiederverwenden um mehrere Eigenschaften zu berechnen:
   Bei kleineren Ausdehnungen kann das Ergebnis derselben Faltmatrix-Multiplikation für die Berechnung mehrerer Eigenschaften verwendet werden, z.B. indem mit verschiedenen Verzerrungen multipliziert wird.

3. Multiplikationen bei der Faltmatrix-Multiplikation vermeiden:
   Da die Gewichtungen auf zwei unterschiedliche Werte begrenzt sind können die meisten Multiplikationen mathematisch ausgeklammert und durch Addition ersetzt werden. Es müssen für jede Zeitreihe X lediglich einmal $-1 * X$ und $2 * X$ berechnet werden und dann können diese Werte verwendet werden um die restlichen Operationen durch Addition auszuführen.

4. Für jede Ausdehnung alle Faltmatrizen (fast) gleichzeitig berechnen:
   Hierbei werden die Ergebnisse der Faltmatrix-Multiplikationen noch weiter vorberechnet und wiederverwendet, die grundlegende Idee besteht darin anstatt $-1 * X$ und $2 * X$ separat zu berechnen in einem einzigen Schritt $(-1 + 2) * X$ zu berechnen.

(Amidon 2021)

3.2 Klassifizierung mit `RidgeClassifierCV`

In diesem Abschnitt sollen die Funktionsweise des in dieser Arbeit verwendeten Klassifizierungsalgorithmus `RidgeClassifierCV` sowie Grundlagen auf welchen diese basiert erläutert werden.

3.2.1 Multivariate Regression

`RidgeClassifierCV` löst das dieser Arbeit zugrundeliegende Multiklassen-Klassifikationsproblem durch die Reinterpretation des Problems als sogenannte Multivariate Regression.
Multivariate Regression (oder Multi-Output Regression), nicht zu verwechseln mit Multipler Regression, bezeichnet den Prozess, basierend auf Eingabewerten (unabhängige Variablen), eine Vorhersage bzgl. mehrerer Zielgrößen (abhängige Variablen) zu treffen (Brownlee 2020, Tutz 2014).
Im Fall von `RidgeClassifierCV` wird für jede der potentiellen Klassen (Techniken) ein numerischer Wert ausgegeben, der höchste dieser Werte entspricht der spezifischen Vorhersage. Dies wird intern durch eine sogenannte One-vs-Rest Strategie (deutsch *Einer-gegen-den-Rest Strategie*, nicht gebräuchlich) erreicht.

3.2.2 One-vs-Rest

One-vs-Rest ist eine Strategie mit welcher Algorithmen, die eigentlich nur für binäre Klassifikationsprobleme ausgelegt sind, auf Multiklassen-Klassifikationsprobleme angewendet werden können. Bei der One-vs-Rest Strategie wird für jede Klasse ***k*** ein eigenes binäres Klassifikationsmodell trainiert, welches als Zielgrößen nur ***k*** oder ***nicht k*** vorhersagt. Damit die One-vs-Rest Strategie angewendet werden kann ist es notwendig dass die einzelnen Modelle ihre binäre Vorhersage nicht in Form eines diskreten Ja/Nein-Werts treffen sondern einen numerischen Wahrscheinlichkeit-ähnlichen Wert ausgeben. Der höchste dieser Werte wird dann als Vorhersage ausgegeben. (Brownlee 2020)
(scikit-learn o.J.)

3.2.3 Lineare Regression

Die verbreitetste und simpelste Variante der Regressionsanalyse ist die lineare Regression, in welcher versucht wird eine Gerade, bzw. bei der Multiplen Linearen Regression eine komplexere Funktion solange diese linear ist, zu bestimmen welche die Eingabedaten am besten approximiert. Die simpelste lineare Regressionsfunktion ist eine Gerade $Y = a + b * X$, hierbei steht a für den Schnittpunkt mit der y-Achse, b für die Steigung und X für die unabhängige Variable und das Ergebnis der Funktion Y für die abhängige Variable. Das Ziel der Regression ist nun a und b so zu bestimmen, dass für eine beliebige unabhängige Variable X das Ergebnis der Funktion die abhängige Variable Y möglichst nah approximiert. Es existieren verschiedene Metriken nach denen der Genauigkeitsgrad der Approximation evaluiert werden kann, die verbreiteteste ist die sogenannte Methode der kleinsten Quadrate (siehe Abschnitt 4.4.1). Der Anpassungsprozess eines Regressionsmodells wird davon geleitet diesen Wert bzw. diese Funktion, die sogenannte Verlustfunktion (englisch *Loss Function*) zu minimieren. (Sunil 2015)

3.2.4 Gratregression

Die Gratregression (englisch *Ridge Regression*) unterscheidet sich von der simplen linearen Regression durch die sogenannte Regularisierung, ein Verfahren welches die Größe der Koeffizienten der Regressionsfunktion begrenzt und so Überanpassung (siehe Abschnitt 4.4.2) vermeiden kann. Dazu wird der zu minimierenden Verlustfunktion (z.B. Methode der kleinsten Quadrate, siehe Abschnitt 4.4.1) ein weiterer Term hinzugefügt, ein sogenannter Strafterm (englisch *Penalty Term*):

$$\alpha * \sum_{j=1}^{p} \beta_j^2 \tag{3.6}$$

Hierbei steht β_j für den j-ten Koeffizienten der Regressionsfunktion und α für einen festen bzw. festzulegenden Faktor (einen sogenannten Hyperparameter, siehe Abschnitt 4.4.2). Es handelt sich also um die Summe der quadrierten Koeffizienten der Regressionsfunktion, dessen Einfluß auf die Verlustfunktion vom Faktor α abhängt. Dies führt dazu dass das Regressionsmodell seinen Anpassungsprozess sowohl an der Minimierung der Approximierungsfehler als auch der Koeffizienten ausrichtet. (Nagpal 2017)

3.2.5 Normalisierung

Normalisierung zählt zu den Methoden der sogenannten Feature Skalierung (englisch *Feature Scaling*) und findet normalerweise statt bevor die Eingabedaten (Features) eines Modells weiterverarbeitet werden, in der Phase des sogenannten Data Preprocessing (Datenvorverarbeitung). Normalisierung ist der Prozess die Elemente individueller Eingabevektoren eines Modells so zu skalieren und zu verschieben dass sie Werte im Intervall $[0,1]$ annehmen und der Eingabevektor die Länge des Einheitsvektors, d.h. die Länge 1 hat. Dies sorgt dafür dass Optimisierungsverfahren, wie das von manchen Regressionsmodellen angewandte Gradientenverfahren, korrekt funktionieren. Beim `RidgeClassifierCV` geschieht die Normalisierung durch die Subtraktion des Durchschnittswerts des Eingabevektors und der darauf folgenden Division durch die sogenannte L2-Norm, der geläufigen Norm für die Länge von Vektoren.
Sei $\mathbf{x}$ ein folgendermaßen definierter Vektor:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{3.7}$$

Dann ist die L2-Norm folgendermaßen definiert:

$$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2} \tag{3.8}$$

(scikit-learn o.J., Weisstein o.J.)

4 Konzeption

4.1 Funktionalitäten

Folgende Funktionalitäten wurden in der Konzeptionsphase festgelegt:

1. Singuläre Klassifizierung in diskretem Zeitfenster
   Die App soll eine Technik, die in einem diskreten Zeitfenster ausgeführt wird, korrekt klassifizieren können.

2. Kontinuierliche Klassifizierung
   Die App soll Techniken, welche kontinuierlich hintereinander ausgeführt werden (*Schattenboxen*, siehe Fußnote auf Seite 5), in Echtzeit erkennen, isolieren und klassifizieren können.

4.2 Trainingsdaten

Wie in Abschnitt 2.2.3 dargelegt sind für das Trainieren eines SL-Algorithmus Trainingsdaten erforderlich.

4.2.1 Techniken

Für die Trainingsdaten wurden die folgenden 16 Techniken ausgewählt:

- *Jab*:
  Ein gerade ausgeführter Schlag mit der linken[1] Hand.

- *Cross*:
  Ein gerade ausgeführter Schlag mit der rechten[1] Hand.

- Linker/Rechter Haken (*Hook*):
  Ein horizontal ausgeführter Schlag mit geknicktem Arm welcher seitlich auf's Ziel trifft.

- Linker/Rechter Körperhaken (*Bodyshot*):
  Ein Haken der etwa auf Höhe vom unteren Ende des Rippenbogens seitlich das Ziel trifft.

[1] Bei Linkshändern würde ein Jab mit der rechten und ein Cross mit der linken Hand ausgeführt werden

- Linker/Rechter Aufwärtshaken (*Uppercut*):
  Ein vertikal ausgeführter Schlag mit geknicktem Arm welcher von unten auf's Ziel trifft.

- Linker/Rechter *Lowkick*:
  Ein Tritt welcher seitlich auf Höhe des Oberschenkels auf's Ziel trifft.

- Linker/Rechter *Bodykick*:
  Ein Tritt welcher seitlich auf Höhe des Rippenbogens auf's Ziel trifft.

- Linker/Rechter *Headkick*:
  Ein Tritt welcher seitlich auf Höhe des Kopfs auf's Ziel trifft.

- Linker/Rechter *Frontkick*:
  Ein Tritt welcher frontal auf Höhe des Brustbeins auf's Ziel trifft.

Zur Form der Ausführung sei zusätzlich noch gesagt:

1. Bei Schlägen wird die nicht-schlagende Hand grundsätzlich zum eigenen Kopf geführt um als Deckung zu dienen.

2. Ähnlich verhält sich die gegenüberliegende Hand auch bei Tritten, hier wird die Hand der tretenden Seite aber zusätzlich als „Gegenbewegung" dem Tritt entgegen in einem Bogen nach hinten bewegt.

4.2.2 Anforderungen

Die Trainingsdaten müssen zwei Anforderungen erfüllen:

1. Sie müssen mit einem Movesense Sensor aufgenommen sein:
   Die Datensätze mit welchen ein SL-Algorithmus trainiert wird müssen denen gleichen die dieser in der späteren Anwendung begegnen wird, sonst ist das trainierte Modell nicht in der Lage die korrekte Vorhersage zu machen. Dementsprechend ist es notwendig dass die Trainingsdaten mit der selben Hardware (Movesense Sensoren) erstellt werden die in der finalen Applikation Verwendung findet.

2. Sie müssen cirka 2 Sekunden lang sein:
   Obwohl es möglich ist Schlag-Techniken sehr schnell auszuführen dauert eine korrekt und vollständig (d.h. bis zur neutralen Ausgangsposition zurück) ausgeführte Tritt-Technik cirka 1.5-2 Sekunden. Daher ist es notwendig ein Zeitfenster von mindestens 1.5-2 Sekunden aufzunehmen damit auch die in Abschnitt 4.2.1 definierten Tritt-Techniken komplett ausgeführt werden können.

Da kein Datensatz der oben definierten Techniken zu den hier definierten Anforderungen existiert müssen selbst Trainingsdaten erstellt werden.

4.3 Scan-Algorithmus

Um der Anforderung an die Funktionalität der kontinuierlichen Klassifizierung (siehe Abschnitt 4.1) gerecht zu werden ist es notwendig einen Algorithmus zu programmieren welcher in der Lage ist in Echtzeit (d.h. ohne nennenswerte Verzögerung) ein kontinuierliches Eingangssignal aus Sensordaten zu analysieren, zu erkennen wo eine Technik anfängt und aufhört, und den entsprechenden Teil des Signals zu isolieren und zu klassifizieren.
Um überhaupt zu erkennen wo die Ausführung einer Technik beginnt, also wo ein Sensor eine „signifikante“ Bewegung misst, muss ein Schwellenwert gefunden werden den ein Sensor-Signal überschreiten muss um die Analyse eines Zeitabschnitts zu aktivieren. Dieser Schwellenwert muss folgende Eigenschaften haben:

- Er muss hoch genug sein dass die Beschleunigungsmessungen bei Stillstand (bzw. insignifikanten Bewegungen) diesen nicht überschreiten.
- Er muss niedrig genug sein dass die Beschleunigungsmessungen bei der Ausführung einer Technik diesen mit sehr hoher Wahrscheinlichkeit überschreiten.
- Aufgrund der, in den visualisierten Datensätzen (siehe Abschnitt 6.2.2) beobachteten, kleineren Beschleunigungswerte von Fußgelenk-Messungen im Vergleich zu Handgelenk-Messungen muss der Schwellenwert für diese niedriger sein.

