HAW HAMBURG

Bachelorarbeit

Dennis Kochinky

Untersuchung von radardatenbasierter Objekterkennung mithilfe maschineller Lernverfahren

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Dennis Kochinky

Untersuchung von radardatenbasierter Objekterkennung mithilfe maschineller Lernverfahren

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr. Andreas Meisel

Eingereicht am: 31. August 2020

Dennis Kochinky

Thema der Arbeit

Untersuchung von radardatenbasierter Objekterkennung mithilfe maschineller Lernverfahren

Stichworte

Radar, MIMO, maschinelle Lernverfahren, Random Forest, MLP, SVM, Sensordaten, Objekterkennung, Kreuzvalidierung

Kurzzusammenfassung

Maschinelle Lernverfahren spielen eine wichtige Rolle in der Informatik. Sie eignen sich unter anderem für Klassifizierungsprobleme. Die Daten, mit denen sie gefüttert werden, sind vielfältig. In dieser Arbeit wird die Objekterkennung mittels Radardaten untersucht, die Messdaten stammen von einem Radarsensor mit mehreren Sende- und Empfangsantennen. Zur Klassifizierung kommen verschiedene maschinelle Lernverfahren zum Einsatz, die abschließend miteinander verglichen werden.

Dennis Kochinky

Title of Thesis

Investigation of radar data-based object recognition using machine learning techniques

Keywords

Radar, MIMO, machine learning, Random Forest, MLP, SVM, sensor data, object recognition, Cross-Validation

Abstract

Machine learning algorithms play an important role in computer science. They are suitable for e.g. classification problems, the data with which they are fed are diverse. In this work, object detection is investigated using radar data, the measurement data comes from a radar sensor with several transmit and receive antennas. Various machine learning processes are used for classification, which are then compared with each other.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Maschinelle Lernverfahren lassen sich vielseitig einsetzen, sie kommen beispielsweise als Spam-Filter, zur Wettervorhersage oder zur Objekterkennung zum Einsatz. Dabei hat jedes Verfahren Vor- und Nachteile, woran sich die Anwendungsgebiete bestimmen. So gibt es Verfahren wie Convolutional Neural Networks (CNN) die durch komplexe Strukturen eine Art Gehirn bilden und vorwiegend zur Analyse von Bild- und Audiodaten genutzt werden. Während zum Beispiel der Random Forest auf Entscheidungsbäume setzt, relativ einfach ist, sich für Regression und Klassifizierung eignet und vergleichsweise wenig Rechenleistung benötigt.

Vor allem im Bereich der Automobilindustrie auf dem Weg zu autonomen Fahrzeugen spielen Klassifizierungsprobleme eine zunehmend größere Rolle. So ist es nach wie vor schwierig Verkehrsteilnehmer in die drei Gruppen Fußgänger, Radfahrer und Fahrzeuge einzuteilen. Die Auswertung von Kamerabildern mittels neuronaler Netze erfordert zum Beispiel eine hohe Rechenleistung, was den Einsatz in Echtzeitsystemen erschwert.

Radar-Daten lassen sich mit weniger Rechenaufwand auswerten, doch werden sie meist nicht zur Klassifizierung genutzt. Typischerweise nutzt man Radar-Daten um Entfernungen und Geschwindigkeiten zu messen. So kommen Radarsensoren in Blitzern (Radarfallen), im Flug- und Schiffsverkehr zum Detektieren von Hindernissen oder in der Meteorologie zum Einsatz. Doch wie in dem Paper „Radarcat“ dargestellt, lassen sich Radar-Daten auch im kleineren Umfeld für die Klassifizierung von Objekten nutzen. In dieser Arbeit wird diese Idee aufgegriffen und eine Objekterkennung anhand von Radar-Daten mithilfe maschineller Lernverfahren untersucht. Dabei kommen ein Random Forest, ein MLP und ein SVM als Lernverfahren zum Einsatz und werden miteinander verglichen.

Als Sensor wird ein sogenanntes MIMO-Radar mit zwei Sende- und 16 Empfangsantennen eingesetzt. Die zu messenden Objekte sind dabei starr und direkt vor bzw. auf dem Radarsensor platziert. Auf die Klassifizierung von bewegten Objekten wird dabei nicht eingegangen.

1.2 Ziel und Abgrenzung

In dieser Arbeit werden Radar-Daten nicht wie üblich für die Entfernungs- oder Geschwindigkeitsmessung, sondern zur Klassifizierung von unterschiedlichen Objekten genutzt. Dazu werden verschiedene maschinelle Lernverfahren eingesetzt und die Ergebnisse miteinander verglichen.

1.3 Zielgruppe

Die vorliegende Arbeit richtet sich an Leser mit grundlegenden Kenntnissen der Informatik, insbesondere im Bereich der maschinellen Lernverfahren. Darüber hinaus sollte ein Interesse an Sensoren beziehungsweise der Radartechnik vorhanden sein.

1.4 Struktur der Arbeit

Die Arbeit ist in sieben Kapitel unterteilt. Kapitel 1 führt in die Arbeit ein und erläutert die zugrundeliegende Problematik. Das zweite Kapitel beschreibt die Grundlagen der Radartechnik, die verwendeten ML-Verfahren sowie die mathematische Grundlage zur verwendeten Datenvorverarbeitung. Das dritte Kapitel stellt Methoden und Parameter zur Evaluation maschineller Lernverfahren vor. Kapitel 4 zeigt den aktuellen Stand der Technik und stellt das zugrundeliegende Experiment aus dem Paper „Radarcat“ vor. In Kapitel 5 wird die experimentelle Umsetzung sowie die verwendete Soft- und Hardware dargestellt. In Kapitel 6 werden die Ergebnisse zusammengefasst und die Ergebnisse der verschiedenen ML-Verfahren verglichen. Kapitel 7 zieht ein Fazit und gibt einen Ausblick auf die künftige Nutzung von radardatenbasierter Objekterkennung.

2 Einführung und Grundlagen

Dieses Kapitel beschreibt die Grundlagen der Radartechnik, die angewendeten maschinellen Lernverfahren sowie die verwendeten mathematischen Verfahren zur Datenvorverarbeitung.

2.1 Radargrundlagen

Radar ist die Abkürzung für „radio detection and ranging“, was übersetzt für „funkgestützte Ortung und Abstandsmessung“ steht. Die Basis der Radartechnik bilden elektromagnetische Wellen und deren Ausbreitung. Die Funktionsweise eines Radars lässt sich in fünf Phasen einteilen [8, S. 15].

In **Phase 1** wird ein Radarsignal – entsprechend dem Radarverfahren – erzeugt und durch eine Antenne ausgesendet.

In **Phase 2** breitet sich das Signal vom Radarsensor ausgehend aus.

Phase 3: Trifft das Radarsignal auf Hindernisse (Objekte), wird es von diesen zum Teil absorbiert und zum anderen Teil gestreut (und reflektiert). Dabei kann ein Phasensprung auftreten.

In **Phase 4** breitet sich ein Teil des reflektierten Signals in Richtung des Radarsensors aus.

In **Phase 5** empfängt der Radarsensor das reflektierte Signal und wertet es aus.

Dabei unterscheiden sich Radarsysteme unter anderem im Aufbau. Nachfolgend werden die wichtigsten vorgestellt.

2.1.1 Monostatisches Radar

Bei einem monostatischen Radar befinden sich die Sende- und Empfangsantenne des Radars am gleichen Ort, meist besteht das System aus nur einer Antenne, die das Senden und Empfangen von Signalen übernimmt.

2.1.2 Bistatisches Radar

Im Gegensatz zum monostatischen Radar besteht ein bistatisches Radar aus zwei getrennten Radarsensoren. Ein Sensor sendet das Radarsignal, ein anderer Sensor empfängt und wertet es aus. Dabei ist eine Kommunikationsverbindung zwischen den Sensoren nötig, um Sende- und Empfangssignal zu synchronisieren.

Besteht ein Radarsystem aus mehreren Sende- und Empfangsantennen spricht man von einem multistatischen Radar [8, S. 17].

2.1.3 Radarfrequenzen

Je nach Einsatzgebiet einer Radaranwendung bieten sich unterschiedliche Frequenzen an. Grob lassen sich Radarsysteme in drei Bereiche einteilen: das Weitbereichsradar, Mittelbereichsradar und Nahbereichsradar.

Das Nahbereichsradar nutzt Frequenzen oberhalb von 10 Gigahertz (GHz) und eignet sich für Entfernungen bis 50 km. Das Mittelbereichsradar (50 bis 100 km) verwendet Frequenzen zwischen 3 und 10 GHz. Ein Weitbereichsradar nutzt Frequenzen zwischen 1 und 5 GHz und hat eine Reichweite von über 100 Kilometern [8, S. 18].

2.2 Radarverfahren

Je nach Anwendungsgebiet kommen nicht nur unterschiedliche Frequenzen, sondern auch unterschiedliche Radarverfahren zum Einsatz. Hauptsächlich unterscheidet man zwischen Puls- und Dauerstrich-Radar.

2.2.1 Puls-Radar

Das Puls-Radar, auch Impuls-Radar genannt, kommt meist zur Entfernungsmessung zum Einsatz. Beim Puls-Radar besteht das Sendesignal aus einer periodischen Folge hochfrequenter Pulse mit kurzer Dauer. Das Empfangssignal ist eine zeitverzögerte und gegebenenfalls verzerrte Kopie des Sendesignals. Über die Signallaufzeit t, also die zeitliche Differenz zwischen Sende- und Empfangssignal lässt sich die Entfernung R mit

$$R = \frac{c \cdot t}{2} \tag{2.1}$$

bestimmen. Da der Empfänger während des Sendeimpulses gesperrt ist, eignet sich ein Puls-Radar nicht für geringe Entfernungen. Signale mit kurzer Laufzeit (das heißt von Objekten in geringer Entfernung) lassen sich nicht erfassen. Ein Vorteil des Puls-Radars ist, dass sich damit einfach mehrere Objekte erfassen lassen [8, S. 73-77].

2.2.2 Doppler-Radar

Ein Doppler-Radar ist im Allgemeinen ein Radar, das die technischen Voraussetzungen hat, den Dopplereffekt auszunutzen. Meist handelt es sich hierbei um ein Dauerstrich-Radar.

2.2.3 Dauerstrich-Radar

Ein Dauerstrich-Radar oder CW-Radar (Continous Wave) sendet ein unmoduliertes Sinus-Signal mit konstanter Amplitude und konstanter Frequenz aus (siehe Abbildung 2.1). Das von bewegten Hindernissen reflektierte Signal enthält neben der gesendeten Frequenz eine überlagerte Dopplerfrequenz. Sprich: Die Frequenzdifferenz zwischen gesendetem und empfangenen Signal ist die Dopplerfrequenz. Bei festen (unbewegten) Objekten tritt lediglich eine Laufzeit zwischen Sende- und Empfangssignal auf, die sich aufgrund der Periodizität der Sinusschwingung nur als Phasenunterschied messen lässt. Wie viele Perioden zusätzlich zu diesem Phasenunterschied vergangen sind, ist nicht ermittelbar, somit eignet sich ein CW-Radar nicht zur Entfernungsmessung. Zur Geschwindigkeitsmessung hingegen schon [20].

Zur Geschwindigkeitsmessung dient die Dopplerfrequenz f_d. Dabei ist zu beachten, dass der Dopplereffekt bei Radarsystemen zweimal auftritt: Auf dem Weg vom Sensor zum

reflektierenden Objekt (Signalquelle fest, Objekt bewegt) und auf dem Rückweg (Signalquelle bewegt, Radarempfänger fest). Für ein monostatisches Radar berechnet sich die Dopplerfrequenz mit:

$$f_d = \frac{2 \cdot v}{\lambda} = \frac{2 \cdot v \cdot f_0}{c} \tag{2.2}$$

dabei ist v die Radialgeschwindigkeit des reflektierenden Objektes, c die Lichtgeschwindigkeit und f_0 die Sendefrequenz [21].