Mithilfe der Erkentnisse der visuellen Analyse der Testdatensätze in Abschnitt 6.2.2 sowie in Python/Jupyter ausgeführten Experimenten wurde ein Scan-Algorithmus definiert welcher nach folgenden Schritten vorgeht:

1. Das erste Signal über dem festgelegten Schwellenwert finden.
2. Von diesem zeitlichen Punkt aus ein Suchfenster fester Breite „nach rechts“[2] aufspannen.
3. Innerhalb dieses Suchfensters nach einem zu definierenden Kriterium (z.B. maximalem Signal) den Zeitpunkt finden an welchem die stärkste Aktivität gemessen wird, sowie den zugehörigen Sensor bestimmen (dieser wird als „ausführender“ Sensor angenommen). Falls es sich beim bestimmten Sensor um einen Fußgelenk-Sensor handelt wird davon ausgegangen dass es sich bei der aktuellen Technik um einen Tritt handelt, in diesem Fall wird das Suchfenster nochmals verbreitert und erneut nach dem aktivsten Sensor gesucht um sicherzugehen im Fall einer Tritts die gesamte Ausführung mit einzuschließen.[3]
4. Ein Fenster um die vermutete Technik ziehen welches zwar die gesamte Ausführung der Technik in sich einschließt, damit diese korrekt klassifiziert weden kann, jedoch

[2] In zeitlich positive Richtung, also nachfolgende, nicht vorherige, Werte mit einschließen
[3] Bei der visuellen Analyse der Testdatensätze in Abschnitt 6.2.2 wurde die Beobachtung gemacht dass Tritte bis zu doppelt so viel Zeit in Anspruch nehmen als Schlagtechniken

möglichst „eng anliegt“, um zu vermeiden Signale einer Technik welche direkt nach der aktuellen ausgeführt wird mit einzuschließen. Die vordere und hintere Grenze dieses Fensters wird auf folgende Weise bestimmt:

a) Als vordere Grenze wird der erste Zeitpunkt innerhalb des Suchfensters gesetzt an welchem einer der Signalströme des ausführenden Sensors zum ersten mal den Schwellenwert überschreitet.

b) Als hintere Grenze wird, ausgehend vom Endpunkt des Suchfensters (aus Schritt 2/3), der späteste Zeitpunkt genommen an welchem ein Signal des ausführenden Sensors noch über dem Schwellenwert liegt.

5. Das Fenster welches im letzten Schritt bestimmt wurde wird ausgeschnitten und, zentriert und mit Nullwerten auf die korrekte Breite gebracht, an den Klassifizierungs-Algorithmus weitergegeben.

4.4 Evaluationsmethodik

4.4.1 Methode der kleinsten Quadrate

Die Methode der kleinsten Quadrate (englisch *Ordinary Least Squares*) ist die am weitesten verbreiteste Methode den Grad der Anpassung einer Funktion an Datenpunkte, die sogenannte Anpassungsgüte, zu messen. Hierfür wird lediglich die Summe der quadrierten Abweichungen der Datenpunkte von der Funktion kalkuliert (siehe Abbildung 4.1). Das Quadrieren der Abweichungen sorgt dafür dass sich negative und positive Werte nicht gegenseitig ausgleichen. (Sunil 2015)

4.4.2 Leave-One-Out-Kreuzvalidierung

Der in dieser Arbeit verwendete Klassifizierer `RidgeClassifierCV` (siehe Abschnitt 6.2.3) führt als Teil seines Anpassungsprozesses (`fit`-Funktion) eine sogenannte Leave-One-Out-Kreuzvalidierung durch, dies hat folgenden Zweck:

- um Überanpassung (englisch *overfitting*) zu vermeiden.
- um aus den ihm übergebenen `alpha`-Werten den besten α-Hyperparameter für die Regularisierung (siehe Abschnitt 3.2.4) auszuwählen.

Die Definition und der Zweck der Kreuzvalidierung, das Konzept der Überanpassung, sowie Hyperparameter sollen in diesem Abschnitt erklärt werden.

Überanpassung und Unteranpassung

Brownlee (2016) erinnert daran dass eine wichtige Eigenschaft eines ML-Modells, zumindest der Bereich des Überwachten Lernens (siehe Abschnitt 2.2.3), seine Fähigkeit zur

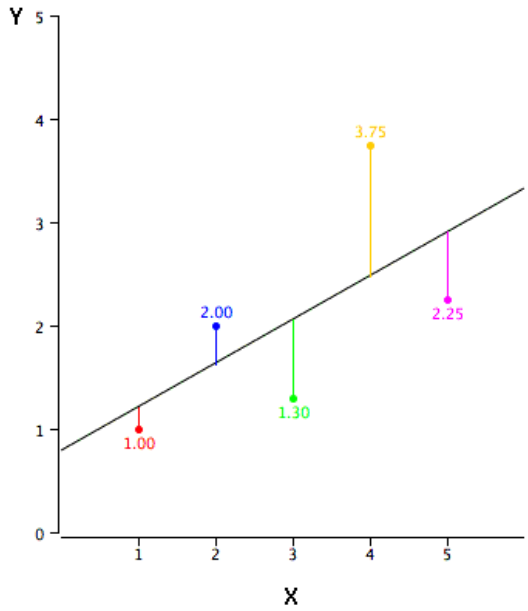


Abbildung 4.1: Quadrierte Abweichungen von einer linearen Regressionsfunktion, Quelle: analyticsvidhya.com

Generalisierung sei, d.h. wie gut ein Modell neue Daten, an welche es sich nicht anpassen konnte, approximieren kann. Dies sei wichtig da die Daten welche dem Modell zur Anpassung zur Verfügung gestellt werden immer nur eine kleine Teilmenge aller möglichen Daten darstellen, sie sind begrenzt und fast immer von statistischem Rauschen[4] betroffen. Die zwei Konzepte bzw. Probleme welche in Zusammenhang mit der Generalisierungsfähigkeit eines Modells auftreten sind Unteranpassung und Überanpassung, diese seinen die zwei häufigsten Ursachen für schlechte Leistungsfähigkeit von ML-Modellen.
Überanpassung (englisch *overfitting*) tritt auf wenn sich ein Modell zu sehr an die Trainingsdaten anpasst, indem es jede kleine Fluktuation (statistisches Rauschen) der Daten berücksichtigt. Dies führt dazu dass das Modell die Trainingsdaten zwar sehr gut approximieren kann, die Genauigkeit bei neuen Daten jedoch stark abfällt.
Unteranpassung (englisch *underfitting*) tritt auf wenn ein Modell nicht in der Lage ist weder die Trainingsdaten noch neue Daten angemessen zu approximieren. Da Unteranpassung, durch simple Evaluationsmetriken wie Kreuzvalidierung, leicht zu erkennen ist führt es sehr viel seltener zu unvorhergesehenen Problemen und lässt sich generell durch einen Wechsel des ML-Algorithmus beheben, es stellt jedoch einen anschaulichen Kontrast zum Problem der Überanpassung dar.
Abbildung 4.2 bietet einen anschaulichen Vergleich zwischen einem unter-, optimal, sowie einem überangepasstem Modell. Die Datenpunkte, welche von einer Kosinus-Funktion

[4] Statistisches Rauschen (englisch *noise*) bezeichnet unerklärbare und/oder zufällige Schwankungen in gemessenen Daten

(der *true function*) generiert und durch zufälliges statistisches Rauschen mit Varianz ausgestattet werden, sollen durch Polynom-Funktionen unterschiedlichen Grades (1, 4 und 15) approximiert werden. Über den Diagrammen wird jeweils die mittlere Quadratische Abweichung[5] der Kreuzvalidierung angezeigt. Es ist sichtbar dass das Polynom 1. Grades (links) die Daten nur sehr ungenau approximiert (Unteranpassung). Das Polynom 4. Grades (mittig) approximiert die Daten sehr viel genauer, die mittlere quadratische Abweichung ist dementsprechend geringer. Das Polynom 15. Grades (rechts) approximiert die Trainingsdaten zwar sehr gut, hat sich jedoch so stark an die Varianz der Werte angepasst dass sie zwischen den approximierten Werten sehr stark ausschlägt und dadurch schlecht auf neue Werte generalisierbar ist, die mittlere quadratische Abweichung ist dementsprechend hoch (Überanpassung).

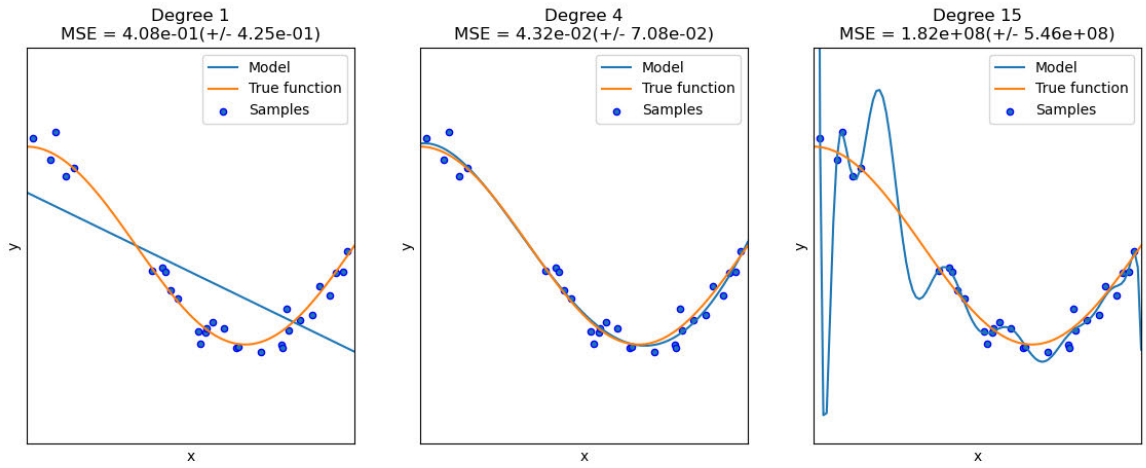


Abbildung 4.2: Unteranpassung, optimale Anpassung und Überanpassung, Quelle: scikit-learn.org

Überanpassung kann durch Methoden der Stichprobenwiederholung (englisch *resampling*) wie z.B. Kreuzvalidierung, durch Regularisierung (siehe Abschnitt 3.2.4) oder durch Validierung mit separaten Validierungsdaten (welche das Modell noch nicht kennt) vermieden werden. (Brownlee 2016)

Kreuzvalidierung

Kreuzvalidierung ist ein Verfahren welches eingesetzt wird um die Generalisierungsfähigkeit eines statistischen Modells zu validieren, d.h. um zu testen ob ein Modell welches an einen bestimmten Datensatz („Trainingsdaten“) angepasst wurde auch in der Lage ist für nicht vorher gesehene Daten („Testdaten“) korrekte Vorhersagen zu erzeugen. Bei einer regulären Validierung werden die für das Training des Modells vorhandenen Daten in Trainings- und Testdaten aufgeteilt. Das Modell wird dann an die Trainingsdaten angepasst

und danach an den Testdaten validiert, d.h. es wird gemessen wie nah die Vorhersagen des Modells bzgl. der Testdaten an den tatsächlichen Werten sind. Die Aufteilung in zwei (oder drei, siehe Hyperparameter) separate Datensätze reduziert jedoch die Anzahl an Datensätzen welche für den Anpassungsprozess zur Verfügung stehen. Es besteht außerdem die Gefahr dass die Leistungsfähigkeit des resultierenden Modells, abhängig von einer beliebigen, zufälligen Stichprobe von Trainings- und Testdaten, stark schwankt.
Eine Lösung dieses Problems ist die sogenannte Kreuzvalidierung. Bei dieser wird die Menge an Trainingsdaten in k Teilmengen zerlegt und für jede Teilmenge folgende Prozedur ausgeführt:

1. Ein Modell wird an $k-1$ der Teilmengen angepasst.
2. Das Modell wird an der übrigen Teilmenge evaluiert, z.B. mit einer Validierungsmetrik wie der mittleren quadratischen Abweichung[5]

Das Ergebnis der Kreuzvalidierung ist dann der Durchschnitt aller k berechneten Evaluationen.
Die Leave-One-Out-Kreuzvalidierung ist der Spezialfall der Kreuzvalidierung in dem $k = 1$ gilt, d.h. jede Teilmenge besteht aus allen Trainingsdaten außer einem Datensatz, an welchem das resultierende Modell gemessen wird. (scikit-learn o.J.)

Hyperparameter

Hyperparameter sind Parameter eines statistischen Modells welche im Trainingsprozess nicht direkt angepasst („gelernt“) werden sondern normalerweise vorher festgelegt werden müssen. Für den in dieser Arbeit verwendeten `RidgeClassifierCV` ist dies der α-Parameter welcher den Grad der Regularisierung (siehe Abschnitt 3.2.4) bestimmt. Um ein optimales Modell zu erhalten werden oft verschiedenene Werte für die Hyperparameter verglichen indem mit jedem Wert ein kompletter Anpassungsprozess, inklusive Kreuzvalidierung, durchgeführt wird. Da hierbei jedoch Wissen über die Testdaten in die Anpassung des Modells (bzw. dessen Hyperparameter) einfließt besteht hierbei die Gefahr der Überanpassung des Modells an die Testdaten, sodass die Ergebnisse einer Evaluation nur noch verminderte Aussagekraft über die Generalisierungsfähigkeit des Modells haben. Um dies zu vermeiden kann ein weiterer Teil der Daten als „Validierungsdaten“ (englisch *validation set*) abgespalten werden, sodass das Modell an die Trainingsdaten angepasst, dann an den Validierungsdaten validiert, und final an den Testdaten evaluiert werden kann. (scikit-learn o.J., o)

[5] Die mittlere quadratische Abweichung (englisch *mean squared error*, MSE) ist eine Evaluationsmetrik welche den Durchschnitt der quadrierten Fehler, d.h. die durchschnittliche quadrierte Differenz zwischen einer Schätzung und dem tatsächlichen Wert bezeichnet

5 Equipment, Tools und Programmiersprachen

5.1 Movesense

> *„We thought it would be shame to keep such a versatile sensor just for ourselves. [...] we decided to turn it into an open sensor platform and invite innovators across the globe to turn their sensor ideas into practice with Movesense.“*[1]
>
> Terho Lahtinen, Mitglied des Movesense-Entwicklungsteams

Die in dieser Arbeit verwendete Movesense Entwicklungsplattform (die IMUs, nachfolgend als Movesense Sensoren bezeichnet, und die dazugehörige Software) werden von der finnischen Firma Movesense hergestellt und vertrieben. Movesense entstand aus einem internen Konzept des finnischen Sportmessgeräteherstellers Suunto, welches sich als so vielseitig erwies dass es auch an Dritte zur Prototypisierung und/oder Implementierung neuer Produkte vertrieben wurde. Hierbei wurde die starke Reduzierung des notwendigen Hardware-Fachwissens, Entwicklungszeit und Kosten für Sensor-Produkte als attraktives Verkaufsargument betrachtet, vor allem für kleinere bis mittelgroße Entwickler.
Die Entwicklung der Movesense Platform begann 2015, im Jahr 2017 wurden die ersten Beta-Versionen der Sensoren an Entwickler ausgeliefert und in 2020 wurde der erste Movesense-Sensor mit Elektrokardiogramm-Funktionalität in der EU offiziell als Medizinprodukt (*medical device accessory*) registriert. Seit Oktober 2021 ist Movesense eine eigenständige Firma, losgelöst von Suunto.
Verwendung finden die Sensoren in allen Bereichen in welchen tragbare Sensor-Konzepte zu finden sind, unter anderem Sport-Equipment, Medizin, Internet-of-Things[2] Anwendungen, sowie viele weitere (allein auf der Webseite von Movesense werden über ein Dutzend an Projekten vorgestellt welche die Sensoren verwenden). (Movesense o.J.)

[1] *„Wir dachten es wäre eine Schande solch einen vielseitigen Sensor nur für uns zu behalten. [...] wir entschieden uns ihn zu einer offenen Sensor-Entwicklungsplattform zu machen und Innovatoren aus der ganzen Welt einzuladen mit Movesense ihre Sensor-Ideen in die Praxis umzusetzen.“*

[2] Das Internet of Things (IoT, deutsch Internet der Dinge) bezeichnet Technologien welche die Vernetzung und Kooperation von physischen und virtuellen Objekten mit Menschen und miteinander in lokalen oder globalen Netzwerken ermöglichen, z.B. kleine tragbare Sensoren oder RFID-Transponder

5.1.1 Hardware

Auf der Seite der Hardware verfügt der Movesense Sensor über folgende Eigenschaften:

- Maße: 10,6mm x 36,6mm (siehe Abbildung 5.1)
- Wasserfeste (bis 30m) und Schock-resistente Verarbeitung
- Schnappverbindung zum Befestigen an Riemenschnalle (siehe Abbildung 5.2)
- Betrieben durch Knopfbatterie (CR2025 3V)
- Mikrocontroller: Nordic Semiconductor nRF52832
- Prozessor: 32-Bit ARM® Cortex®-M4
- IMU mit 9 „Freiheitsgraden“: Beschleunigungssensor, Gyroskop, Magnetometer
- Herzfrequenz-Sensor
- Temperatur-Sensor
- Bluetooth-Modul für Bluetooth Low Energy (BLE) Verbindung
- Rote LED (Steuerbar)

(Movesense 2020)

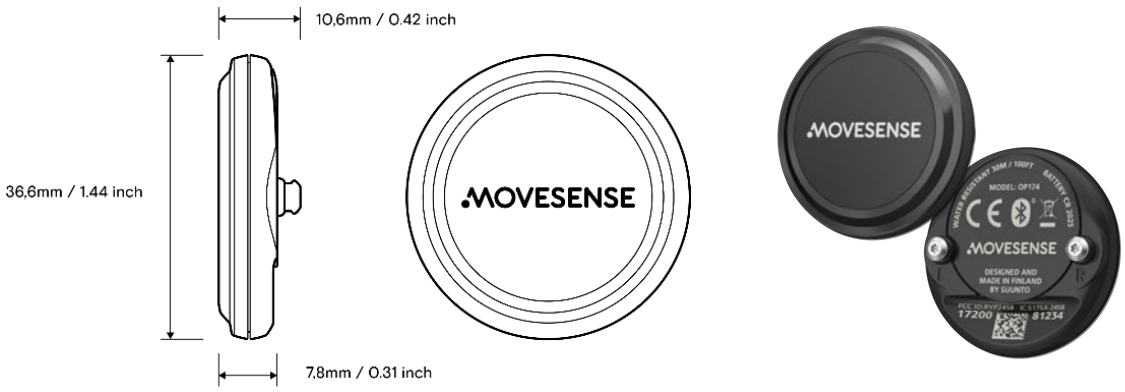


Abbildung 5.1: Movesense Sensor, Quelle: Movesense, 2020

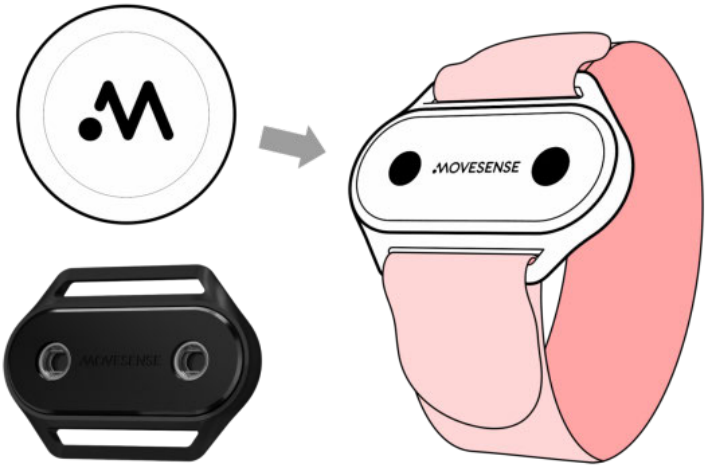


Abbildung 5.2: Movesense Riemenschnalle, Quelle Movesense

5.1.2 Software

Movesense bietet zwei separate SDKs für die Entwicklung mit der Movesense Plattform an:

1. Movesense Device Lib:
   Für die Entwicklung von Software welche auf den Sensoren installiert wird, hiermit ist es z.B. möglich Datenverarbeitung auf dem Sensor auszuführen und nur das Ergebnis an ein Endgerät zu übertragen.

2. Movesense Mobile Lib:
   Für die Entwicklung von Software/Apps welche auf Smartphones ausgeführt werden und mit Movesense Sensoren interagieren.

5.2 Python und Jupyter

Für die initiale Datenanalyse, das Trainieren des ML-Algorithmus, sowie die Entwicklung des Scan-Algorithmus wurde mit der Programmiersprache Python und der Entwicklungsumgebung *Jupyter Notebook* gearbeitet.
Python ist eine Skriptsprache[3] welche im Laufe der letzten Jahre zu einer der beliebtesten Programmiersprachen geworden ist (Stack Overflow 2021). Im Scientific Computing (SC, deutsch Wissenschaftliches Rechnen) findet Python dank Programmbibliotheken wie

[3]Eine Skriptsprache ist eine Programmiersprache dessen Code direkt, ohne vorherige Kompilierung in Maschinencode, von einem Programm (einem sogenannten Interpreter) ausgeführt wird

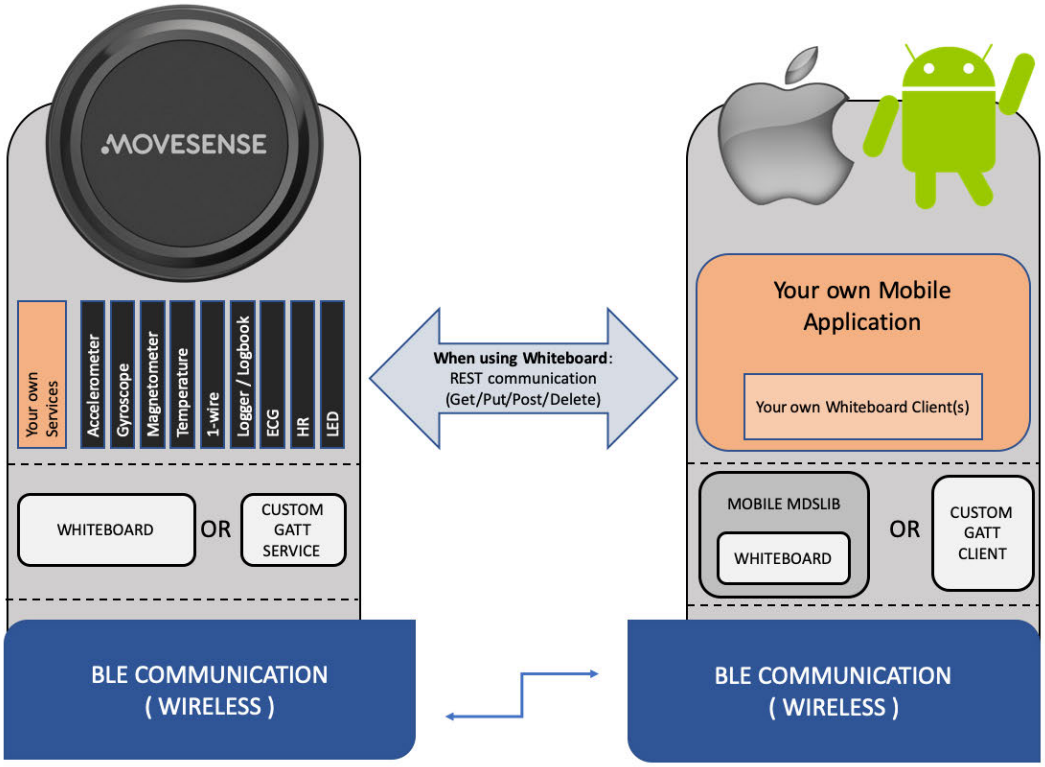


Abbildung 5.3: Movesense Software Architektur, Quelle: Movesense

SciPy, NumPy und Matplotlib weit verbreitet Verwendung (Millman & Aivazis 2011). Mithilfe von Bibliotheken wie Scikit-learn, Pytorch und TensorFlow wird Python auch im Bereich des Maschinellen Lernens großflächig eingesetzt und ist dort zum Industrie-Standard geworden (Hamel 2020).
Jupyter Notebook ist eine Web-basierte interaktive Programmierumgebung in welcher Nutzer die Möglichkeit haben Code in separate Eingabezellen zu schreiben welcher ausgeführt und dessen Ausgabe unter der Zelle angezeigt wird, diese Art von visueller Programmierumgebung wird auch als Read–Eval–Print Loop (REPL, deutsch *Lesen-Ausführen-Anzeigen Schleife*) bezeichnet.

5.2.1 Python-Bibliotheken

pandas

Die Programmbibliothek pandas bietet Datenstrukturen für numerische Tabellen und Zeitreihen an, sowie Funktionen um diese zu analysieren und manipulieren.

sklearn/scikit-learn

sklearn (bzw. scikit-learn) ist eine Programmbibliothek die Tools und Algorithmen für Maschinelles Lernen zur Verfügung stellt, sie basiert auf den numerischen/wissenschaftlichen Programmbibliotheken NumPy und SciPy.

sktime

Die Programmbibliothek sktime bietet, ähnlich wie sklearn, Tools und Algorithmen für Maschinelles Lernen, jedoch spezialisiert für Anwendungen auf Zeitreihen (z.B. Analyse, Vorhersage, Klassifikation, etc). Schnittstellen und Tools sind größtenteils mit, der sehr viel weiter verbreiteten, sklearn-Bibliothek kompatibel, so dass sktime wie eine Erweiterung von dieser verwendet werden kann.

5.3 Dart und Flutter

Für die Realisierung des App-Prototyps wurde das, in der Programmiersprache Dart implementierte, Framework Flutter verwendet.
Flutter ist ein quelloffenes Software-Entwicklungs-Kit (SDK) mit welchem Applikationen für veschiedene Plattformen (Android, iOS, Web, Windows/Linux Desktop, etc) von einer Codebasis kompiliert werden können, es wird seit 2015 von Google entwickelt. Flutter Apps werden in der von Google entwickelten Programmiersprache Dart geschrieben, einer Objektorientierten Programmiersprache die spezifisch auf die Programmierung von Benutzeroberflächen ausgerichtet wurde. Während der Entwicklungs- und Testphase einer Flutter-Applikation wird der Dart-Code in einer Virtuellen Maschine ausgeführt, dies erlaubt es Änderungen an der Dart-Codebasis sofort in der laufenden Applikation

begutachten zu können ohne diese erneut kompilieren oder neustarten zu müssen, was den Entwicklungsprozess enorm optimieren kann. Für die Veröffentlichung einer Flutter-Applikation wird der Dart-Code direkt zu Maschinencode (oder Javascript im Fall von Web-Applikationen) kompiliert, dies soll ähnliche Performance wie nativ programmierte Applikationen bewirken. (Flutter 2021)

5.3.1 Flutter-Module

mdsflutter

Das Flutter-Modul mdsflutter agiert als Wrapper[4] für Movesense's MDS (Movesense Device Service) Programmbibliothek, welche als Android Archive (AAR) Datei in die zu entwickelnde Applikation eingebunden wird und die unterste Ebene der Kommunikation mit einem Movesense Sensor implementiert. (Akdogan 2020)

5.4 C und FFI

Die in Dart integrierte Programmbibliothek ffi bietet die Möglichkeit über das sogenannte *Foreign Function Interface* (FFI, deutsch Fremdfunktionsschnittstelle) direkt mit, in der Applikation enthaltenem, C-Code zu interoperieren. Dies wird aufgrund der Steigerung der möglichen Performance vor allem für rechenintensive Programmteile empfohlen, weshalb z.B. Flutter-Module wie tflite (TensorFlow Lite) oder google_ml_kit ihre ML-Algorithmen auf diese Weise in Flutter verfügbar machen. Aus diesem Grund wurden die Programmteile dieses Projekts welche rechenintensiv sind und/oder so schnell wie möglich ausgeführt werden müssen (Scan-Algorithmus, MINIROCKET-Transformation und Klassifizierungsfunktion) in C implementiert und über das FFI in die Flutter-Applikation integriert. (Flutter 2022, Dart 2022)

[4] Ein Wrapper (deutsch „Verpackung") oder Adapter ist ein Programm welches ein anderes Programm umgibt

6 Realisierung

6.1 Erzeugung der Trainingsdaten

Für das trainieren eines SL-Algorithmus sind Trainingsdaten erforderlich (siehe Abschnitt 2.2.3). Da für den spezifischen Anwendungsfall dieser Arbeit keine vorgefertigten Datensätze existieren mussten diese selbst aufgenommen werden. Hierfür wurde ein Update der Sensor-Software durchgeführt um diese über eine BLE-Verbindung mit dem Computer verbinden zu können und mithilfe eines Web-Clients Trainingsdaten der verschiedenen Techniken aufzunehmen.