Im Gegensatz zum Puls-Radar, ist es beim Dauerstrich-Radar nicht uneingeschränkt möglich mehrere Objekte zu erfassen. Dazu sind bei analogen Systemen zum Beispiel Filterbänke oder bei digitalen Lösungen FFT-Verfahren nötig. Die Entfernungsauflösung lässt sich mit

$$d_R = \frac{c}{2 \cdot B} \tag{2.3}$$

bestimmen. Dabei ist B für Puls-Radare mit i.A. ($\approx 1/t_p$) und beim Dauerstrich-Radar mit ($B \triangleq \Delta f_H$) festgelegt [8, S. 115]. Im Allgemeinen eignen sich Dauerstrich-Radare für Messungen in geringen Entfernungen [8, S. 116].

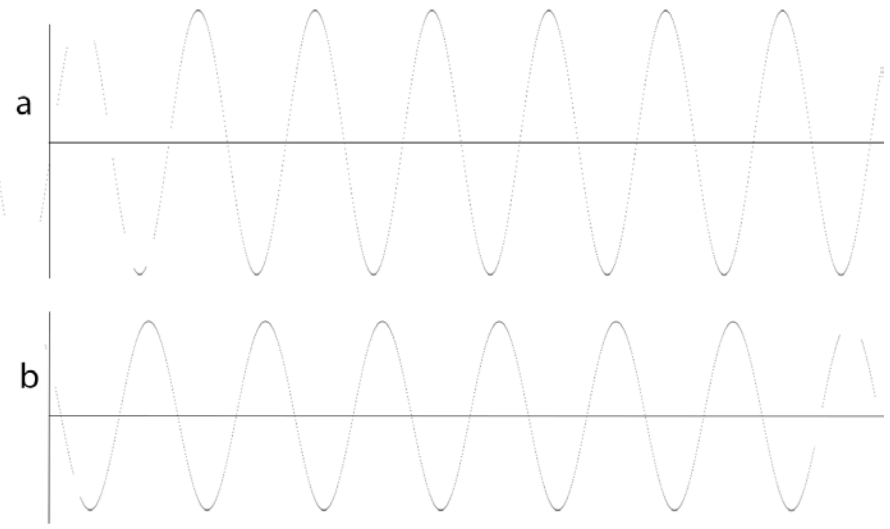

Abbildung 2.1: In Anlehnung an: [8, S. 103] Der Frequenzverlauf eines Sendesignals (a) und Empfangssignals (b)

2.2.4 FMCW-Radar

Ein FMCW-Radar (frequency modulated continous wave radar) ändert die Frequenz zeitlich periodisch. Dadurch erweitert sich der eindeutig messbare Bereich. Zudem ist die gleichzeitige Messung von Entfernung und Geschwindigkeit möglich, da die Laufzeit der Veränderung des Sendesignals messbar ist. Der Typ der Frequenzmodulation ist dabei nur von sekundärer Bedeutung [8, S. 107]. Der Verlauf von der tiefsten bis zur höchsten Frequenz wird Signalfolge genannt und kann mehrere Gigahertz betragen. Die Steilheit des Frequenzanstiegs bestimmt das Auflösungsvermögen und die Genauigkeit des Radars. Die maximale Reichweite wird durch die Dauer des Frequenzanstiegs bestimmt [22].

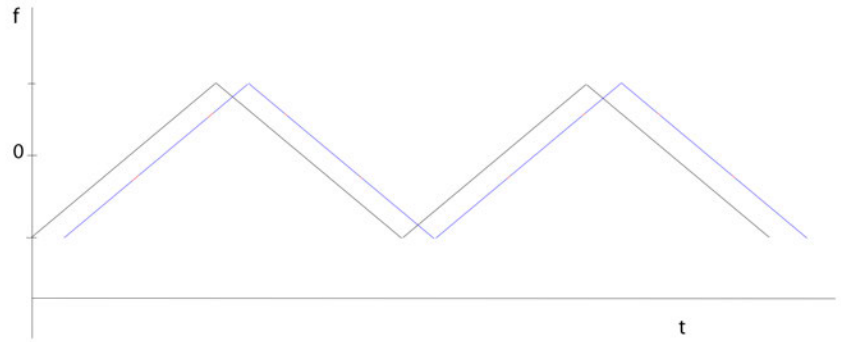

Abbildung 2.2: Frequenzverlauf eines FMCW-Signals mit dreieckiger Frequenzmodulation. In Anlehnung an: [8, S. 108]

2.3 Maschinelle Lernverfahren

Maschinelles Lernen beschreibt Computerprogramme, deren Verhalten nicht fest einprogrammiert ist. Ein solches System lernt aus Beispielen (Trainingsdaten) und kann diese nach einer Lernphase verallgemeinern und auf neue, unbekannte Daten anwenden. Das bedeutet: Ein maschinelles Lernverfahren lernt nicht auswendig, sondern erkennt Muster und Regeln in den Trainingsdaten. Bei der Erstellung eines maschinellen Lernverfahrens gibt es drei wesentliche Schritte:

1. Trainingsphase:
   Hier wird anhand der Trainingsdaten ein Modell gebildet, welches die Objekte in verschiedene Klassen einteilt. Das Modell verknüpft die Merkmale bzw. eine Merkmalskombination der Objekte aus den Trainingsdaten mit den Klassen. Dieses Modell wird auch Classifier oder Klassifikator genannt.

2. Testphase:
   Hier testet man die Performance des erstellten Classifiers. Dabei kommen je nach Verfahren verschiedene Methoden zum Einsatz. In der Regel wird hierbei mit den sogenannten Testdaten (oder Testsets) geprüft, ob das Modell wie gewünscht Objekte klassifiziert. Verschiedene Testverfahren und Parameter zur Bewertung eines Classifiers beschreibt Kapitel 3.

3. Klassifikationsphase:
   Nach erfolgreicher Testphase kommt der Classifier zum Einsatz und klassifiziert neue unbekannte Objekte.

Die Umsetzung der Lernverfahren geschieht mittels Algorithmen, die sich grob in drei Gruppen einteilen lassen: supervised learning, unsupervised learning und reinforcement learning. Krause beschreibt sie in [12, S.22] wie folgt:

Beim Unsupervised Learning (unüberwachtes Lernen) enthalten die Trainingsdaten keine Information zur Lösung. Das heißt, das System weiß nicht, was es erkennen soll. Es sucht Muster und bildet eigene Klassen für die Einteilung. Ein weitverbreiteter Ansatz des unsupervised learning ist das Clustering. Beim Clustering teilt der Algorithmus die Daten verschiedenen Stapeln (Clustern) zu.

Sollen beispielsweise Bilder von Hunden, Katzen und Mäusen erkannt werden, teilt das Verfahren die Bilder anhand von Merkmalen wie Größe, Farbe und Form ein, ohne die Klassen (Hund, Katze, Maus) zu kennen.

Beim Supervised Learning (überwachtes Lernen) hingegen lernt der Algorithmus mit Trainingsdaten, die die richtige Lösung enthalten. Das System sucht hier nur nach Merkmalen, mit denen es Elemente in vorgegebene Klassen einteilen kann. Für die Trainings- und Testdaten heißt das, dass diese entsprechende Klassen-Label enthalten. Klassische Beispiele für überwachtes Lernen sind Regression und Klassifizierung.

Beim Reinforcement Learning (bestärkendes Lernen) lernt ein sogenannter Agent selbstständig eine Strategie, um seine Belohnungen zu maximieren. Dabei wird dem Agenten

nicht die beste Strategie gezeigt, er bekommt lediglich zu bestimmten Zeitpunkten eine Belohnung, die sowohl positiv als auch negativ sein kann.

Ein wichtiger Bestandteil des maschinellen Lernens sind die Trainingsdaten und das Training. Dabei spielen die Verzerrung (Bias) und die Varianz eine wichtige Rolle, beides sollte möglichst klein gehalten werden. Verzerrung und Varianz lassen sich gut mithilfe einer Zielscheibe zeigen: Liegen die Treffer im Durchschnitt in der Mitte, das bedeutet, zu niedrige Treffer werden durch zu hohe Treffer ausgeglichen, ist die Verzerrung klein. Liegen die Treffer nah beieinander, spricht man von einer niedrigen Varianz. Abbildung 2.3 stellt die Varianz und Verzerrung grafisch dar. Dabei ist ein Ergebnis, wie es in der oberen linken Ecke zu sehen ist, in den meisten Fällen wünschenswert. Denn das Modell soll möglichst gute Vorhersagen oder Klassifikationen machen, also in die Mitte treffen.

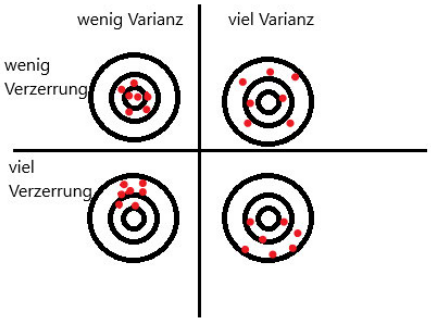

Abbildung 2.3: Verzerrung und Varianz anhand von Treffern auf einer Zielscheibe. In Anlehnung an: [10, S. 121]

Modelle mit hoher Varianz passen sich oft zu stark an die Trainingsdaten an. Das heißt, für leicht verschiedene Trainingsdatensätze sehen die Vorhersagen/ Klassifizierungen auf den Testdaten stark verschieden aus. Im maschinellen Lernen heißt diese Überanpassung Overfitting. Dies geschieht häufig, wenn der Datensatz relativ klein ist und das Modell komplex. Daher sollte das Modell so simpel wie möglich gehalten werden.

Der Gegensatz zum Overfitting ist das sogenannte Underfitting, hier bildet das Modell die Trainingsdaten zu „einfach“ ab und kann ein komplexes Problem somit nicht lösen.

Das Modell weist eine große Verzerrung auf und macht durchweg schlechte Vorhersagen/ falsche Klassifikationen.

Ein Modell mit niedriger Varianz und niedriger Verzerrung zu finden ist nicht einfach, vielmehr sind Varianz und Verzerrung voneinander abhängig. Modelle mit hoher Varianz haben eine geringe Verzerrung und umgekehrt. Daher muss ein Kompromiss zwischen Komplexität des Modells und Anpassung an die Testdaten gefunden werden [10, S. 122/123].

2.3.1 Entscheidungsbäume

Entscheidungsbäume (Decision Trees) zählen zum überwachten Lernen und folgen einem simplem Konzept: Die Klassifikation erfolgt durch eine Reihe von Entscheidungen. Entscheidungsbäume sind sehr beliebt, weil die Entscheidungen Schritt für Schritt für den Menschen nachvollziehbar bleiben. Ausgehend von der Wurzel des Baumes teilt der Algorithmus Daten in verschiedene Äste ein. Nach jeder Astgabelung erfolgt die nächste Entscheidung, bis ein sogenanntes Blatt erreicht ist. Die Blätter beschreiben das Ende des Baumes, also die zu bestimmenden Klassen.

Wichtig beim Entscheidungsbaum ist, wann welche Entscheidungen getroffen werden. Die Trainingsphase entscheidet nach welchen Kriterien (Ästen) geteilt wird und in welcher Reihenfolge [11, S. 113-117].

2.3.2 Random Forest

Random Forest ist ein flexibles maschinelles Lernverfahren. Es lässt sich einfach für Regressions- und Klassifizierungsprobleme einsetzen und ist deshalb auch einer der am häufigsten verwendete Algorithmen. Random Forest zählt zu den überwachten Lernverfahren. Wie der Name verrät, basiert das Verfahren auf einem Wald, der zufällig erstellt wird. Dieser Wald ist ein Ensemble aus Entscheidungsbäumen (Decision Trees) die meist nach dem sogenannten Bagging-Verfahren trainiert werden.

Bagging stammt von der englischen Bezeichnung Bootstrap Aggregating. Bootstrapping geht auf die Statistik zurück und beschreibt ein Verfahren zum Resampling von Daten mit Zurücklegen. Zur Erinnerung: Beim ziehen einer Kugel aus einer Urne mit vier blauen und zwei roten Kugeln, liegt die Wahrscheinlichkeit, eine blaue Kugel zu ziehen bei 2/3.

In der Annahme eine blaue Kugel gezogen zu haben, steigt die Wahrscheinlichkeit, eine rote zu ziehen – sofern die gezogene Kugel nicht wieder in die Urne zurückgelegt wird. Legt man die Kugel zurück, bleibt die Wahrscheinlichkeit allerdings gleich.

Beim Bagging werden nun Datensätze auf diese Art generiert. Aus einem Trainingsset D mit n Datensätzen, generiert das Verfahren m neue Mengen D_i mit der Größe n'. Die Entnahme aus D verläuft dabei uniform, sprich mit gleicher Wahrscheinlichkeit für jede Probe – also mit zurücklegen. Dadurch besteht die Möglichkeit, dass einige Datensätze in D_i doppelt sind. Für den Fall $n = n'$ und eine hinreichend große Datenmenge lässt sich zeigen, dass D_i einen Anteil von $1-1/e$ der ursprünglichen Daten enthält. Dies entspricht circa 63 Prozent, der restliche Anteil von etwa 37 Prozent sind Mehrfachziehungen. Kurz zusammengefasst bedeutet das: Man nimmt einen Datensatz, löscht einen Teil der Informationen und ersetzt sie durch Duplikate. So werden Lerner (Entscheidungsbäume) erzeugt, die sich leicht verschieden auf demselben Problem verhalten. Die Ergebnisse der Lerner werden dann kombiniert. Vereinfacht gesagt: Random Forest erzeugt mehrere Entscheidungsbäume und fügt sie zusammen [7, S. 154/155].

2.3.3 Support Vector Machines

Support Vector Machines (SVM) sind keine Maschinen im physischen Sinn, sondern Algorithmen, die eine Trennlinie zwischen Klassen finden [1, S. 96]. SVMs dienen als Klassifikator und Regressor und zählen zu den sogenannten Large Margin Classifiers. Sie unterteilen eine Menge von Elementen in zwei Klassen, sodass um die Klassengrenzen herum ein möglichst großer Bereich frei von Elementen bleibt. Durch Optimierung des Abstands zwischen den Elementen und der Trennlinie erzielen SVMs gute Ergebnisse. Für die Berechnung der Trennlinie braucht die SVM nur drei Datenpunkte (die sogenannten Stützvektoren). Dadurch wird die Trennlinie definiert. Bei zwei verschiedenen Klassen braucht man zwei Punkte aus der einen Klasse, weil diese eindeutig eine Gerade festlegen sowie einen Datenpunkt aus der anderen Klasse, der nur den Abstand für die zweite parallele Gerade bestimmt. Abbildung 2.4 zeigt ein Beispiel einer SVM, die zwei Punktwolken (Klassen) trennt. Dabei stellt die rote Linie die Trennungslinie dar, die den größten Abstand zu den beiden Klassen aufweist. Zur Berechnung dienten hier beiden grauen Linien.

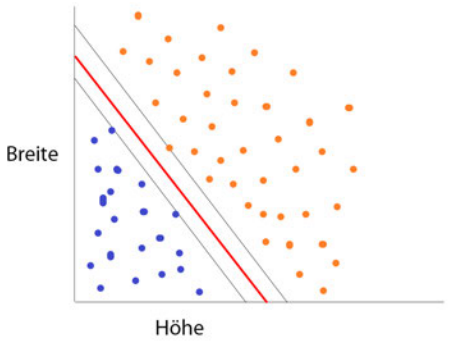

Abbildung 2.4: Bestimmung der Trennlinie einer SVM. In Anlehnung an: [1, S. 98]

Allerdings lassen sich die meisten Probleme nicht mit einer Geraden in zwei Klassen trennen. Doch mit dem sogenannten Kernel-Trick lässt sich eine SVM auch hier einsetzen. Dabei wird ein nicht lineares Problem in ein anderes Koordinatensystem übertragen. Dazu müssen die Achsen des Koordinatensystems verdreht und verbogen werden. Dieses neue Koordinatensystem wird auch Merkmalsraum genannt. Im Merkmalsraum kann die SVM nun die Klassen mit einer Gerade teilen. Anschließend wird alles wieder zurück übertragen und man erhält im ursprünglichen Koordinatensystem beispielsweise eine gekrümmte Kurve [1, S. 101-103]. Ein Nachteil von SVM ist, dass das Verfahren nicht sonderlich gut skaliert und sich oftmals nicht für größere Datensets eignet [7, S. 286].

2.3.4 Multilayer Perceptron

Ein Perzeptron (vom englischen Perception, Wahrnehmung) ist ein einfaches künstliches neuronales Netz. Es besteht in der einfachsten Variante aus einem einzelnen künstlichen Neuron (einfaches Perzeptron). Dieses besitzt anpassbare Gewichtungen und einen Schwellenwert. Man unterscheidet generell zwischen einlagigen und mehrlagigen Perzeptren (englisch Multilayer Perceptron, kurz MLP). Perzeptron-Netze überführen einen Eingabevektor in einen Ausgabevektor. Ein Perzeptron lässt sich bei linear separierbaren Klassen einsetzen und stellt eine Funktion $P : \mathbb{R} \rightarrow \{0,1\}$ dar. Hierbei rechnet es die Inputs $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}$ in Outputs O_i um. Dabei bestimmen die Gewich-

tungsfaktoren $w = (w_1, w_2, \ldots, w_n) \in \mathbb{R}$ die Relevanz der einzelnen Inputs für den Output.

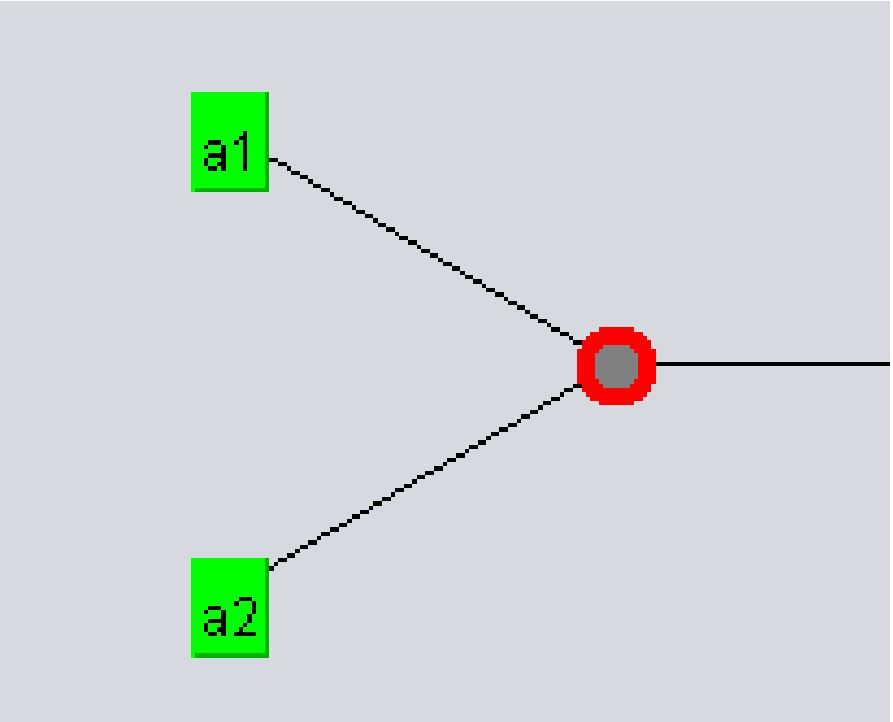

Abbildung 2.5: Einfaches Perzeptron mit zwei Eingabewerten

Der Output ist die gewichteten Summe über die Inputs und folgt folgender Regel:

$$P(x) = \begin{cases} 1 & \text{falls } w_x = \sum_{i=1}^{N} w_i \cdot x_i > 0 \\ 0 & \text{sonst} \end{cases} \tag{2.4}$$

[5, S.201]

An der Formel $\sum_{i=1}^{N} w_i \cdot x_i > 0$ ist zu erkennen, dass alle Punkte über der trennenden Hyperebene $\sum_{i=1}^{N} w_i \cdot x_i = 0$ als positiv ($P(x) = 1$) und alle anderen als negativ ($P(x) = 0$) klassifiziert werden. Sei M_+ die Menge der positiven und M_- die Menge der negativen Trainingsmuster, dann lautet die Perzeptron-Lernregel:

w = beliebiger Vektor reeller Zahlen

Repeat
    For all $x \in M_+$
        If wx <= 0 Then w = w + x
    For all $x \in M-$
        If wx > 0 Then w = w – x
Until alle $x \in M_+ U M_-$ werden korrekt Klassifiziert
[5, S.202]

Ziel ist es, dass das Perzeptron für alle $x \in M+$ den Wert 1 ausgibt. Laut Definition 2.4 muss dafür $wx > 0$ sein. Ist dies nicht so, addiert der Algorithmus x zum Gewichtungsfaktor w, somit verschiebt sich der Gewichtungsfaktor in die richtige Richtung. Dies ist zu erkennen, wenn das Perzeptron auf den geänderten Vektor $w + x$ angewendet wird, denn

$$(w + x) \cdot x = wx + x^2$$

Wiederholt man dieses Verfahren oft genug, wird der Wert wx irgendwann – wie gewünscht – positiv. Dies gilt analog für die negativen Trainingsdaten, für die das Perzeptron immer kleiner werdende Werte

$$(w - x) \cdot x = wx - x^2$$

berechnet, die irgendwann negativ werden [5, S.202].

Durch Hinzufügen einer oder mehrerer weiterer Schichten, den sogenannten verdeckten Schichten (Hidden Layer), wird die Mächtigkeit des Perzeptron-Netzes erhöht. So lassen sich auch nicht linear separierbare Probleme lösen.

Netzwerke in denen die Ausgänge der Neuronen einer Schicht nur mit den Eingängen einer nachfolgenden Schicht verbunden sind, sodass Informationen nur in einer Richtung fließen, nennt man Feed-Forward-Netze. Sind dabei die Neuronen einer Schicht mit allen Neuronen der direkt folgenden Schicht verknüpft, spricht man von „fully connected“. Haben einige Neuronen nicht nur eine Verbindung zur nächsten Schicht, sondern auch zu einer weiteren, spricht man von Short-Cuts. Gibt es in dem Netz hingegen Neuronen, die mit Neuronen aus der gleichen Schicht oder Neuronen einer vorherigen Schicht verbunden sind, liegt ein rekurrentes neuronales Netz (RNN) vor.

Für Multilayer Perceptrons sind komplexere Lernregeln als für einlagige Perceptrons nötig. Ein weitverbreiteter Algorithmus ist die Backpropagation. Grund dafür ist die universelle Einsetzbarkeit für beliebige Aufgaben [6, S. 291]. Beim Backpropagation-Algorithmus wird als Aktivierungsfunktion die Sigmoid-Funktion auf die gewichtete Summe der Eingaben angewendet. Typischerweise besteht ein Backpropagation-Netz aus drei Schichten: der Eingabeschicht, einer verdeckten Schicht und der Ausgabeschicht. Folgende Abbildung zeigt ein solches dreilagiges Netz.