6.1.1 Device Firmware Update

Für die Aufnahme und Weiterverarbeitung von Trainingsdaten war es zuallererst notwendig ein Device Firmware Update (DFU) der Sensoren durchzuführen, da diese regulär nicht für die Verbindung zu einem Computer ausgelegt sind. Für die Durchführung dieses Updates wurde die Android-App „nRF Connect“ von Nordic Semiconductor verwendet, welche auch das im Movesense Sensor verbaute SoC *nRF52832* herstellen. Als Update wurde die Beispiel-Applikation GATT SensorData (Movesense 2021) von Movesense auf die Sensoren übertragen, diese erlaubt das Empfangen von bis zu vier Sensordaten-Streams pro Sensor mithilfe eines Web-Clients. Aufgrund der Limitierung der Anzahl an Sensordaten-Streams wurden die Messungen der drei Beschleunigungssensoren jedes Sensors verwendet.

6.1.2 Web-Client

Für die Verbindung zu den Sensoren und die Aufnahme von Trainingsdaten wurde auf Computer-Seite mit einem Web-Client[1] gearbeitet, dieser basierte auf dem Client der GATT SensorData Beispiel-Applikation (Movesense 2021). Da der Beispiel-Client jedoch nur für die Visualisierung von empfangenen Beschleinigungswerten programmiert war musste er für diesen Anwendungszweck erweitert werden.

Countdown

Um jede Aufnahme einer Technik aus einer neutralen Position heraus beginnen zu können war es notwendig einen Countdown zu implementieren (siehe Code 6.1). Der Funktions-

[1] HTML-Datei mit eingebettetem Javascript

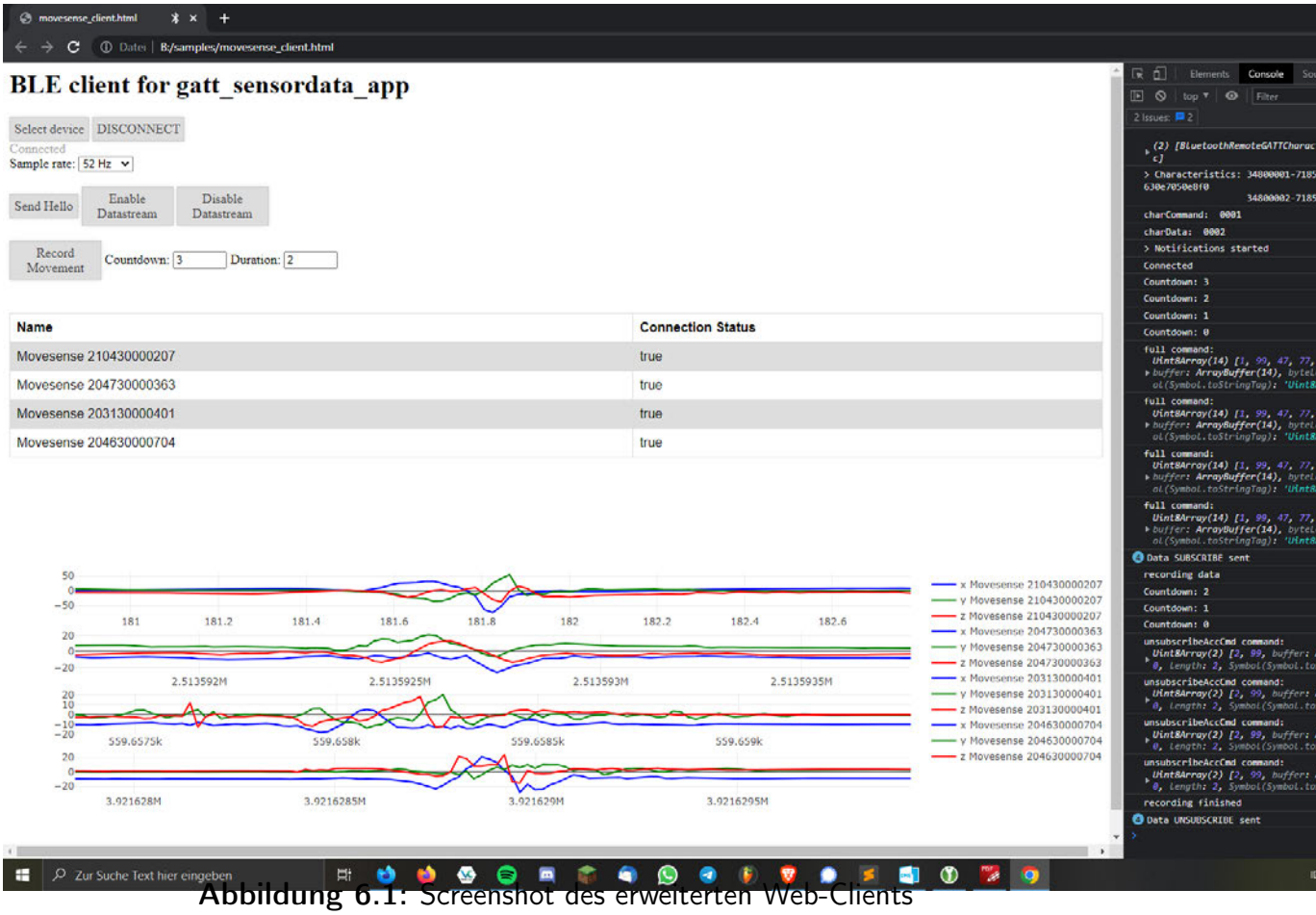


Abbildung 6.1: Screenshot des erweiterten Web-Clients

aufruf `await countdown(start)` pausiert die Ausführung des aufrufenden Skripts bis das zurückgegebene `Promise`-Objekt seine `resolve`-Funktion aufruft. Dies geschieht in der internen `count`-Funktion (welche sich selbst rekursiv aufruft) nachdem die durch den `start`-Parameter übergebene Zeitspanne abgelaufen ist.

```
function countdown(start) {
    return new Promise(resolve => {
        (function count(n) {
            console.log(`Countdown: ${n}`);
            if(n <= 0) {
                resolve();
            } else {
                setTimeout(() => {count(--n)}, 1000);
            }
        })(start);
    });
}
```

Code 6.1: Javascript `countdown` Funktion

Signalton

Zum Beginn und Ende der Aufnahme ertönt zusätzlich ein kurzer Signalton welcher durch die `playSound`-Funktion mithilfe der Web Audio API erzeugt wird (siehe Code 6.2). Hierfür wird der Funktion ein `AudioContext`-Objekt übergeben welches die verschiedenen Bearbeitungsschritte, von der Erstellung eines Tons über die Verstärkung bis zur Ausgabe, in sich vereint. Im `audioContext` wird nun ein Oszillator erstellt welcher ein Audiosignal, in diesem Fall eine pure Sinuswelle mit einer Frequenz von 440Hz[2], erzeugt. Dieses Signal wird dann verstärkt indem es durch ein sogenanntes `GainNode`-Objekt geleitet wird und danach ausgegeben. Nach einer Viertelsekunde wird das Signal wieder ausgeschaltet. (MDN 2022)

Signalerwartung

Damit die Aufnahme nicht beginnt sobald der Sendebefehl an die Sensoren geschickt wurde sondern erst sobald die ersten Daten von diesen eintreffen wurde noch die Funktion `wait_for_data` hinzugefügt (siehe Code 6.3). Der Funktionsaufruf `await wait_for_data` pausiert die Ausführung des Skripts bis von einem der Sensoren die ersten Daten empfangen werden.

[2] Den sogenannten *Kammerton*, welcher auch zur Einstimmung von Instrumenten einer Musikgruppe verwendet wird

```
function playSound(audioContext) {
    var osc = audioContext.createOscillator();
    var gainNode = audioContext.createGain();
    osc.type = 'sine';
    osc.frequency.value = 440; // Hz
    gainNode.gain.value = 0.5;
    osc.connect(gainNode).connect(audioContext.destination);
    osc.start();
    osc.stop(audioContext.currentTime + 0.25);
}
```

Code 6.2: Javascript `playSound` Funktion

```
async function wait_for_data() {
    return new Promise(resolve => {
        deviceNames.forEach(name => {
            charData[name].addEventListener(
                'characteristicvaluechanged',
                event => {
                    resolve();
                }
            );
        });
    });
}
```

Code 6.3: Javascript `wait_for_data` Funktion

CSV-Download

Um die Aufnahme einer Bewegung zu sichern und weiter zu verarbeiten war es notwendig sie abzuspeichern, hierfür wurde Code von Neill (2019) verwendet um die Daten aus dem Visualisierungs-Plugin *Plotly* zu extrahieren und als CSV-Tabellendatei zu exportieren. In Code 6.4 wird gezeigt wie eine `config`-Variable, welche danach als Parameter bei der Erstellung des Diagramms übergeben wird, mit der Funktion eines CSV-Download Knopfs ausgestattet wird. Zuerst wird dem zweidimensionalen Array `csvData` der Array `header` hinzugefügt, welcher ein `X` für die Werte der x-Achse sowie die Namen der zwölf verschiedenen Beschleunigungssensoren (drei pro Movesense-Sensor) enthält. Danach folgen Arrays welche jeweils den nächsten Wert der x-Achse sowie die zugehörigen Werte jedes Beschleunigungssensors enthalten. Wenn der `csvData`-Array mit allen Werten gefüllt wurde wird er in eine CSV-Datei umgewandelt indem die einzelnen Elemente mit Anführungsstrichen umgeben und durch Kommata getrennt werden, außerdem wird jede neue Reihe durch einen Zeilenumbruch getrennt. Zuletzt wird ein temporärer Link auf die soeben erstellte Datei erstellt und vom Skript ein „angeklickt“ um den Download-Dialog des Browser zu öffnen und die Datei abzuspeichern.

Mithilfe des angepassten Web-Clients wurden jeweils 10 Ausführungen der in Abschnitt 4.2.1 beschriebenen Techniken aufgenommen.

6.2 Machine Learning

Für das trainieren des ML-Algorithmus wurden die Trainingsdaten in Jupyter/Python importiert und weiterverarbeitet.

6.2.1 Import und Preprocessing

Basierend auf Wang (2018) wurden in Jupyter die CSV-Dateien mit den Trainingsdaten eingelesen, vorverarbeitet (*Preprocessing*) und in sogenannte Dataframes, dem primären Datenformat der pandas-Bibliothek (siehe Abschnitt 5.2.1), umgewandelt (siehe Code 6.5). Die Funktion `read_data` erhält als Argument einen Dateipfad welcher auf einen Ordner mit CSV-Dateien zeigt. Die Schleife in Zeile 3 geht jede CSV-Datei im Ordner durch und die Funktion `pandas.read` in Zeile 6 liest diese ein und wandelt sie in ein Dataframe um, mit dem `usecols`-Argument werden nur die Spalten mit den tatsächlichen Messwerten ausgewählt da die Zeitstempel in der ersten Spalte nicht benötigt werden. Da durch minimale Ungenauigkeiten im Aufnahmevorgang nicht alle Datensätze die exakt gleiche Länge haben, dies aber für die weiteren Schritte wichtig ist, werden sie in Zeile 9 auf die größte gemeinsame Länge (99) zugeschnitten. Da es ein paar seltene Lücken (fehlende Messwerte) in den Datensätzen gibt werden diese in Zeile 10 mit der `pandas.interpolate`-Funktion gefüllt, hierbei wird für einen fehlenden Wert der Durchschnitt der beiden angrenzenden Werte verwendet. Anschließend wird in Zeile 11 die

```
config = {
    modeBarButtonsToAdd: [{
        name: 'downloadCsv',
        title: 'Download data as CSV',
        icon: Plotly.Icons.disk, click: function(){
            var csvData = [];
            var plotData = document.querySelector("#graph").data;
            // Fill Header Row
            var header = ['X']
            plotData.forEach((data) => {
                header.push(data.name);
            });
            csvData.push(header);
            // Fill Data Rows
            plotData[0].x.forEach((x, i) => {
                var row = [x];
                plotData.forEach((data) => {
                    row.push(data.y[i]);
                });
                csvData.push(row);
            });
            // Quote all fields, escape quotes by doubling them,
            // add newline after each row
            var csvFile = csvData.map(
                row => row.map(
                    element => '"'
                    + ((element||"").toString().replace(/"/gi,'""'))
                    + '"' )
                    .join(",") ).join("\r\n");
            var blob = new Blob([csvFile], {
                type: 'text/csv;charset=utf-8;'
            });
            var link = document.createElement("a");
            var url = URL.createObjectURL(blob);
            link.setAttribute("href", url);
            link.setAttribute("download", "movesense-data.csv");
            link.style.visibility = 'hidden';
            document.body.appendChild(link);
            link.click();
            document.body.removeChild(link);
        }
    }],
};
```

Code 6.4: Javascript `Plotly` CSV-Download Knopf (Neill 2019)

```
1  def read_data(folder):
2      traces = []
3      for f in sorted(glob.glob(folder + "/*.csv")):
4          f = os.path.normpath(f)
5          filename = os.path.basename(f)[:-4]
6          trace = pandas.read_csv(f, header=0, \
7              usecols=[i for i in range(1,13)])\
8              .apply(pandas.to_numeric, errors='coerce')
9          trace = trace.iloc[-99:].reset_index(drop=True)\
10             .interpolate(method="linear", axis=0).T
11         nested = sktime.from_2d_array_to_nested(trace, \
12             index=trace.index).T
13         nested = pandas.concat([nested, \
14             pandas.Series([folder_name], name='label')], axis=1)
15         traces.append(nested)
16     frame = pandas.concat(traces, ignore_index=True)
17     return frame
18
19 traces = []
20 for folder_name in FOLDER_NAMES:
21     traces.append(read_data(PATH + folder_name))
22 data = pandas.concat(traces)
```

Code 6.5: Python Trainingsdaten Import

sktime-Funktion `from_2d_array_to_nested` verwendet um das Dataframe in das, von der sktime-Bibliothek bevorzugte, verschachtelte Dataframe-Format[3] zu bringen. In Zeile 13 wird nun jedem Datensatz noch die Bezeichnung der jeweiligen Technik (Ordnername) hinzugefügt. Danach werden die im `traces`-Array gesammelten Datensätze mit `pandas.concat` zu einem Dataframe konkateniert und zurückgegeben.
Unterhalb der Funktion ist dargestellt wie die Funktion verwendet wird um die in unterschiedlichen Ordnern abgespeicherten Datensätze zu importieren und schlussendlich in einem Dataframe zu kombinieren.
Nach dem Import wird das Dataframe mit den Datensätzen aufgeteilt (siehe Code 6.6) in die Eingabedaten `X` sowie die korrekten Ausgaben `y`, hierfür wird der index-basierte Zugriffsoperator `iloc` verwendet. Mit diesem wird `y` die letzte Spalte des Dataframes zugewiesen sowie `X` die restlichen. Danach werden die beiden Dataframes weiter aufgeteilt in tatsächliche Trainingsdaten, mit welchen der ML-Algorithmus trainiert wird, und einen kleineren Datensatz an Testdaten, mit welchen der trainierte Algorithmus getestet

[3] Das von der sktime-Bibliothek (siehe Abschnitt 5.2.1) bevorzugte *nested format* bezeichnet das Einfügen der gesamten Zeitreihenwerte einer Variable, in Form eines `pandas.Series`-Objekts, in eine Zelle einer Tabelle/Matrix

wird. Hierfür wird die Funktion `train_test_split` der sklearn-Bibliothek (siehe Abschnitt 5.2.1) verwendet.

```
X, y = data.iloc[:, :-1], data.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Code 6.6: Python Trainingsdaten Split

6.2.2 Visualisierung

> *„Visualization gives you answers to questions you didn't know you had."* [4]
>
> Ben Shneiderman, US-amerikanischer Informatiker

Um einen Überblick über die aufgenommenen Datensätze zu erhalten war es notwendig diese zu visualisieren. Dafür wurde die pandas-Funktion `Dataframe.plot` verwendet verwendet, welche intern die weit verbreitete Python Visualisierungs-Bibliothek Matplotlib nutzt. Das aufrufen der `plot`-Funktion wurde in eine eigene `plotDataframe`-Funktion (siehe Code 6.7) eingebettet um Redundanz zu vermeiden und auch komplexere Diagramme mit eingefärbten Bereichen oder zusätzlichen Linien zeichnen zu können. Mithilfe der `plotDataframe`-Funktion wurden nachfolgend Diagramme der Trainingsdatensätze erstellt (siehe Abbildung 6.3), dabei wurden jeweils alle Beschleunigungsmessungen eines Movesense-Sensors in einer Farbe dargestellt um klar zwischen Bewegungen der linken/rechten Hand/Fuß unterscheiden zu können. Der visuellen Analyse der Datensätze konnten folgende Beobachtungen entnommen werden:

1. Schläge dauern etwa 20-25 Samples (cirka 0.4-0.5 Sekunden), Tritte etwas länger bis doppelt so lang mit cirka 35-45 Samples (cirka 0.65-85 Sekunden).

2. Bei Schlägen sind die Beschleunigungsmessungen der schlagenden Hand klar und sichtbar stärker und länger als die der anderen Sensoren. Die Beschleunigungsmessungen bei Tritten sind weniger eindeutig, hier misst z.B. der Hand-Sensor der tretenden Seite auch starke (teils sogar stärkere) Beschleunigung als der tretende Fuß-Sensor selbst, dies hat mir der Gegenbewegung zu tun welche bei Tritten ausgeführt wird (siehe Abschnitt 4.2.1). Dies spielt später beim programmieren des Scan-Algorithmus (siehe **??**) eine Rolle.

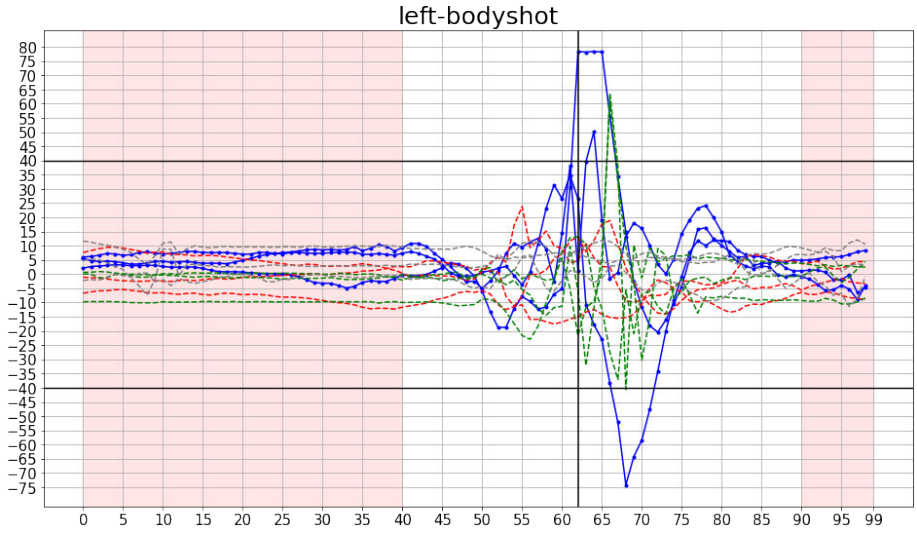
6.2.3 Machine Learning

Basierend auf der Recherche in Abschnitt 3.1 wurde der MINIROCKET-Algorithmus aus der sktime-Bibliothek für die Transformation der Datensätze von Zeitreihen in Feature-Vektoren verwendet (siehe Code 6.8). Dessen `fit`-Funktion passt den Algorithmus auf

[4] *„Visualisierung beantwortet Fragen von denen man gar nicht wusste dass man sie hat."*

```python
def plotDataframe(df, label, threshold=40, significantArea=None, \
                  significantPoint=None, fillValues=None, style='.-', \
                  legend=True):
    colors = ["blue"] * 3 + ["red"] * 3 + ["green"] * 3 + ["grey"] * 3
    plot = df.plot(figsize=(16,9), title=label, use_index=False, \
        grid=True, \
        xticks=[i for i in range(0,df.shape[0],5)]+[df.shape[0]], \
        yticks=[-i for i in \
            range(0,np.abs(df.min().min().astype(int))\+5,5)]\
            + [i for i in range(5,df.max().max().astype(int)+5,5)], \
        colormap=ListedColormap(colors), style=style, legend=legend)
    if threshold is not None:
        plot.axhline(y=threshold, linewidth=1.4, color='black')
        plot.axhline(y=-threshold, linewidth=1.4, color='black')
    if significantArea is not None and len(significantArea) == 2:
        plot.axvspan(0, significantArea[0], color='red', alpha=0.1)
        plot.axvspan(significantArea[1], df.shape[0], color='red', \
                     alpha=0.1)
    if significantPoint is not None:
        plot.axvline(x = significantPoint, linewidth=1.5, \
                     color='black')
    if fillValues is not None and threshold is not None:
        plot.fill_between(x=fillValues.index.to_list(), \
                          y1=threshold if fillValues.to_list()[0] > 0\
                          else -threshold, \
                          y2=fillValues.to_list(), color='red')
```

Code 6.7: Python `plotDataframe`-Funktion




```python
plotDataframe(X_train.iloc[5:6], y_train[5], threshold=40, \
                 significantArea=[40,90], significantPoint=62, \
                style=['.-']*3 + ['--']*9, legend=False)
```

Abbildung 6.2: Python `plotDataframe`-Aufruf

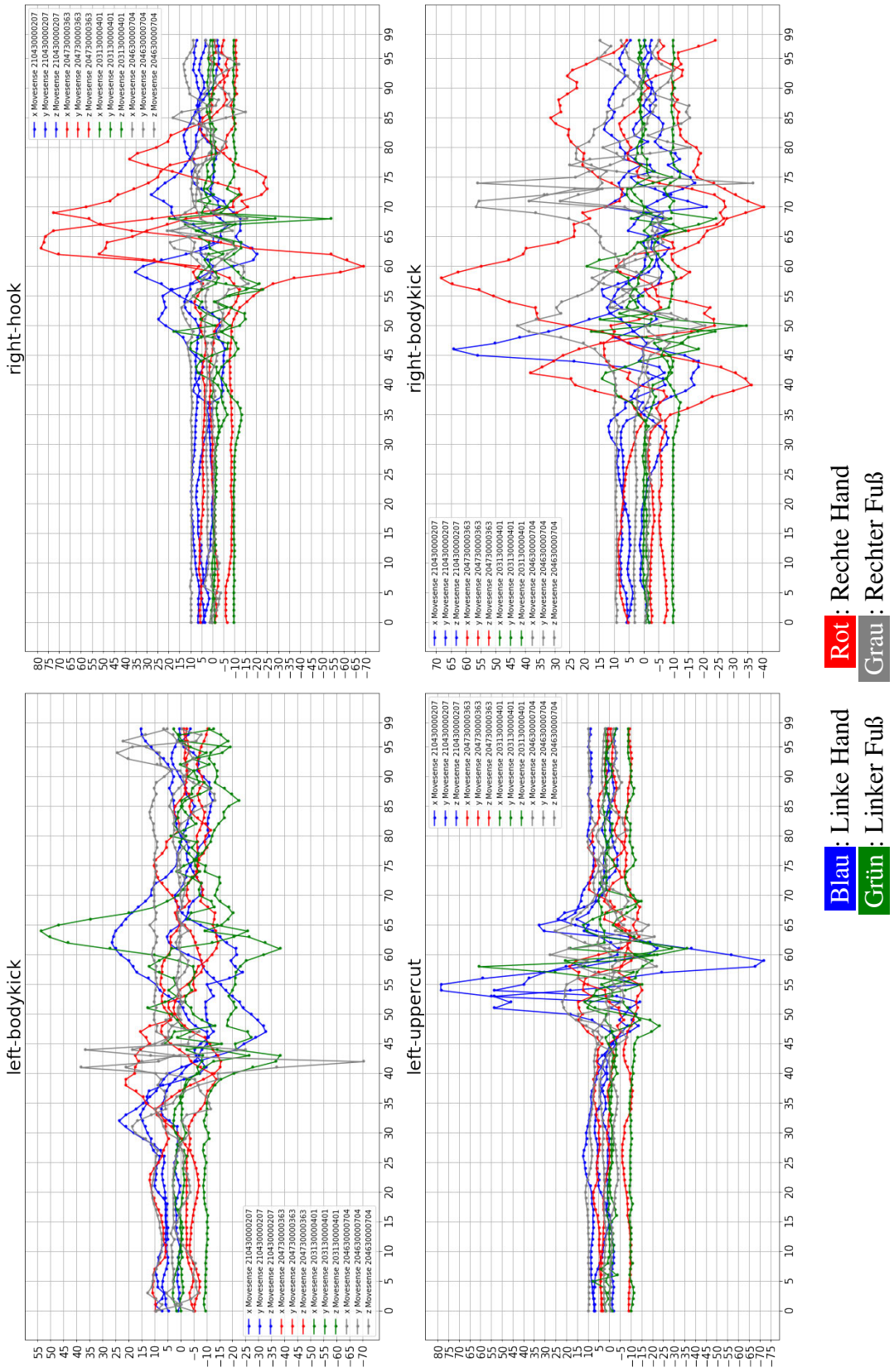


Abbildung 6.3: Visualisierte Trainingsdatensätze

```
from sktime.transformations.panel.rocket import MiniRocketMultivariate
minirocket = MiniRocketMultivariate(num_features=3192)
minirocket.fit(X_train)
X_train_transform = minirocket.transform(X_train)
```

Code 6.8: Python MINIROCKET-Transformation

die Eigenschaften der Trainingsdatensätze an und die `transform`-Funktion transformiert die zweidimensionalen Datensätze in eindimensionale Listen von Features. Hierbei wurde mit dem Argument `num_features=3250` die Anzahl der erstellten Features pro Datensatz auf 3192 (im Gegensatz zu den voreingestellten 10 000) begrenzt, der Grund wird in Abschnitt 6.4.1 erklärt.
Für die Klassifizierung wurde der lineare Klassifizierer `RidgeClassifierCV` aus der sklearn-Bibliothek verwendet (siehe Code 6.9, empfohlen von sktime (2021)), dieser führt unter anderem eine automatische Leave-One-Out-Kreuzvalidierung (siehe Abschnitt 4.4.2) durch (mehr zu `RidgeClassifierCV` in Abschnitt 3.2).