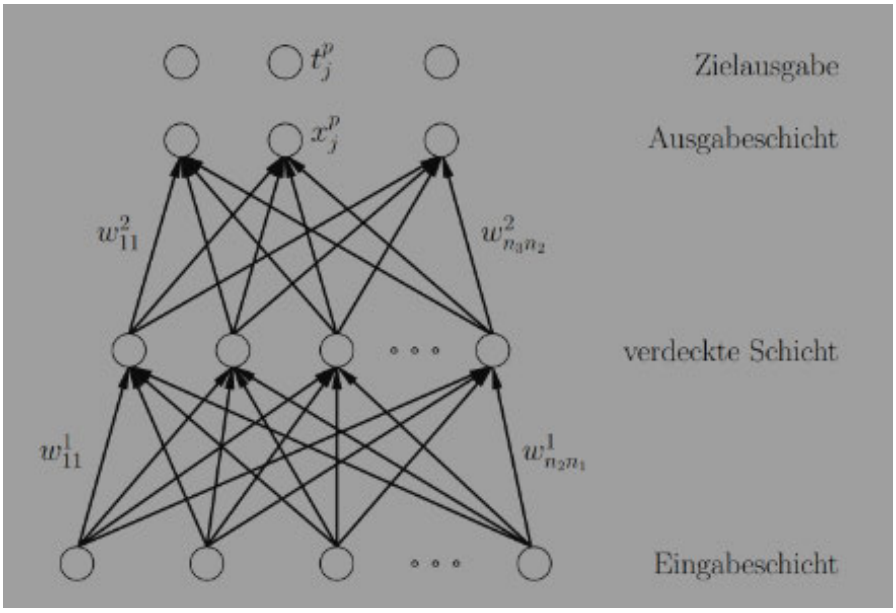

Abbildung 2.6: Ein dreilagiges Backpropagation-Netz [6, S. 291]

Dabei sind die Ausgabewerte X_j^p parallel zu den Zielausgabewerten t_j^p eingezeichnet, da diese miteinander verglichen werden. Bis auf die Eingabeneuronen berechnen alle Neuronen ihren aktuellen Wert x_j nach der Regel:

$$x_j = f(\sum_{i=1}^{N} w_{ji} \cdot x_i)$$

mit der Sigmoid-Funktion

$$f(x) = \frac{1}{1 + e^{-1}}$$

Dabei werden die Gewichte entsprechend dem negativen Gradientverfahren der über die Ausgabeneuronen aufsummierten quadratischen Fehlerfunktion

$$E_p(w) = \frac{1}{2} \sum_{k \in Ausgabe} (t_k^p - x_k^p)^2$$

für das Trainingsmuster p geändert:

$$\Delta p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$$

Der nachfolgende Pseudocode zeigt den Lernablauf des Backpropagation-Algorithmus. Zuerst berechnet das Modell die Ausgabe (Vorwärtspropagieren) für ein Trainingsbeispiel und bestimmt den Approximationsfehler. Diesen nutzt das Verfahren dann, um rückwärts die Gewichte – Schicht für Schicht – zu ändern (Rückwärtspropagieren). Diesen Prozess wiederholt der Algorithmus solange, bis sich die Gewichte nicht mehr ändern oder eine vorgegebene Zeitschranke erreicht ist.

Repeat
    For all $(q^p, t^p) \in$ Trainingsbeispiele
        1. Anlegen des Vektors q^p an die Eingabeschicht
        2. Vorwärtspropagieren:
        Für alle Schichten ab der ersten verdeckten Schicht aufwärts, für alle Neuronen
        Berechne die Aktivierungsfunktion
        3. Quadratischen Fehler berechnen
        4. Rückwärtspropagieren:
        Abwärts für jedes Gewicht von der letzten Schicht aus
Until w konvergiert oder Zeit abgelaufen
[6, In Anlehnung an S. 293]

Netze mit mindestens einem Hidden Layer sind in der Lage, nicht lineare Abbildungen zu lernen. Ohne einen Hidden Layer sind die Ausgabeneuronen trotz der Sigmoidfunktion nicht mächtiger als ein lineares Neuron. Dies ist in der strengen Monotonie der Sigmoidfunktion begründet. Gleiches gilt auch für mehrlagige Netze mit linearen Aktivierungsfunktionen. Grund hierfür ist die Linearität des hintereinander Ausführens linearer Abbildungen [6, S. 293].

2.4 Datenverarbeitung

Eine Datenvorverarbeitung im eigentlichen Sinn durch Fensterfunktionen, Faltungen (FFT) oder Normalisierungen wird in dieser Arbeit nicht vorgenommen. Es wird mit den sogenannten Rohdaten des Radars gearbeitet. Zur Verbesserung der Ergebnisse werden ähnlich wie im Original-Experiment, welches in Kapitel 4 vorgestellt wird, lediglich einige Extra-Features berechnet. Hierzu zählen: der Mittelwert, das Minimum, das Maximum und das Root Mean Square (RMS).

- Mittelwert: In dieser Arbeit wird der arithmetische Mittelwert, also die Summe aller Messwerte geteilt durch die Anzahl, benutzt.
- Minimum beschreibt das globale Minimum, den kleinsten Messwert eines Datenpunktes.
- Maximum beschreibt das globale Maximum, den größten Messwert eines Datenpunktes
- RMS ist das quadratische Mittel, sprich die Quadratwurzel des Quotienten aus der Summe der Quadrate der Messwerte und deren Anzahl.

3 Evaluation der maschinellen Lernverfahren

Hinweis: Die Idee und Struktur dieses Kapitels entstammt der Bachelorarbeit "Evaluation von Data-Mining-Algorithmen am Beispiel einer Anwendung zur Parkplatzsuche" von Moritz Knüppel [9].

In dieser Arbeit werden Klassifikationsverfahren eingesetzt und unter verschiedenen Gesichtspunkten analysiert. In diesem Kapitel werden Parameter und Methoden zur Evaluation solcher Verfahren vorgestellt. Wie in Abschnitt 2.3 beschrieben, neigen maschinelle Lernverfahren zu Overfitting beziehungsweise Underfitting. Beide Effekte lassen sich durch das Datenset sowie das Trainingsverfahren beeinflussen.

3.1 Testdaten bereitstellen

Eine gängige Methode ist es, zwei separate Datensets zu erstellen: einerseits die Trainings- und andererseits die Testdaten. Dabei ist es wichtig, dass beide Datensets unter gleichen Bedingungen erstellt wurden und beide Datensets sämtliche zu bestimmenden Klassen enthalten. Steht nur ein Datenset zur Verfügung, wird dieses einfach in zwei disjunkte Teilmengen (Trainings- und Testdaten) geteilt. Diese Methode wird auch Hold-Out genannt, weil dem Classifier ein Teil der Trainingsdaten vorenthalten und später zum Testen genutzt wird. Meist geschieht diese Aufteilung nach einem bestimmten Prozentsatz, dem sogenannten Percentage-Split. Hierbei wird ein bestimmter Anteil des gesamten Datensatzes zum Training ausgewählt. Dabei ist es wichtig, auf eine ausgewogene Verteilung der Klassen zu achten. Dazu wird der Datensatz meist zufällig gemischt. Die Gefahr dabei ist allerdings, dass die Ergebnisse eines auf diese Weise erstellten Modells von den zufälligen Datensets abhängen. So besteht die Möglichkeit, dass einige Klassen nicht in den Trainingsdaten auftauchen und das Modell sie nicht abbildet. Dies gilt vor allem für relativ kleine Datensets. Eine feste Regel für die Aufteilung beim Percentage-Split gibt

es nicht. Eine gute Aufteilung ist unter anderem abhängig von der Größe der Datensets und der Komplexität des Modells. Typischerweise nutzt man zwischen 60% und 80% der Daten als Trainingsdaten und somit zwischen 40% und 20% zum Testen[4].

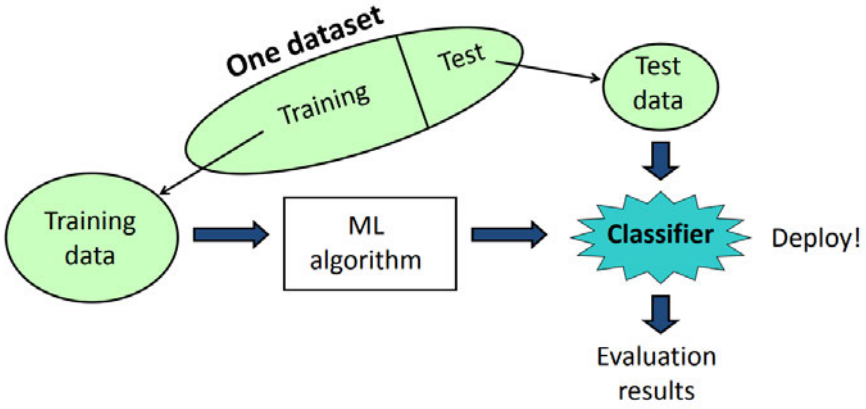

Abbildung 3.1: Funktionsweise des Hold-Outs [17, S.8]

3.2 Cross-Validation

Die Cross-Validation (Kreuzvalidierung) ist ein systematisches Evaluierungsverfahren bei dem der Datensatz in möglichst gleichgroße Teilstücke k (folds) zerlegt wird. Eines dieser Stücke wird fürs Testen zurückgelegt, die übrigen Teilstücke dienen als Trainingsdaten. Die Anzahl der folds bestimmt die Anzahl der Testdurchläufe. Für eine 10-fold-Cross-Validation bedeutet dies, dass der Datensatz in zehn Stücke zerteilt wird, neun dieser Stücke werden zum Training benutzt, eins zum Testen. Dabei dient jedes der zehn Stücke einmal als Testmenge. Aus den zehn einzelnen Testergebnissen lässt sich abschließend ein durchschnittliches Ergebnis bilden.

Eine besondere Form ist die stratified Cross-Validation. Hier wird bei der Zerteilung auf eine korrekte Proportion der Klassen in jedem fold geachtet. Die Cross-Validation ist eine gute Option für kleinere Datensätze. Dadurch, dass alle Datenpunkte fürs Training und Testen verwendet werden, wird die Gefahr, dass wichtige Daten nicht zum Training genutzt werden, reduziert. Dies könnte dazu führen, dass das Model nicht ausreichend generalisiert. Im Vergleich zum einfachen Hold-Out-Verfahren ist die Cross-Validation meist das bessere Verfahren, da sie Ergebnisse mit weniger Varianz liefert [19]. Allerdings ist die Cross-Validation aufwendiger, da k Modelle erstellt werden und es k Testdurchläufe

gibt. Deshalb wird bei sehr großen Datensets (mehr als 1.000 Datenpunkte bei einem Zwei-Klassen-Problem), bei denen das normale Hold-Out gute Ergebnisse liefert, die Cross-Validation nicht verwendet. Abbildung 3.2 zeigt die Cross-Validation in Weka. Hier gibt es eine Besonderheit: Weka führt den Algorithmus zum Abschluss ein elftes Mal mit dem gesamten Datenset aus.

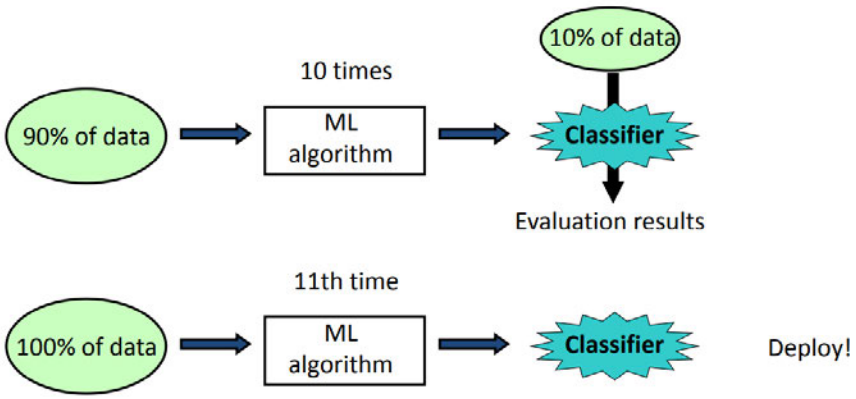

Abbildung 3.2: Cross-Validation in Weka [17, S.26]

3.3 Confusion Matrix

Ein Werkzeug zur Bewertung von Klassifikatoren ist die Wahrheitsmatrix, auch Confusion Matrix genannt. In ihr werden die korrekt und inkorrekt klassifizierten Objekte systematisch dargestellt. Die Anzahl der Klassen bestimmt die Ordnung der quadratische Matrix. Für ein Zwei-Klassen-Problem besteht die Matrix aus zwei Zeilen und zwei Spalten. Die Spalten geben die Anzahl der einer Klasse zugeordneten Elemente an, die Zeilen die wirkliche Klasse. Bei dieser Anordnung stehen auf der Hauptdiagonalen die korrekt klassifizierten Elemente und in den übrigen Feldern die falsch klassifizierten.

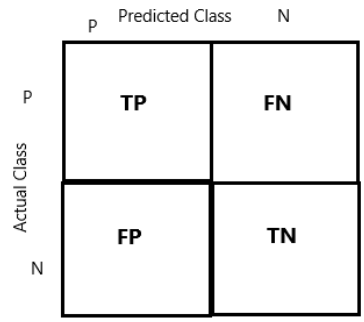

Abbildung 3.3: Eigene Darstellung: Confusion Matrix für ein Zwei-Klassen-Problem

Im Beispiel in Abbildung 3.3 existieren die Klassen P und N. Das linke obere Feld der 2x2-Matrix beschreibt nun diejenigen Elemente, die der Klasse P zugeordnet wurden und tatsächlich zur Klasse P gehören. Dies sind die sogenannten True Positives (TP). Im linken unteren Feld stehen die Elemente, die der Classifier fälschlicherweise der Klasse P zugeordnet hat. Dies sind die „False Positives“ (FP). Oben rechts stehen die „False Negatives“ (FN), also die Elemente, die der Classifier der Klasse N zugeordnet hat, obwohl sie zur Klasse P gehören. Unten rechts sind die „True Negatives“ (TN) also Elemente, die der Algorithmus richtigerweise als Klasse N klassifiziert hat.

Die Confusion Matrix ist ein gutes Werkzeug, um das Ergebnis eines Classifiers übersichtlich darzustellen. Daraus lässt sich schnell erkennen, an welchen Stellen ein Algorithmus Fehler macht beziehungsweise ob er Elemente einer bestimmte Klasse besonders häufig falsch klassifiziert. Die Confusion Matrix lässt sich auch für Probleme mit mehr als zwei Klassen einsetzen. Außerdem lassen sich anhand der Confusion Matrix einfach weitere Werte bestimmen, dazu gehören wie in [16] beschrieben:

- Die „True-Positive-Rate“ (TPR) auch Sensitivity oder Recall genannt, sie gibt das Verhältnis zwischen den korrekt als positiv klassifizierten Objekten und allen tatsächlich positiven Elementen an.

$$TPR = TP/P = \frac{TP}{TP + FN}$$

- Die „False-Negative-Rate“ (FNR) gibt das Verhältnis zwischen den fälschlich als negativ klassifizierten Objekten und sämtlichen wirklichen positiven Elementen an.

$$FNR = FN/P = \frac{FN}{TP + FN}$$

- Die „True-Negative-Rate“ (TNR) auch Specificity genannt, gibt das Verhältnis der korrekt als negativ erkannten Objekte an der Gesamtheit der negativen Objekte an.

$$TNR = TN/N = \frac{TN}{TN + FP}$$

- Die „False-Positive-Rate“ (FPR), welche den Anteil der fälschlich als positiv klassifizierten Objekte an den tatsächlich negativen Objekten angibt.

$$FPR = FP/N = \frac{FP}{TN + FP}$$

- Die „Accuracy“ (ACC), also den Anteil der korrekt klassifizierten Objekte.

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + FP + TN + FN}$$

- Der „Positiv-Prediction-Value“ (PPV), auch Precision genannt, gibt den Anteil der richtigerweise als positiv klassifizierten Objekte an den als positiv klassifizierten Objekten an.

$$PPV = \frac{TP}{TP + FP}$$

- Die „Fehlklassifikationsrate“ (EER), auch Error Rate genannt, also den Anteil der falsch klassifizierten Elemente.

$$EER = \frac{FP + FN}{TP + FP + TN + FN}$$

3.4 Matthews Correlation Coefficient

Der Matthews Correlation Coefficient, kurz MCC ist ein Maß zur Bewertung, wie gut ein Classifier-Modell funktioniert. Der berechnete Wert des MCC liegt zwischen -1 und 1, wobei -1 einen komplett falschen Classifier beschreibt, der alle Elemente falsch klassifiziert. Der Wert 1 hingegen indiziert ein komplett korrektes Modell, ein MCC von 0 bedeutet: Der Classifier ist nicht besser als zufälliges Wählen. Der MCC gilt als ausgewogenes Maß und lässt sich auch für unausgewogene Datensets, das heißt für Datensets mit deutlich unterschiedlich großen Klassen, nutzen. Für ein Zwei-Klassen-Problem, wie in Abbildung 3.4 dargestellt, wird der MCC mit:

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

berechnet. Für den Fall, dass einer der Werte: TP + FP, FN + TN, FP + TN oder TP + FN Null ist, ist der MCC nicht definiert [14].

Die Aussagekraft des MCC lässt sich anhand einiger Beispiele verdeutlichen. Gegeben sei folgende Confusion Matrix als Ergebnis eines Classifiers:

	P	N				
P	0	50				
N	0	350				
			Precison	Recall	Accuracy	MCC
			0	0	0.875	0

Abbildung 3.4: MCC im Vergleich zur Accuracy

In diesem Fall hat das Modell einfach alle Objekte der Klasse N zugeordnet. Dabei lag es 350-mal richtig und in 50 Fällen falsch. Betrachtet man für dieses Modell nur die Accuracy, könnte man durch den Wert von 0,875 auf ein relativ gutes Modell schließen. Der MCC ist in diesem Fall eigentlich nicht definiert, da aber auch der Zähler 0 ist, beträgt der MCC hier 0.

	P	N				
P	50	0				
N	350	0				
			Precison	Recall	Accuracy	MCC
			0.125	1	0.125	0

Abbildung 3.5: MCC im Vergleich zum Recall

Die Abbildung 3.5 zeigt ein Modell, welches sämtliche Objekt der Klasse P zuordnet. Es klassifiziert also 50 Objekte korrekt und 350 falsch. Bewertet man den Classifier anhand des Recalls, gelangt man zu einem perfekten Ergebnis von 1. Auch hier beträgt der MCC 0 und lässt auf ein schlechtes Modell schließen.

	P	N				
P	325	50				
N	20	5				
			Precison	Recall	Accuracy	MCC
			0.942028986	0.866666667	0.825	0.04686013

Abbildung 3.6: MCC nicht Null im Vergleich zum Recall und Accuracy

Das Beispiel in Abbildung 3.6 zeigt die Confusion Matrix eines weiteren unausgewogenen Datensatzes. Dieses Modell klassifiziert die Objekte aus der Klasse P gut, und Objekte aus der Klasse N schlecht. Betrachtet man für dieses Beispiel nur die Werte Precision und Recall, erscheint der Classifier gut, weil die Anzahl der Objekte in Klasse P deutlich größer ist als die Anzahl der Objekte der Klasse N. Der MCC bezieht diesen Umstand mit ein und liefert eine Bewertung des Modells, welche die Gesamtzuverlässigkeit besser darstellt.

3.5 ROC-Kurve und AUC

Viele Classifier sind nicht nur fähig, Objekte zu klassifizieren, sondern liefern gleichzeitig einen Wert, wie sicher sie sich bei der Zuordnung sind. Dies ist die sogenannte predicted probability. Sie wird anhand des sogenannten Threshold bestimmt. Der Threshold ist der Grenzwert, anhand dessen der Classifier entscheidet, ob ein Objekt zu einer Klasse gehört oder nicht. Bei einem binären Classifier mit den Klassen-Labeln P und N, normalisierten predicted probabilities und einem Threshold von 0,5, heißt das: Alle Objekte mit einem Wert kleiner als 0,5 werden der Klasse N zugeordnet und alle Objekt mit dem Wert 0,5 oder höher werden als Klasse P klassifiziert. Je weiter die Werte vom Threshold entfernt sind, desto sicherer ist sich der Classifier (desto größer ist die predictet probability). Eine Möglichkeit ein Modell zu verbessern, ist es den Threshold zu verändern, so lässt sich beispielsweise das Verhältnis zwischen False Positives und False Negatives angleichen.

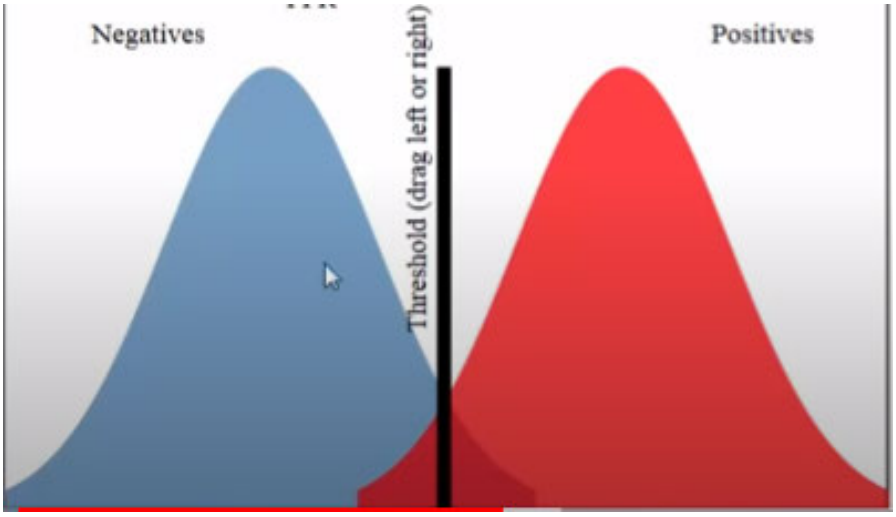

Abbildung 3.7: Verteilung der Objekte zweier Klassen. Blau die Objekte der Klasse Negative (N) und rot die Objekte der Klasse Positive (P). Der schwarze Balken ist der Threshold, die X-Achse zeigt die predicted probability, die Y-Achse die Anzahl der Objekte[15]

.

Die Abbildung 3.7 zeigt einen Classifier, der zwei Klassen relativ gut voneinander trennt. Bei einem Threshold von 0,5 klassifiziert er den Großteil der Objekte richtig, gleichzeitig ist das Verhältnis zwischen False Positives und False Negatives ausgeglichen. Durch Verschieben des Thresholds auf 0,4 erhöht sich der Anteil der False Positives, während die Anzahl der False Negatives auf null sinkt. Setzt man den Threshold auf 0,6, erhält man hingegen einen Classifier, der keine False Positives aufweist. Allerdings steigt hier der Anteil der False Negatives.

Die sogenannte Receiver Operating Characteristic Curve zu Deutsch „ROC-Kurve“ visualisiert dieses Verhalten. Sie stellt in einem Diagramm das Verhältnis zwischen der False-Positive-Rate auf der X-Achse und der True-Positive-Rate auf der Y-Achse für verschiedene Thresholds dar. Die Kurve beginnt immer im Punkt $(0, 0)$ und endet im Punkt $(1, 1)$. Die ROC-Kurven verschiedener Modelle lassen sich direkt oder für unterschiedliche Thresholds vergleichen und liefern ein gutes Bild über die Performance eines Classifiers. Dabei gibt die Form der Kurve wichtige Informationen. So indizieren kleinere Werte auf der X-Achse weniger False Positives und mehr True Negatives, während größere Werte auf der Y-Achse mehr True Positives und weniger False Negatives bedeuten [3].

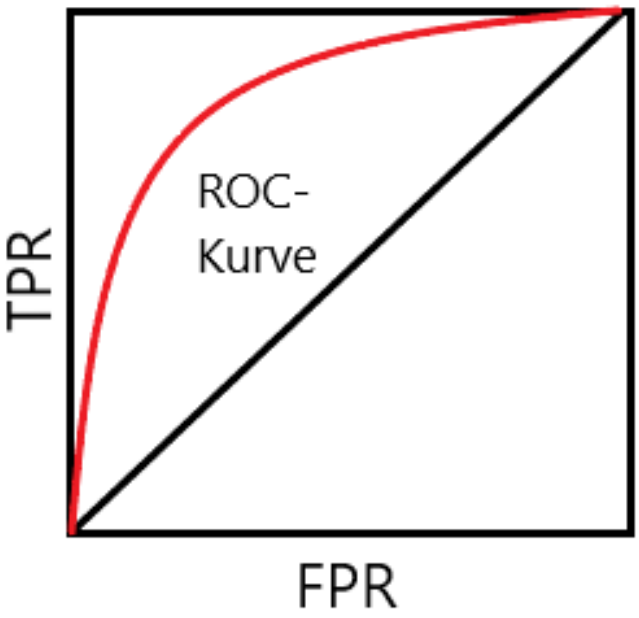

Abbildung 3.8: Eigene Darstellung: ROC-Kurve

Interpretation der ROC-Kurve: Eine ROC-Kurve die möglichst weit in der oberen linken Ecke des Diagramms verläuft, repräsentiert einen guten Classifier. Im Optimalfall verläuft die Kurve durch den Punkt (0, 1). Es lässt sich also ein Threshold wählen, bei dem das Modell alle Objekte korrekt klassifiziert. Liegen die Werte nahe der Diagonalen von links unten nach rechts oben, deutet dies auf ein Modell hin, welches nicht viel besser als der Zufall ist. Bleibt die Kurve deutlich unterhalb der Diagonalen, spricht dies für eine falsche Interpretation der Werte.