```
from sklearn.linear_model import RidgeClassifierCV
alphas = numpy.logspace(-3, 3, 10)
classifier = RidgeClassifierCV(alphas=alphas, normalize=True)
classifier.fit(X_train_transform, y_train)
```

Code 6.9: Python `RidgeClassifierCV`

Der Aufruf `logspace(-3, 3, 10)` erzeugt eine Liste von 10 Zahlen welche auf einer logarithmischen Skala gleichmäßig im Intervall von 10^{-3} bis 10^{3} verteilt sind, dies ist equivalent zu einer Liste mit Werten 10^{x} wenn x gleichmäßig verteile Werte auf einer linearen Skala (`linspace(-3, 3, 10)`) im Intervall von -3 bis 3 annimmt (Numpy 2021). Dies ergibt die Werte:

$$\texttt{linspace(-3, 3, 10)} = \left\{-3, -2\tfrac{1}{3}, -1\tfrac{2}{3}, -1, -\tfrac{1}{3}, \tfrac{1}{3}, 1, 1\tfrac{2}{3}, 2\tfrac{1}{3}, 3\right\}$$

$$\begin{aligned}\texttt{logspace(-3, 3, 10)} &= \left\{10^{-3}, 10^{-2\frac{1}{3}}, 10^{-1\frac{2}{3}}, 10^{-1}, 10^{-\frac{1}{3}}, 10^{\frac{1}{3}}, 10^{1}, 10^{1\frac{2}{3}}, 10^{2\frac{1}{3}}, 10^{3}\right\} \\ &\approx \{0.001, 0.0046, 0.022, 0.1, 0.46, 2.15, 10, 46.42, 215.44, 1000\}\end{aligned}$$

Diese Werte werden dem Klassifizierer als `alpha`-Parameter übergeben und dann als potentielle Kandidaten für den α-Faktor der Regularisierung (siehe Abschnitt 3.2.4) ausprobiert. Außerdem wird durch den `normalize`-Parameter die Normalisierung (siehe Abschnitt 3.2.5) der Eingabedaten aktiviert. Mit der `fit`-Funktion wird der Klassifizierer dann an die Trainingsdaten angepasst („trainiert“), d.h. der Algorithmus passt sein internes Modell (Koeffizienten und α-Faktor) so an dass es für jeden Trainingsdatensatz die korrekte

Ausgabe (Technik) erzeugt. Zuletzt wird der Klassifizierer mit den Testdaten, den Datensätzen welche der Klassifizierer noch nicht kennt und sich dementsprechend nicht auf diese anpassen konnte, evaluiert (siehe Code 6.10).

```
X_test_transform = minirocket.transform(X_test)
classifier.score(X_test_transform, y_test)
```

Code 6.10: Python Test-Klassifizierung

Hierfür werden die Testdatensätze (`X_test`) mit `minirocket.transform` in Listen von Features transformiert welche dann zusammen mit dem Vektor `y_test`, welcher die korrekten Ausgaben enthält, der `score`-Funktion des Klassifizierers übergeben werden. Die `score`-Funktion evaluiert die Treffsicherheit (die sogenannte „Anpassungsgüte“) des Modells und gibt diese in Form eines Werts zwischen 0 und 1 zurück. In diesem Fall produziert die `score`-Funktion den Wert `1.0`, ergo weist das trainierte Modell tatsächlich jedem Datensatz der Testdaten die korrekte Klasse (Technik) zu. Mit `classifier.alpha_` kann der gewählte α-Faktor des Modells ausgelesen werden, dieser ist `0.001`, das Modell hat sich also für den kleinstmöglichen α-Wert entschieden, was bedeutet dass die Leave-One-Out-Kreuzvalidierung die besten Ergebnisse erzielt wenn die Größe der Koeffizienten so wenig wie möglich beschränkt wird.

6.2.4 Parameter-Export

Um die trainierten Algorithmen in eine Flutter-App einzubinden war es notwendig diese aus der Jupyter/Python-Umgebung zu exportieren.

MINIROCKET-Parameter

Da MINIROCKET's `transform`-Funktion in **??** manuell in C implementiert wird ist nur der Export der Parameter des trainierten Modells notwendig, hierfür wurde eine eigene Funktion (siehe Code 6.11) geschrieben welche die Parameter als C Header-Datei exportiert.

Klassifizierer

Für den Export des trainierten linearen Klassifizierers (`RidgeClassifierCV`) wurde die Python-Bibliothek mc2gen verwendet (siehe Code 6.12), diese stellt Funktionen zur Verfügung die eine breite Auswahl an statistischen Modellen in andere Programmiersprachen transcompilieren[5] können (mc2gen 2020).

[5] Ein Transcompiler (oder Transpiler, deutsch „Quer-Übersetzer“) kompiliert Quellcode einer Programmiersprache in Quellcode einer anderen Programmiersprache

```python
from itertools import combinations
def exportMinirocketParameters(rocket):
    indices = np.array([_ for _ in combinations(np.arange(9), 3)]) \
              .tolist()
    (num_channels_per_combination, channel_indices, dilations, \
        num_features_per_dilation, biases) = rocket.parameters
    file = open('transformation_parameters.h', 'w')
    file.write('static int indices[][3] = {{{}}};\n\n' \
        .format(', '.join(map(str,indices)).replace('[','{') \
        .replace(']', '}')))
    file.write('static int numChannelsPerCombination[] = {{{}}};\n\n'\
        .format(', '.join(map(str,num_channels_per_combination)) \
        .replace('[','{').replace(']', '}')))
    file.write('static int channelIndices[] = {{{}}};\n\n' \
        .format(', '.join(map(str,channel_indices)).replace('[','{') \
        .replace(']', '}')))
    file.write('static int dilations[] = {{{}}};\n\n' \
        .format(', '.join(map(str,dilations)).replace('[','{') \
        .replace(']', '}')))
    file.write('static int numFeaturesPerDilation[] = {{{}}};\n\n' \
        .format(', '.join(map(str,num_features_per_dilation)) \
        .replace('[','{').replace(']', '}')))
    file.write('static double biases[] = {{{}}};\n\n' \
        .format(', '.join(map(str,biases)).replace('[','{') \
        .replace(']', '}')))
    file.close()
```

Code 6.11: Python MINIROCKET-Parameter-Export

Testdaten

Um die App-Implementierung der Machine Learning Pipeline[6] zu testen ist es notwendig Testdatensätze in die App zu importieren. Dafür wurde eine Funktion geschrieben welche einen beliebigen Testdatensatz als Dart-Datei exportiert (siehe Code 6.13).

[6]Eine Pipeline (oder Pipe) ist ein Datenstrom zwischen zwei oder mehr Prozessen, d.h. die Ausgabe eines Prozesses (MINIROCKET-`transform`) wird als Eingabe eines weiteren (Klassifizierer) verwendet

```
def exportClassifierToC():
    file = open('decision_function.c', 'w')
    file.write(m2c.export_to_c(classifier))
    file.close()
```

Code 6.12: Python Klassifizierer-Export