Ein weiterer Wert, der sich mit der ROC-Kurve bestimmen lässt, ist die „area under the curve“ (AUC) also die Fläche unter der ROC-Kurve. Der AUC quantifiziert die Classifier-Performance. Je näher die ROC-Kurve am Punkt (0, 1) verläuft, desto größer ist die Fläche unter der Kurve. Theoretisch liegt der AUC zwischen 0 und 1, praktisch zwischen 0,5 und 1, denn ein Wert unterhalb von 0,5 bedeutet die ROC-Kurve liegt unterhalb der Diagonalen, sprich der Classifier ist schlechter als zufälliges Raten. Ein AUC von 1 beschreibt ein perfektes Modell.

3.6 Precision-Recall-Curve

Die Precision-Recall-Curve (Precision-Recall-Kurve; PCR-Kurve) ist ein weiteres Maß zur Bewertung maschineller Lernverfahren. Für Modelle mit unausgewogenen Datensets ist es sinnvoll die Werte für Precision und Recall zu betrachten. Vor allem bei Modellen, in denen es viele Objekte in der einen Klasse (N) und wenige in der anderen Klasse (P) gibt. Denn im Falle von vielen Objekten in Klasse N (also vielen True Negatives), ist die Fähigkeit, diese zu erkennen, typischerweise weniger interessant als die Fähigkeit, Objekte der Klasse P zu erkennen. Sowohl bei der Berechnung der Precision als auch des Recalls, spielen die True Negatives keine Rolle. Somit wird der Fokus auf die korrekt klassifizierten Objekte der Minderheitsklasse gelegt. Die Precision-Recall-Curve stellt die Precision auf der Y-Achse und den Recall auf der X-Achse grafisch dar – ähnlich wie die ROC-Kurve für verschiedene Thresholds.

Die PCR-Kurve eines schlechten Classifiers ist dabei – für ein ausgewogenes Datenset – eine horizontale Linie. Diese wird bestimmt durch das Verhältnis y der positiven Objekte P zu den Negativen N mit:

$$y = \frac{P}{(P + N)}$$

Für einen ausgewogenen Datensatz ist y = 0,5. Ein perfekter Classifier liegt im Diagramm im Punkt (1, 1), demnach repräsentiert eine PCR-Kurve, die möglichst nah am Punkt (1, 1) verläuft ein gutes Modell. Abbildung 3.9 zeigt eine typische PCR-Kurve eines guten (orange) und eines schlechten Classifiers (blau).

Im Vergleich zur ROC-Kurve eignet sich die PRC-Kurve besser für Modelle mit unausgewogenen Datensets [3].

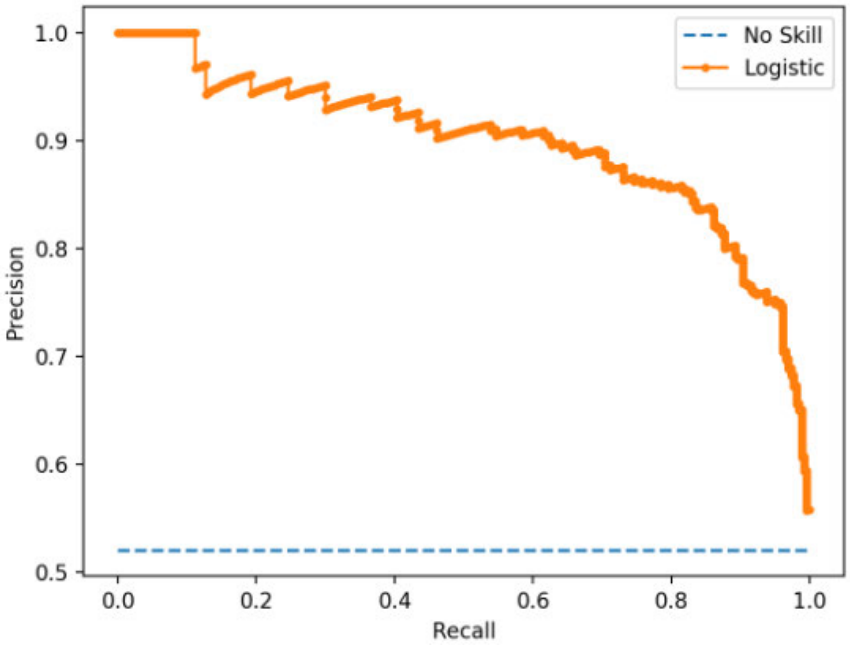

Abbildung 3.9: Beispiel: PCR-Kurve [3]

4 Das Radarcat-Paper

Dieses Kapitel stellt das in dieser Arbeit nachempfundene Experiment aus dem Paper „RadarCat: Radar Categorization for Input & Interaction“ vor [23].

Das Paper präsentiert ein kleines, vielseitiges Radar-basiertes System zur Material und Objektklassifizierung, welches neue Formen der Interaktion mit digitalen Devices ermöglicht. Zur Klassifizierung nutzt das System die Roh-Daten eines Multichannel-Radars, dem Project-Soli-Sensor. Der Sensor ist ein monostatisches Radar mit zwei Sende- und vier Empfangsantennen, welche die gleichzeitige Erfassung von acht Kanälen ermöglichen. Diese Multichannel-Radardaten bieten eine hohe Charakteristik, wenn alltägliche Objekte sie reflektieren. So unterscheiden sich die Signale unterschiedlicher Materialien, unterschiedlich dicker Objekte und von Dingen mit verschiedenen Formen. Da diese die Signale unterschiedlich stark absorbieren und reflektieren. Die reflektierten Signale enthalten Reflexionen von der Oberfläche des Objektes, der internen Struktur des Materials und der Rückseite. Dabei spielen auch eine Reihe physikalischer Größen wie die Dichte des Materials eine Rolle. In Abbildung 4.1 sind gemessene Signale unterschiedlicher Objekte zu sehen. Dabei weisen die unterschiedlichen Materialien verschiedene Charakteristiken auf. Dargestellt sind (von links nach rechts) Aluminium, Kupfer, Stahl, die Frontseite eines Nexus 5 sowie die Rückseite desselben Smartphones.

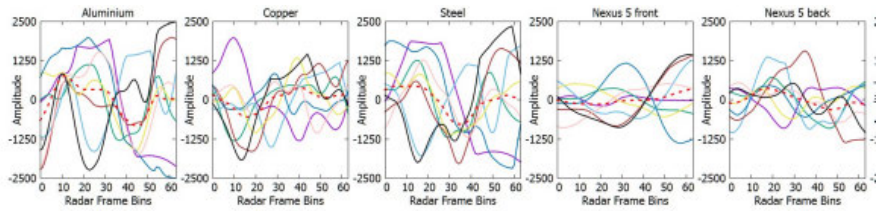

Abbildung 4.1: Gemessene Signale unterschiedlicher Objekte [23]

Ziel des RadarCat-Projekts war es, die Limitationen eines Kamera-basierten Systems zu übertreffen. Das eingebettete System sollte schnell und mit hoher Genauigkeit Objekte

und Materialien identifizieren. Die geringe und gleichbleibende Distanz zwischen Messobjekt und Sensor ermöglicht dabei eine genaue Klassifizierung. Der Soli-Sensor stellt ein FMCW-Radar dar und nutzt Frequenzen in dem Bereich von 57 bis 64 GHz (Mittenfrequenz 60 GHz). Die Messobjekte wurden direkt auf dem Sensor platziert, er wurde also wie eine Art Stethoskop verwendet. Zur Klassifikation dienten in diesem Experiment die Messdaten aller acht Kanäle mit jeweils 64 Messpunkten, was im ersten Schritt zu 512 Features (Attributen) führte. Zusätzlich berechnete man weitere Features, unter anderem den Durchschnitt, den Absolutwert, globale Minima, globale Maxima sowie den Root Mean Square (RMS) für jeden Kanal, daraus ergibt sich eine Anzahl von 661 Features. Der Classifier basiert auf Weka, als maschinelles Lernverfahren wird hier der Random Forest benutzt, da er gegenüber anderen Verfahren wie der SVM leichte Performance-Vorteile bot. Zudem weist das Paper daraufhin, dass für die erfolgreiche Klassifikation die berechneten Features sehr wichtig waren, die restlichen aber trotzdem wichtig für das Lernen waren. Insgesamt testete man 26 unterschiedliche Objekte und erreichte eine durchschnittliche Accuracy von 96 Prozent. Die Confusion Matrix in Abbildung 4.2 zeigt das Ergebnis im Detail.

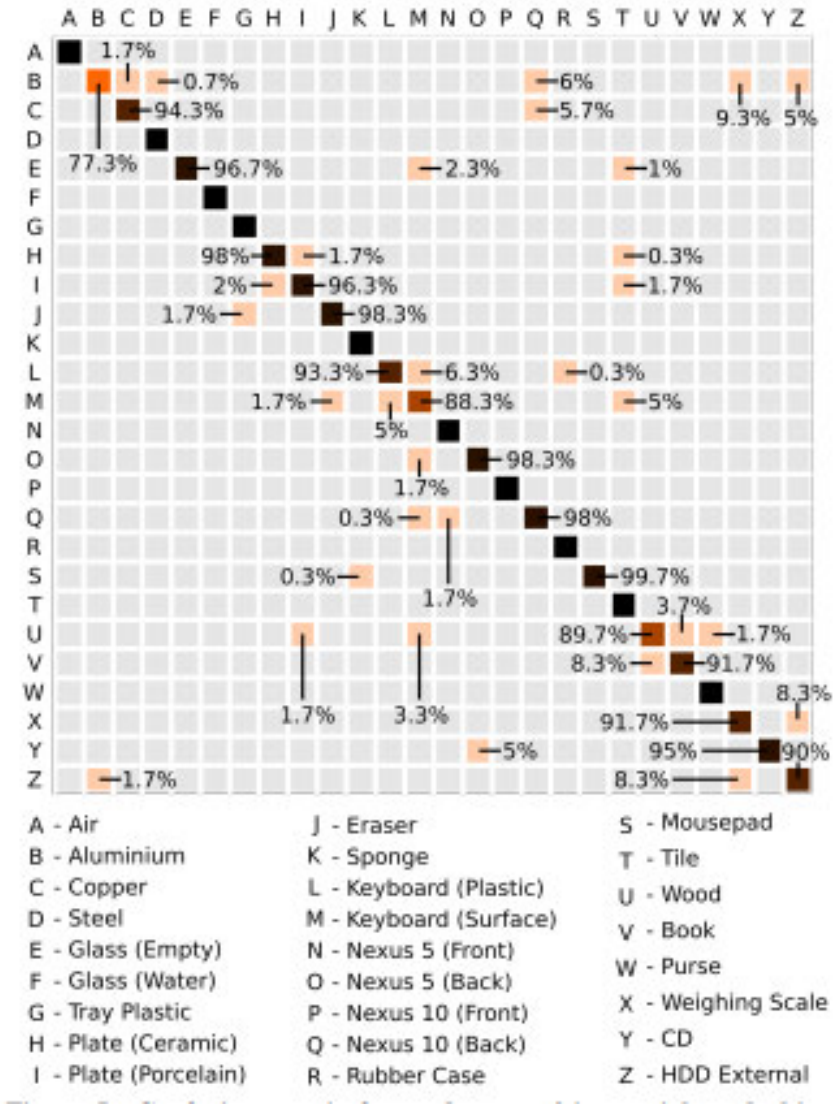

Abbildung 4.2: Confusin Matrix aus dem RadarCat-Paper [23]

5 Experimentelle Umsetzung

Dieses Kapitel beschreibt die Umsetzung des Experiments und gibt einen Überblick über die genutzte Soft- und Hardware.

5.1 Sensor

In dieser Arbeit wurde nicht wie im Original-Experiment der Soli-Sensor verwendet, stattdessen kommt der RadarLog aus dem Hause Inras zum Einsatz. Er bietet insgesamt zwei Sende- und 16 Empfangsantennen. Das FMCW-Radar ermöglicht das Datensamplen von 16 Kanälen und arbeitet mit einer Frequenz von bis zu 77 GHz. Zum Ansteuern des Sensors stehen verschiedene Möglichkeiten zur Verfügung. Er lässt sich per Matlab-Skript oder Python-Programm verwenden. Die Verbindung zwischen Sensor und Computer erfolgt per USB-3.0-Schnittstelle. Der RadarLog ermöglicht eine Abtastung mit 65 MSPS (Mega Samples pro Sekunde). Die Signalverarbeitung und das Timing sind in einem FPGA programmiert [2]. Folgende Abbildung zeigt die Vorderseite des Sensors.

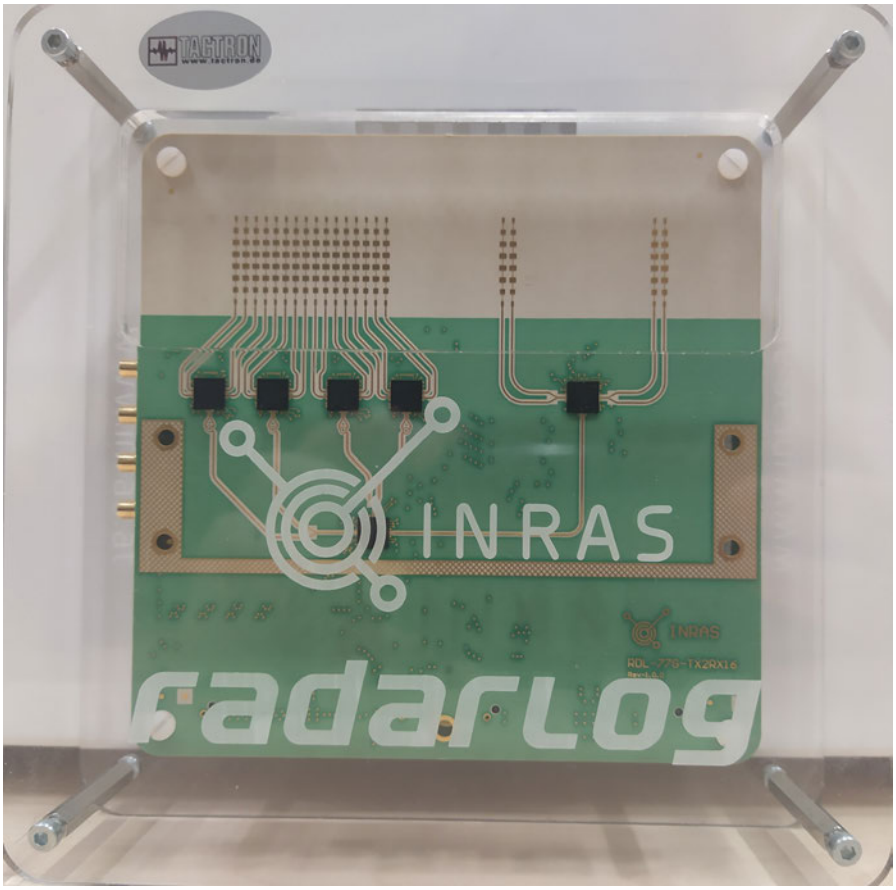

Abbildung 5.1: Eigenes Foto: Inras Radarlog

5.2 Matlab

Der vorliegende Radarsensor wird mittels Matlab-Skript (siehe Anhang) angesteuert. Die Messung geschieht auf allen 16 Empfangskanälen, pro Messung werden 1024 Samples pro Kanal gesammelt. Im Verlauf des Experiments stellte sich heraus, dass die Datenpunkte mit 16.384 (16x1024) Werten/ Attributen pro Gegenstand zu groß für eine performante Klassifizierung sind. Deshalb wurden die Datensätze verkleinert und enthalten nur jedes 64 Sample. Ebenso wurden die Messwerte auf zwei Nachkommastellen gerundet.

Die Rohdaten werden in einer CSV-Datei gespeichert und müssen für die weitere Nutzung nur transponiert, gelabelt und um die Extra-Features erweitert werden. Danach werden die Daten wie in Abschnitt 5.4 beschrieben ins ARFF-Format gebracht.

5.3 Weka

Hinweis: Die Idee und Struktur dieses Unterkapitels entstammt der Bachelorarbeit "Evaluation von Data-Mining-Algorithmen am Beispiel einer Anwendung zur Parkplatzsuche" von Moritz Knüppel [9].

Zur Evaluation der maschinellen Lernverfahren dient in dieser Arbeit das „Waikato Enviroment for Knowledge Analysis", kurz „Weka", in der Version 3.8.4. Dabei handelt es sich um ein an der neuseeländischen University of Waikato entwickeltes Open-Source-Tool. Es ist in Java programmiert und vereint verschiedene Machine-Learning-Algorithmen auf einer Plattform. Darüber hinaus bietet Weka ein Datenkonvertierungstool, um beispielsweise CSV-Dateien in das ARFF-Format zu überführen, Visualisierungswerkzeuge und eine GUI.

Die zentrale Komponente in Weka ist der Explorer, er ist in folgende Bestandteile unterteilt:

- Preprocess: Hier lassen sich Datensätze aus diversen Quellen wie Datenbanken, ARFF-Dateien und CSV-Dateien laden und für die Analyse vorbereiten. Dazu bietet Weka unterschiedliche Möglichkeiten: So lassen sich hier verschiedene Filter anwenden oder einfach einzelne Attribute löschen.
- Classify: Hier finden sich zahlreiche von Klassifizierungsverfahren sowie Evaluationstechniken dazu.
- Cluster: Bietet eine Reihe von Clusteringverfahren.
- Associate: Hier lassen sich Assoziationsverfahren auf die vorliegenden Daten anwenden.
- Select Attributes: Hier lassen die wichtigsten Attribute/Features eines Datensatzes finden.
- Visualize: Hier lassen sich die Daten mittels Streudiagramm anzeigen. Dazu bietet Weka jede Kombination der Attribute als eigenes Diagramm.

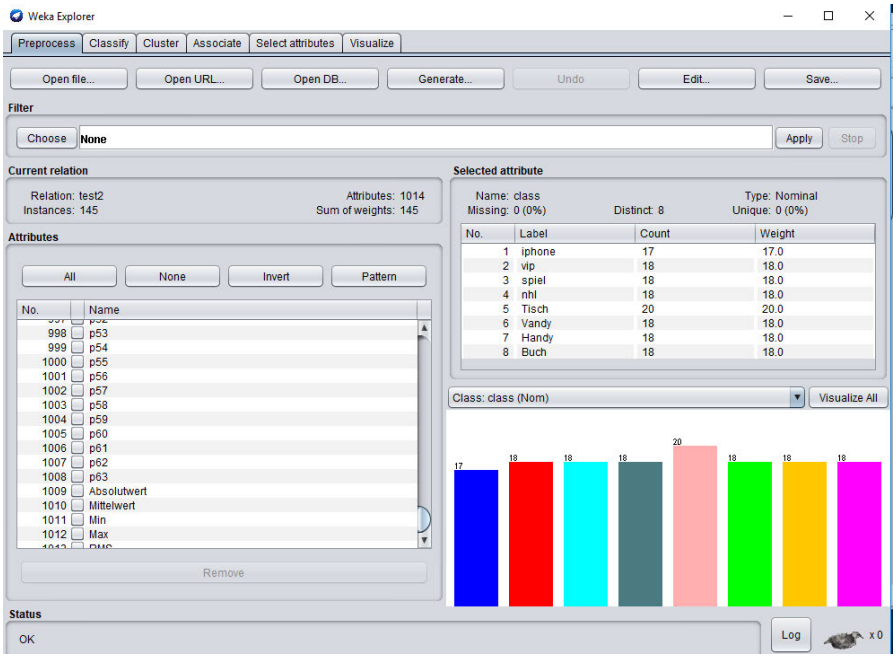

Abbildung 5.2: Screenshot: Weka-Explorer mit geladenem Datensatz

Unter dem Reiter „Classify“ lässt sich nicht nur der gewünschte Klassifizierungs-Algorithmus auswählen, sondern auch die wichtigsten Parameter dazu einstellen. Unter dem Punkt „Test options“ bietet Weka zudem die Möglichkeit, das Testverfahren zu bestimmen. So lassen sich Datensets als Trainingsset oder Testset einsetzen, ein Datenset automatisch teilen (Percentage-Split) oder für die Cross-Validation nutzen.

Im Bereich „Classifier output“ gibt Weka per Standard-Einstellung folgende Werte aus:

- Die korrekt und inkorrekt klassifizierten Objekte (Instanzen), absolut und prozentual
- Die Kappa Statistik
- Den Mean absolute error
- Den Root mean squared error
- Den Relative absolute error
- Den Root relative squared error

In der detaillierten Übersicht gibt Weka zusätzlich noch folgende Werte je Klasse aus:

- True-Positive-Rate
- False-Positive-Rate
- Precision
- Recall
- F-Measure, die Kombination aus Precision und Recall, berechnet mit $(\frac{2 \cdot Precision \cdot Recall)}{(Precision + Recall)}$
- MCC
- ROC Area
- PRC Area

```
Time taken to build model: 0.73 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         137               94.4828 %
Incorrectly Classified Instances         8                5.5172 %
Kappa statistic                          0.9369
Mean absolute error                      0.0501
Root mean squared error                  0.1193
Relative absolute error                 22.8874 %
Root relative squared error             36.0571 %
Total Number of Instances              145

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,824    0,008    0,933      0,824    0,875      0,862    0,984     0,929     iphone
               1,000    0,008    0,947      1,000    0,973      0,969    1,000     1,000     vip
               0,944    0,000    1,000      0,944    0,971      0,968    1,000     1,000     spiel
               0,944    0,016    0,895      0,944    0,919      0,908    0,997     0,979     nhl
               1,000    0,008    0,952      1,000    0,976      0,972    1,000     1,000     Tisch
               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Vandy
               0,889    0,024    0,842      0,889    0,865      0,846    0,987     0,949     Handy
               0,944    0,000    1,000      0,944    0,971      0,968    0,999     0,994     Buch
Weighted Avg.  0,945    0,008    0,946      0,945    0,945      0,938    0,996     0,982
```

Abbildung 5.3: Screenshot: Classifier Output

Abschließend liefert Weka die Ergebnisse in einer Confusion Matrix zusammengefasst.

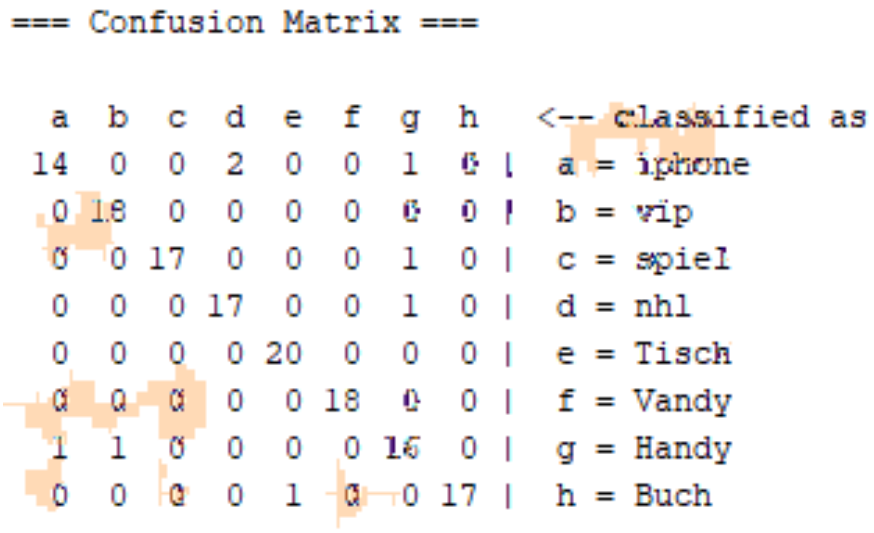

Abbildung 5.4: Screenshot: Confusion Matrix

5.4 Datenformat

Als Datenformat wird für Weka ARFF (Attribut-Relation File Format) verwendet. ARFF-Dateien sind in zwei Teile unterteilt. Der obere Teil, der Header, enthält neben dem Titel eine Liste der Attribute (die Spalten der Daten) und deren Typen. Der zweite Teil enthält zeilenweise die eigentlichen Daten [13].