```
def exportClassificationTestdata(dataset, label):
    file = open('classification_testparameters.dart', 'w')
    file.write('class ClassificationTestParameters { \n\n')
    file.write('\tstatic const List<List<double>> dataset = {};\n\n' \
        .format(from_nested_to_3d_numpy(dataset)[0].tolist()))
    file.write('\tstatic const List<double> features = {};\n\n' \
        .format(minirocket.transform(dataset).to_numpy()[0].tolist()))
    file.write("\tstatic const String label = '{}';\n\n" \
        .format(label))
    file.write('}')
    file.close()
```

Code 6.13: Python Testdaten-Export

6.3 Flutter-App

Für die Realisierung der Flutter-App wurde die im Flutter-Modul `mdsflutter` (siehe Abschnitt 5.3.1) enthaltene Beispiel-App analysiert und erweitert. Dabei wurden diverse nicht notwendige Funktionalitäten identifiziert und entfernt, z.B. das Auslesen der Temperatur- und Puls-Messungen der Sensoren. Grundlegende Teile des Codes welche für die Verbindung und Interaktion mit Movesense Sensoren notwendig sind wurden jedoch beibehalten und als Grundlage für die zu entwickelnde Applikation verwendet.

6.3.1 Widgets

Aus Sicht des Nutzers besteht die App aus drei Widgets:

1. Scan-Widget (siehe Abbildung 6.5):
   Das Scan-Widget ist das erste was ein Nutzer beim öffnen der App sieht. Es stellt die Funktionalität zur Verfügung nach vorhandenen Movesense Sensoren zu scannen, sich mit einem oder mehreren Sensoren zu verbinden und diese in die korrekte Reihenfolge zu sortieren.

2. Countdown-Widget (siehe Abbildung 6.6):
   Das Countdown-Widget erlaubt dem Nutzer auf Knopfdruck einen Countdown zu starten nach dessen Ende ein Zeitfenster von cirka 2 Sekunden (99 Samples bei

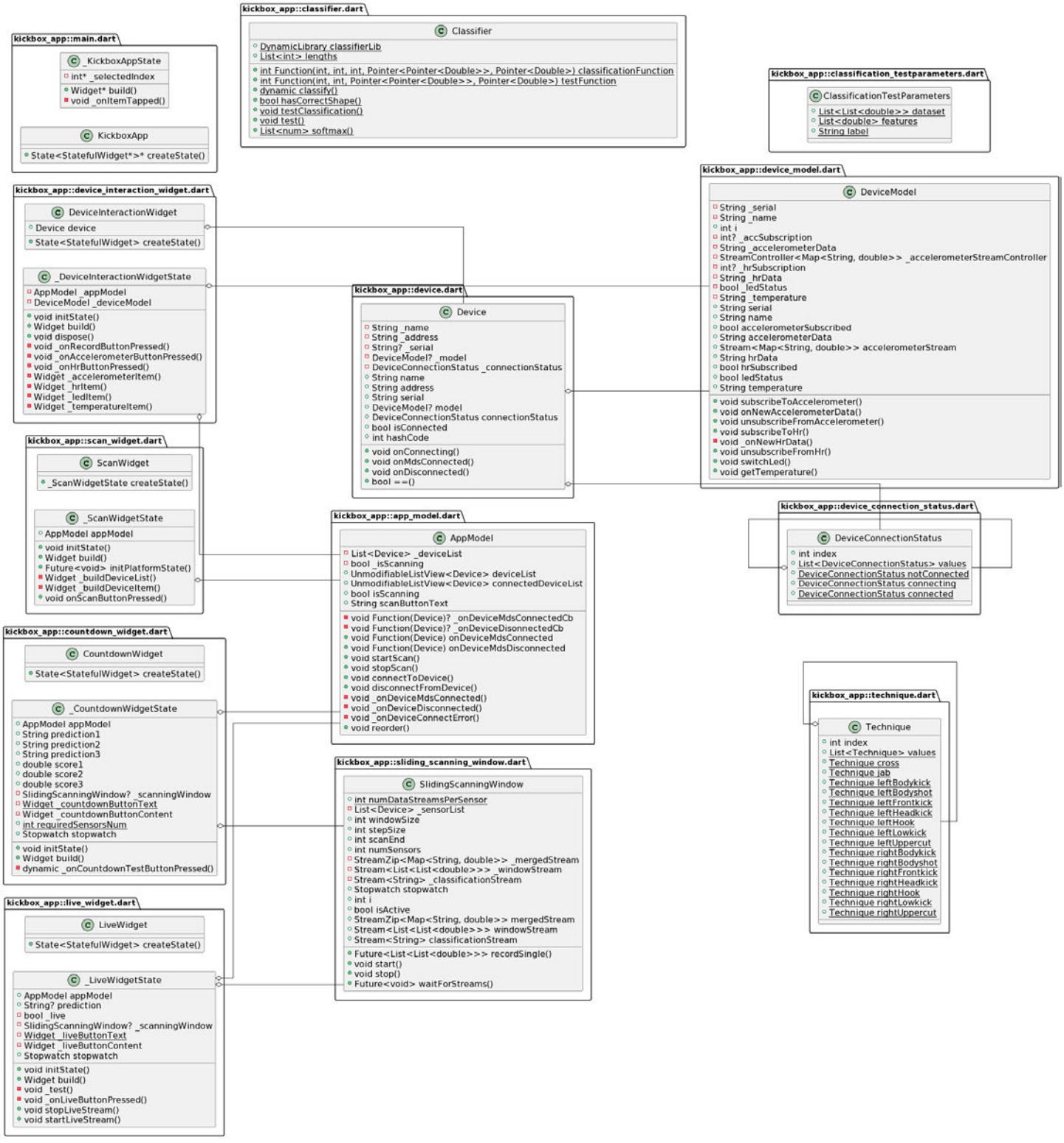


Abbildung 6.4: UML-Klassendiagramm der Flutter-App

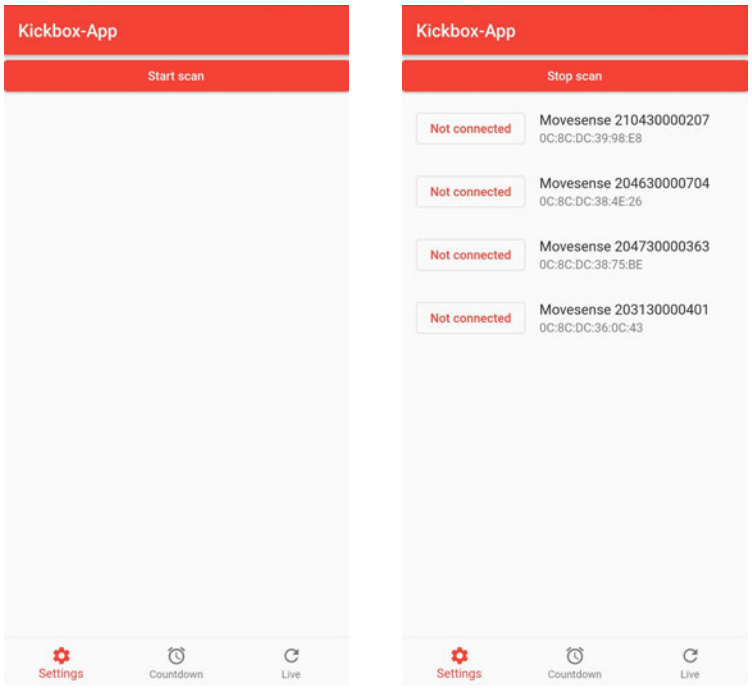


Abbildung 6.5: Scan-Widget

52Hz) besteht während dessen eine Technik ausgeführt werden kann. Nach Ende des Aufnahmefensters wird die Aufnahme klassifiziert und drei Vorhersagen (Platz 1-3) inklusive Konfidenzwert (confidence score) werden ausgegeben.

3. Live-Widget (siehe Abbildung 6.7):
   Das Live-Widget stellt die Funktionalität der kontinuierlichen Klassifizierung zur Verfügung und ist zur Verwendung während des Schattenboxens (siehe Fußnote auf Seite 5) gedacht. Auf Knopfdruck werden die Messwerte kontinuierlich dem Scan-Algorithmus (siehe **??**) zugeführt und die zuletzt klassifizierte Technik wird angezeigt.

6.3.2 Klassen

Diese Sektion stellt eine kurze Übersicht und Erläuterung der in der Flutter-App verwendeten Klassen (und ähnlicher Objekte) dar. Klassen welche von der abstrakten Klasse `StatefulWidget` (Flutter-Framework) erben werden zusammen mit ihren State-Klassen (erben von `State`-Klasse) aufgeführt, da diese sowohl strukturell als auch funktional als eine Einheit zu betrachten sind.

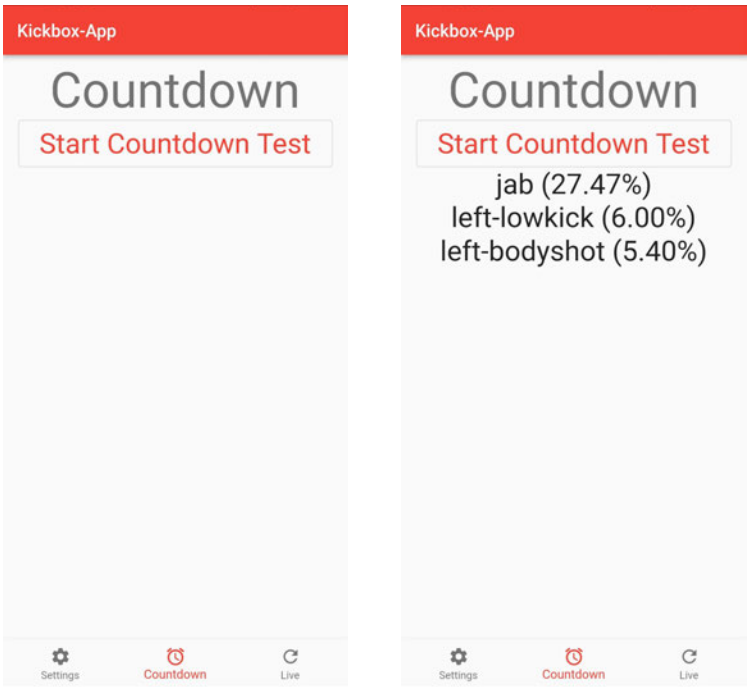


Abbildung 6.6: Countdown-Widget

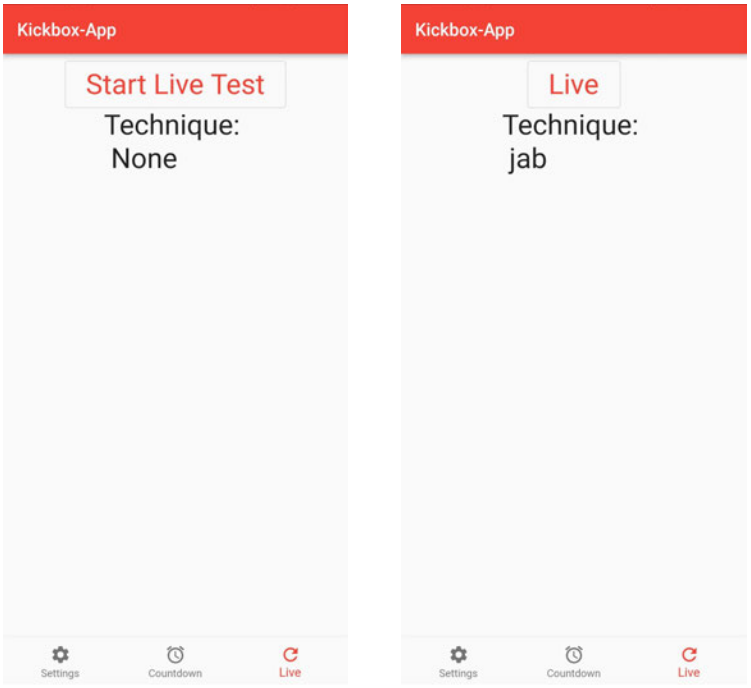


Abbildung 6.7: Live-Widget

`AppModel`[7]

Die `AppModel`-Klasse stellt das primäre Datenmodell der App dar, sie verwendet die `Mds`-Klasse des `mdsflutter`-Moduls um nach Movesense Sensoren zu scannen, sich mit diesen zu verbinden und verwaltet die Liste (`deviceList`) der verfügbaren und/oder verbundenen Sensoren.

`ClassificationTestParameters`

Diese Klasse enthält einen aus Python (siehe Abschnitt 6.2.4) exportierten Testdatensatz (Sensordaten, Features und Label) und wurde während der Entwicklung eingesetzt um die implementierten Algorithmen auf Korrektheit zu testen, hat jedoch in der finalen App keine Verwendung.

`Classifier`

Die `Classifier`-Klasse repräsentiert die Gesamtheit der in C implementierten und über das Foreign Function Interface (FFI) in die App eingebetteten Algorithmen (Scan-Algorithmus, MINIROCKET-Transformation und Klassifizierungsfunktion) und stellt die Schnittstelle zu diesen dar.

`CountdownWidget/_CountdownWidgetState`

In diesen Klassen ist die Funktionalität des Countdown-Widgets (siehe Abschnitt 6.3.1) implementiert.

`DeviceConnectionStatus`[7]

Dies ist ein sogenannter Aufzählungstyp (`enum`, englisch *enumerated type*), eine Art von Objekt welches aus einer, bei der Deklaration festgelegten, endlichen Menge von Konstanten besteht. Diese Klasse definiert drei Konstanten (`notConnected`, `connecting`, `connected`) welche den Verbindungsstatus mit einem Movesense Sensor repräsentieren.

`Device`[7]

Diese Klasse repräsentiert einen Movesense Sensor und ist für die Verbindung zu diesem zuständig. Sie speichert verschiedene Sensor-spezifische Daten (Name, Seriennummer, MAC-Adresse) und initialisiert, wenn es zu einem Verbindungsaufbau kommt, ein `DeviceModel`-Objekt welches für die Interaktion mit diesem Sensor zuständig ist.

`DeviceModel`[7]

Die `DeviceModel`-Klasse ist für die Interaktion mit einem verbundenen Sensor zuständig, z.B. für das Aus-/Anschalten der Sensor-LED oder das Anfordern und Empfangen von Messdaten.

`LiveWidget/_LiveWidgetState`

In diesen Klassen ist die Funktionalität des Live-Widgets (siehe Abschnitt 6.3.1) implementiert.

`KickboxApp/_KickboxAppState`[7]

Die `KickboxApp/_KickboxAppState`-Klassen stellen die oberste visuelle Ebene der App dar, sie initialisieren und verwalten die drei Widgets mit welchen der Nutzer interagieren kann (siehe Abschnitt 6.3.1) sowie die Logik um zwischen diesen zu navigieren. Ihre besondere Rolle als Einstiegspunkt in die App wird auch durch die Benennung der Datei in welcher sie definiert sind (`main.dart`) und die in derselben Datei vorhandene `main`-Funktion (siehe Code 6.14) sichtbar. Die `main`-Funktion findet traditionell als Haupt- und Einstiegsfunktion in diversen Programmiersprachen (z.B. C, Java, Dart) Verwendung und stellt auch die einzige Funktion dar welche außerhalb einer Klasse definiert ist. Die darin aufgerufene `runApp`-Funktion des Flutter Frameworks nimmt ein Widget entgegen und führt dieses auf dem Bildschirm (standardmäßig im Vollbildmodus) aus. Das hier übergebene Objekt ist ein sogenannter `ChangeNotifierProvider`, aus Sicht des klassischen Model-View-Viewmodel (MVVM) Entwurfsmuster wäre es als ein ViewModel zu bezeichnen, welches den Zweck erfüllt die Daten sowie Änderungen an diesen Daten vom `AppModel` (Model) an das `KickboxApp`-Widget (View) weiterzuleiten.

```dart
void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => AppModel(),
      child: const KickboxApp(),
    ),
  );
}
```

Code 6.14: Dart `main`-Funktion (`main.dart`)

`ScanWidget/_ScanWidgetState`[7]

In diesen Klassen ist die Funktionalität des Scan-Widgets (siehe Abschnitt 6.3.1) implementiert.

`SlidingScanningWindow`

Dieser Klasse wird bei ihrer Initialisierung eine Liste von verbundenen Sensoren übergeben, für welche diese dann als zentrale Schnittstelle zuständig ist. Sie hat die Funkti-

[7] Diese Klasse wurde aus der `mdsflutter` Beispiel-App übernommen und angepasst

on Messdaten von diesen anzufordern, zu empfangen und zusammenzuführen, in einem Puffer zwischenzuspeichern und entweder direkt an den Klassifizierer oder in Form von sogenannten Streams anderen Klassen verfügbar zu machen.

`Technique`

Dies ist der zweite Aufzählungstyp (siehe Abschnitt 6.3.2) der App. Die 16 Konstanten welche in `Technique` definiert sind repräsentieren die für die Klassifizierung verwendeten Techniken (siehe Abschnitt 4.2.1).

6.4 Probleme

6.4.1 Anzahl von Features VS C-Compiler

Leider musste aufgrund eines, nicht in einem angemessenen Zeitrahmen lösbares, Problem mit dem C-Compiler der App die Anzahl der vom MINIROCKET-Algorithmus generierten Eigenschaften (Features) von den regulären 10 000 auf 3192 reduziert werden, dies hatte jedoch keinen nennenswerten Einfluss auf die Leistungsfähigkeit des Algorithmus. Abbildung 6.8 zeigt dass die Performance von MINIROCKET bei der halben Anzahl an Features (4956), was nur geringfügig mehr ist als in dieser Arbeit verwendet wurde, nur geringfügig nachlässt. Ein nennenswerter Abfall in Performance scheint erst einzutreten wenn sich die Anzahl der Features der 1000 nähert.

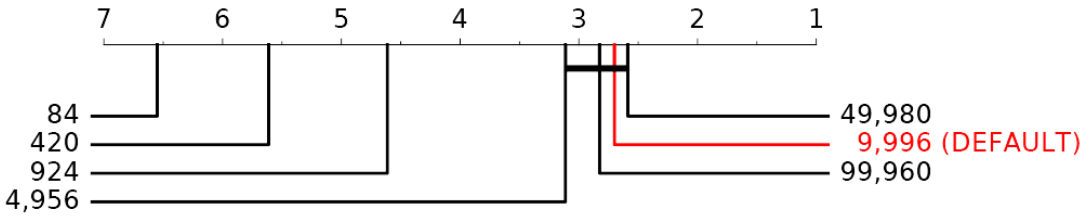


Abbildung 6.8: MINIROCKET-Performance bei Anzahl an Features, Quelle: Dempster et al. (2021)

7 Evaluation

Zur Evaluation der Klassifizierungs-Pipeline sowie des Countdown-Widgets der Flutter-App wurde jede der verwendeten Techniken 20 mal ausgeführt und die Vorhersagen des Klassifizierers ausgewertet. Die Ergebnisse wurden in Form einer Konfusionsmatrix (siehe Abbildung 7.1) dargestellt um einschätzen zu können wo die Stärken und Schwächen liegen. (Grandini et al. 2020)

Die Klassifizierungsgenauigkeit ist für die meisten Techniken gut bis perfekt, nur leider

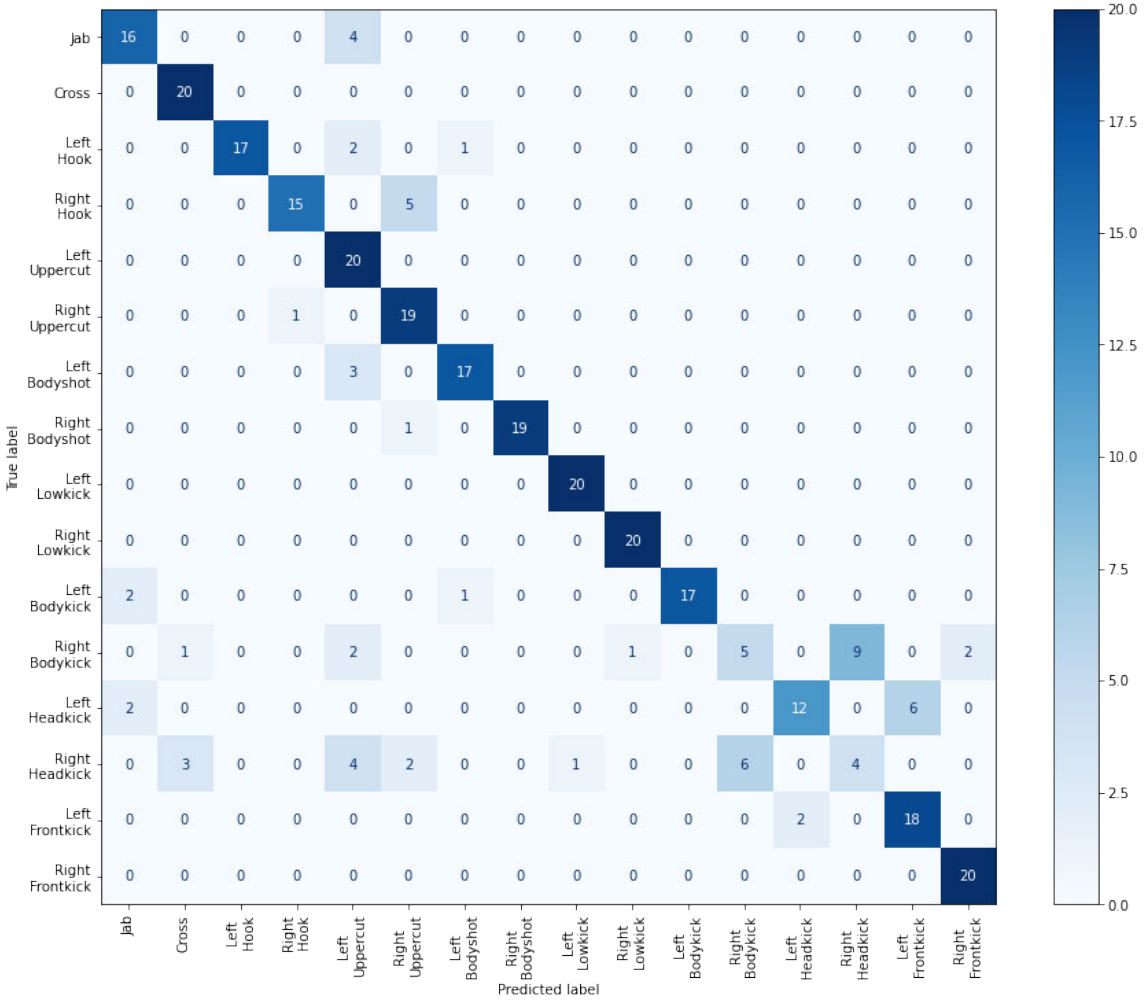


Abbildung 7.1: Konfusionsmatrix

fällt sie bei dem rechten Bodykick, sowie dem linken und rechten Headkick stark ab.

8 Fazit und Ausblick

In dieser Arbeit wurde aus einem Bedürfnis eine Idee, ein Konzept, und letztendlich ein Prototyp. Die Performance des diskreten Klassifizierungsalgorithmus (Countdown-Widget) sowie das Funktionieren der komplexen Hard- und Software-Pipeline waren größtenteils zufriedenstellend, auch wenn es noch viel zu verbessern gibt. Die geplante Funktionalität der Live-Klassifizierung war leider bis zum Ende des Projekts nicht in befriedigendem Ausmaß funktional und wurde daher auch nicht evaluiert, hier gibt es noch viel zu tun. Abgesehen von der in dieser Arbeit definierten Funktionalitäten bestehen eine Vielzahl von Möglichkeiten das Konzept einer Kickbox-App weiterzuentwickeln und mit neuen Funktionen auszustatten, z.B.:

- Datenverarbeitung im Sensor:
  Die Beschleunigungsmessungen könnten bereits innerhalb des Sensors zu einem einzelnen Signal kombiniert werden, dies würde Übertragungsbandbreite sparen.

- Individuelle interaktive Konfiguration von Parametern:
  Da sich jeder Sportler ein wenig anders bewegt und Techniken unterschiedlich ausführt würde es die flexible Einsatzfähigkeit und Verbreitung der App erhöhen wenn es die Möglichkeit gäbe dem Algorithmus die eigenen Bewegungsmuster beizubringen.

- Gyroskop-Messungen verwenden:
  Die in dieser Arbeit verwendeten Beschleunigungsmessungen haben zwar eine gute Leistungsfähigkeit erbringen können, es ist aber davon auszugehen dass die Kombination von Beschleunigungs- und Gyroskop-Messungen die Genauigkeit der Klassifizierung noch erhöhen könnte.

Abkürzungsverzeichnis

ANN Artifical Neural Network: Künstliches Neuronales Netz. 11

BLE Bluetooth Low Energy. 30, 35

CNN Convolutional Neural Network. 15

DFU Device Firmware Update/Upgrade: Firmware-Aktualisierung. 35

FFI Foreign Function Interface: Fremdfunktionsschnittstelle. 3, 34, 52

GATT Generic Attribute Profile: Generisches Attributprofil. 35

IMU Inertial Measurement Unit: Inertiale Messeinheit. 7, 8, 29, 30

IoT Internet of Things: Internet der Dinge. 29

MINIROCKET MINImally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform: Minimal zufälliger Faltungsmatrix-Transformer. 4, 14–19, 42, 54, 60

ML Machine Learning: Maschinelles Lernen. 9, 25, 26, 31, 34, 39, 41

MSE Mean Squared Error: Mittlere quadratische Abweichung. 28

PPV Proportion of Positive Values:. 18, 19

ROCKET Rand**O**m **C**onvolutional **KE**rnel **T**ransform: zufälliger Faltungsmatrix-Transformer. 15, 16, 18

REPL Read-Eval-Print Loop: Lesen-Ausführen-Anzeigen Schleife. 33

RL Reinforcement Learning: Bestärkendes Lernen. 10

SC Scientific Computing: Wissenschaftliches Rechnen. 31

SDK Software Development Kit. 31, 33

SL Supervised Learning: Überwachtes Lernen. 10, 22, 23, 35

SoC System-on-a-chip: Ein-Chip-System. 35

UL Unsupervised Learning: Unüberwachtes Lernen. 10

Code

Abbildungsverzeichnis

Literaturverzeichnis

Akdogan, Tugberk, 2020: „mdsflutter“, https://github.com/tugberka/mdsflutter, letzter Zugriff: 15. 4. 2022

Alpaydin, Ethem: *Introduction to Machine Learning*, 4. Aufl., MIT Press 2020

Amidon, Alexandra, 2021: „MiniRocket: Fast(er) and Accurate Time Series Classification“, https://towardsdatascience.com/minirocket-fast-er-and-accurate-time-series-classification-cdacca2dcbfa, letzter Zugriff: 26. 4. 2022

Brownlee, Jason, 2016: „Overfitting and Underfitting With Machine Learning Algorithms“, https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/, letzter Zugriff: 24. 4. 2022

Brownlee, Jason, 2020: „How to Develop Multi-Output Regression Models with Python“, https://machinelearningmastery.com/multi-output-regression-models-with-python/, letzter Zugriff: 25. 4. 2022

Brownlee, Jason, 2020: „One-vs-Rest and One-vs-One for Multi-Class Classification“, https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/, letzter Zugriff: 25. 4. 2022

Dart, 2022: „C interop using dart:ffi“, https://dart.dev/guides/libraries/c-interop, letzter Zugriff: 23. 4. 2022

Dempster, Angus & Petitjean, François & Webb, Geoffrey I.: „ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels“, *Data Mining and Knowledge Discovery* vol. 34, S. 1454–1495, 2020

Dempster, Angus & Schmidt, Daniel F. & Webb, Geoffrey I.: „MINIROCKET: A Very Fast (Almost) Deterministic Transform for Time Series Classification“, *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* S. 248–257, 2021

Deutscher Olympischer Sportbund, 2021: „Bestandserhebung 2021“, https://cdn.dosb.de/user_upload/www.dosb.de/uber_uns/Bestandserhebung/BE-Heft_2021.pdf, letzter Zugriff: 10. 3. 2022

Dudek, Gregory & Jenkin, Michael: „Inertial Sensors, GPS, and Odometry“, in: Siciliano, Khatib (Hrsg.): *Handbook of Robotics*, Springer 2008

Flutter, 2021: „Flutter architectural overview“, https://docs.flutter.dev/resources/architectural-overview, letzter Zugriff: 23. 4. 2022

Flutter, 2022: „Binding to native code using dart:ffi“, https://docs.flutter.dev/development/platform-integration/c-interop, letzter Zugriff: 23. 4. 2022

Frasch, Thomas, 2019: „Inertialsensoren erfassen Bewegungen und Position präzise“, https://www.all-electronics.de/elektronik-entwicklung/inertialsensoren-erfassen-bewegungen-und-position-praezise.html, letzter Zugriff: 15. 3. 2022

Ganesh, Prakhar, 2019: „Types of Convolution Kernels : Simplified“, https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37, letzter Zugriff: 26. 4. 2022

Grandini, Margherita & Bagli, Enrico & Visani, Giorgio: „Metrics for Multi-Class Classification: an Overview“, https://arxiv.org/abs/2008.05756, letzter Zugriff: 21. 4. 2022

Hamel, Greg, 2020: „Data Science Tools Popularity, animated“, https://www.kdnuggets.com/2020/06/data-science-tools-popularity-animated.html, letzter Zugriff: 23. 4. 2022

Iosa, Marco & Picerno, Pietro & Paolucci, Stefano & Morone, Giovanni: „Wearable Inertial Sensors for Human Movement Analysis“, *Expert Review of Medical Devices* vol. 13, 2016

Bayes' Witnesses, 2020: „m2cgen“, https://github.com/BayesWitnesses/m2cgen/tree/v0.9.0, letzter Zugriff: 15. 4. 2022

Mohri, Mehryar & Rostamizadeh, Afshin & Talwalkar, Ameet: *Foundations of Machine Learning*, The MIT Press 2012

Movesense: „Technical Brief“, https://www.movesense.com/wp-content/uploads/2020/12/Movesense-Spec-Sheet-12-2020.pdf, letzter Zugriff: 22. 4. 2022

Movesense: „Showcases“, https://www.movesense.com/showcase/, letzter Zugriff: 22. 4. 2022

Movesense, 2021: „GATT SensorData“, https://www.movesense.com/docs/esw/sample_applications/#gatt-sensordata, https://bitbucket.org/movesense/movesense-device-lib/src/master/samples/gatt_sensordata_app/, letzter Zugriff: 6. 9. 2021

Mozilla Developer Network, 2022: „Web Audio API“, https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API, letzter Zugriff: 1. 4. 2022

Millman, K. Jarrod & Aivazis, Michael: „Python for Scientists and Engineers“, *Computing in Science & Engineering* vol. 13, S. 9–12, 2011

Nagpal, Anuja, 2017: „L1 and L2 Regularization Methods“, https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c, letzter Zugriff: 26. 4. 2022

Neill, Richard, 2019: „Download data button“, https://github.com/plotly/plotly.js/issues/2171#issuecomment-482869511, letzter Zugriff: 1. 4. 2022

NumPy, 2021: „numpy.logspace“, https://numpy.org/doc/1.21/reference/generated/numpy.logspace.html, letzter Zugriff: 9. 4. 2022

Ruiz, Alejandro Pasos & Flynn, Michael & Large, James & Middlehurst, Matthew & Bagnall, Anthony: „The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances“, *Data Mining and Knowledge Discovery* vol. 35, 2021

Russell, Stuart J. & Norvig, Peter: *Artificial Intelligence: A Modern Approach*, 3. Aufl., Prentice Hall 2009

scikit-learn: „Classification“, https://scikit-learn.org/stable/modules/linear_model.html#classification, letzter Zugriff: 25. 4. 2022

scikit-learn: „Cross-validation: evaluating estimator performance“, https://scikit-learn.org/stable/modules/cross_validation.html, letzter Zugriff: 24. 4. 2022

scikit-learn: „Normalization“, https://scikit-learn.org/stable/modules/preprocessing.html#normalization, letzter Zugriff: 26. 4. 2022

scikit-learn: „Tuning the hyper-parameters of an estimator“, https://scikit-learn.org/stable/modules/grid_search.html, letzter Zugriff: 24. 4. 2022

sktime, 2021: „MiniRocket“, https://github.com/alan-turing-institute/sktime/blob/v0.8.0/examples/minirocket.ipynb, letzter Zugriff: 9. 4. 2022

Stack Overflow, 2021: „2021 Developer Survey“, https://insights.stackoverflow.com/survey/2021/, letzter Zugriff: 22. 4. 2022

Sunil, 2015: „7 Regression Techniques you should know!“, https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/, letzter Zugriff: 26. 4. 2022

Tutz, Gerhard, 2014: „Multivariate Regression“, http://semsto.userweb.mwn.de/Lehre/statIV2014/material/MultiRegression.pdf, letzter Zugriff: 25. 4. 2022

Wang, Jennifer, 2018: „Gesture Recognition Magic Wand“, https://github.com/jewang/gesture-demo, letzter Zugriff: 3. 4. 2022

Weisstein, Eric W.: „L$\hat{2}$-Norm“, https://mathworld.wolfram.com/L2-Norm.html, letzter Zugriff: 26. 4. 2022

Xsens: „Xsens Motion Capture“, https://www.xsens.com/motion-capture, letzter Zugriff: 15. 3. 2022

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Tobias Saladauski