Folgendes Beispiel zeigt eine ARFF-Datei für ein Datensatz mit zwei Klassen: Buch und Tisch, wobei die Daten aus den zwei Attributen channel1 und channel2 bestehen:

% Title Radardaten

%

% Jahr 2020

%

@RELATION radar

@ATTRIBUTE channel1 NUMERIC

@ATTRIBUTE channel2 NUMERIC

@ATTRIBUTE class Buch, Tisch

@DATA

5.0, 3.5, Buch

4.9, 3.6, Buch

5.1, 3.4, Buch

12.3, 8.1, Tisch

11.9, 7,8, Tisch

12.2, 8.0, Tisch

5.5 Datensatz

Für die Untersuchungen werden in dieser Arbeit drei unterschiedliche Datensätze verwendet. Der erste Datensatz enthält nur die aufgezeichneten Daten des Radarsensors für alle 16 Kanäle. Der zweite Datensatz enthält dieselben Daten des Radarsensors, erweitert um die berechneten Extra-Features, die im Abschnitt 2 beschrieben sind. Abschließend wurde ein drittes Datenset, welches nur die berechneten Extra-Features enthält, zur Klassifikation genutzt. Alle Datensätze enthalten insgesamt 145 Datenpunkte für insgesamt acht unterschiedliche Klassen. Dabei unterscheidet sich die Anzahl der Datenpunkte pro Klasse leicht, zu jedem Objekt sind 17 bis 20 Datenpunkte vorhanden. Abbildung 5.5 zeigt die Übersicht über die Klassenverteilung.

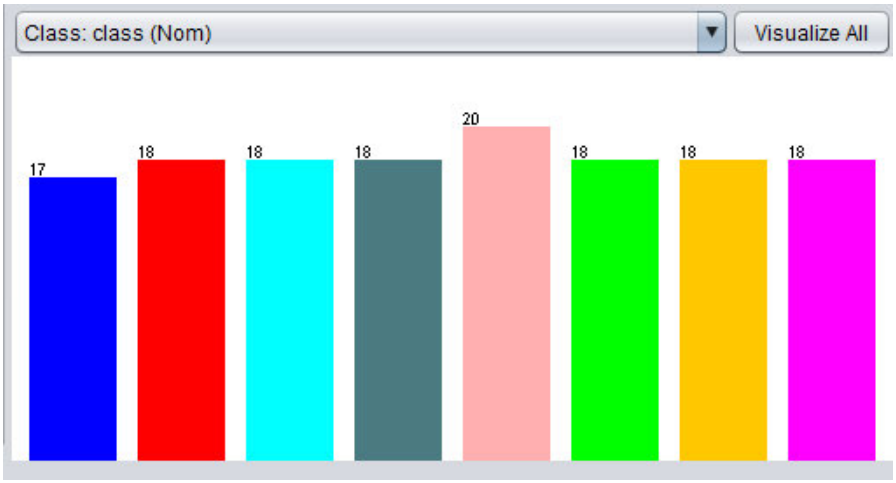

Abbildung 5.5: Screenshot: Klassenübersicht in Weka

Folgende Objekte wurden dabei untersucht:

- iPhone 6 Rückseite (metallisch)
- iPhone 6 Vorderseite (Glas)
- Steelbook Rückseite (Glatt)
- Steelbook Vorderseite (unterschiedlich dick durch Prägung)
- Tisch (Holz)
- Xiaomi Redmi Note 7 Rückseite (Glas)
- Xiaomi Redmi Note 7 Vorderseite (Glas)
- Buch

Die Messdaten stammen aus Messungen von verschiedenen Tagen. Dabei wurden in jeder Session alle Messobjekte nacheinander in zufälliger Reihenfolge auf dem Radarsensor platziert. So sollte einerseits verhindert werden, dass der Sensor durch mehrmalige aufeinanderfolgende Messungen des gleichen Objekts immer die gleichen Werte liefert. Andererseits wurde so die Varianz in den Messdaten durch eine leicht veränderte Platzierung des Objekts auf dem Sensor erhöht. Eine Besonderheit stellt die Klasse Tisch dar, hier wurde nicht wie bei allen anderen Klassen für jede Messung dasselbe Objekt verwendet, sondern gleiche Tische, sprich mehrere baugleiche Tische. In Abbildung 5.6

und 5.7 sind beispielhafte gemessene Signale zu sehen. Zur besseren Übersicht sind hier nur die Daten der ersten acht Kanäle dargestellt.

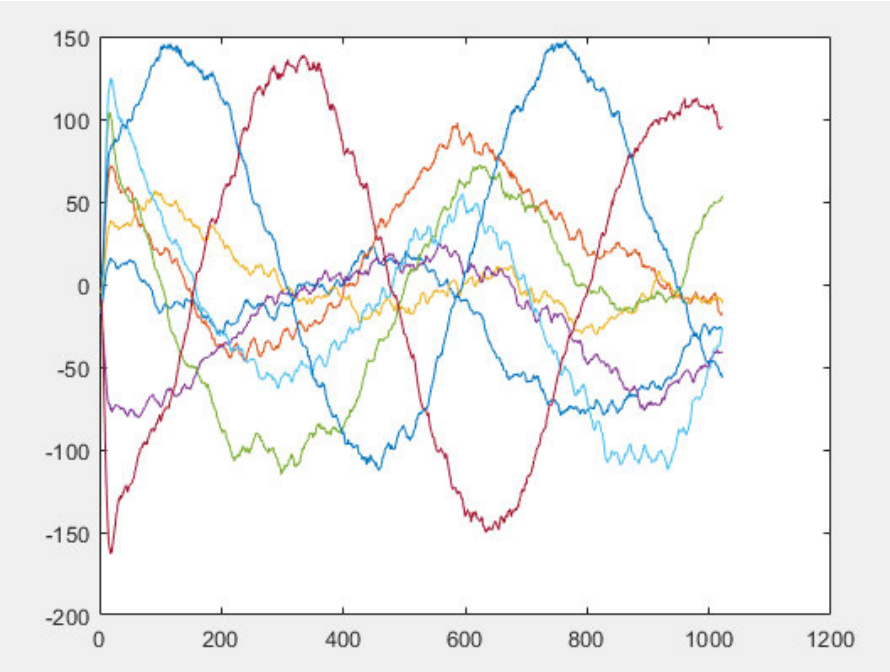

Abbildung 5.6: Signalverlauf Tisch

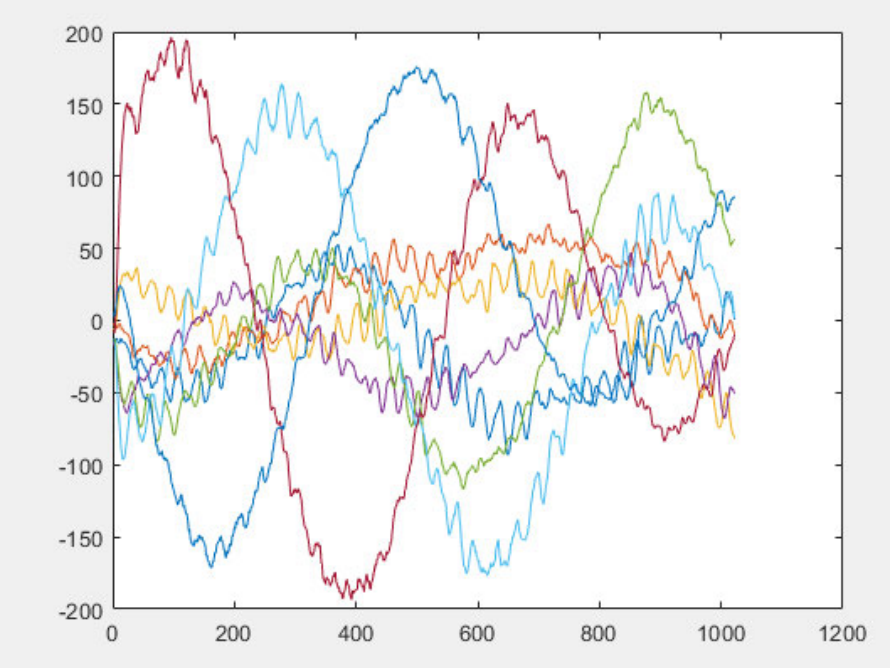

Abbildung 5.7: Signalverlauf iPhone 6

6 Ergebnisse

Dieses Kapitel fasst die Ergebnisse der durchgeführten Untersuchungen zusammen. Dabei werden die Ergebnisse der einzelnen untersuchten maschinellen Lernverfahren dargestellt und untereinander verglichen. Untersucht wurden insgesamt drei maschinelle Lernverfahren. Zum einen eine SVM auf Basis der Bibliothek Libsvm, ein Random-Forest-Ansatz und abschließend ein Multilayer Perzeptron. Dabei kamen pro Lernverfahren jeweils drei Datensätze zum Einsatz. Datensatz 1 besteht aus den reinen gemessenen Daten inklusive Klassenlabel. Der Datensatz 2 beinhaltet zusätzlich die in Abschnitt 2.4 beschriebenen Extra-Features. Datensatz 3 besteht ausschließlich aus den Extra-Features.

Als Testmethoden kommen aufgrund des kleinen Datensatzes einerseits ein Percentage-Split (P-S) und die Cross-Validation (C-V) zum Einsatz. Beim Percentage-Split wurden 66 Prozent der Daten zum Training und 33 Prozent zum Testen verwendet. Das bedeutet, zum Testen dienen hier nur 49 Instanzen (Objekte). Die Cross-Validation ist wie in Abschnitt 3.2 beschrieben, eine stratified 10-Fold-Cross-Validation. Des Weiteren zeigen die Tabellen die Anteile der korrekt und inkorrekt klassifizierten Objekte, den MCC, die ROC Area (ROC AUC), die PRC Area (PCR AUC) sowie die zum Erstellen des Classifiers benötigte Zeit. Die kompletten Classifier-Outputs befinden sich im Anhang.

6.1 Untersuchung mit Datensatz 1

6.1.1 SVM

Die verwendete SVM wurde mit den Standard-Parametern von Weka erstellt, dabei kommt eine lineare Kernel-Funktion zum Einsatz.

Tabelle 6.1: Ergebnis: SVM mit Datensatz 1

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	95,1724	4,8276	0,946	0,972	0,915	0,05
P-S	93,8776	6,1224	0,930	0,964	0,898	0,06

Die SVM klassifiziert im ersten Test (C-V) sieben Objekte falsch. Auffällig dabei ist, dass der Classifier gleich vier Instanzen der Rückseite des iPhones nicht korrekt zuordnet. Weitere Fehler verteilen sich auf die Rückseite des Xiaomi Smartphones sowie das Steelbook. Also die Gegenstände bzw. Seiten der Messobjekte, die aufgrund ihres Materials am meisten reflektieren. Beim Percentage-Split macht das Modell drei Fehler, alle betreffen Instanzen der Rückseite des iPhones.

6.1.2 Random Forest

Die verwendeten Random-Forest-Classifier wurden mit den Standard-Parametern von Weka erstellt. Die Anzahl der Bäume liegt hier bei 100 Stück.

Tabelle 6.2: Ergebnis: Random Forest mit Datensatz 1

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	95,8621	4,1379	0,953	0,996	0,978	0,31
P-S	93,8776	6,1224	0,930	0,994	0,975	0,27

Der Random-Forest-Ansatz wirkt auf dem ersten Datensatz etwas besser als die SVM. Mit der Cross-Validation klassifiziert er sechs Objekte inkorrekt. Wie auch bei der SVM entstammt ein Großteil der Fehler (drei Stück) der Klasse iPhone. Mit dem Unterschied, dass der Random Forest zwei Mal das iPhone nicht erkennt und ein Mal die Rückseite des Xiaomi Smartphones als iPhone klassifiziert. Beim Percentage-Split betreffen alle drei gemachten Fehler die Klasse iPhone.

6.1.3 MLP

Das verwendete und getestete MLP wurde mit einer Lernrate von 0,2 und einem Momentum von 0,3 über 500 Epochen trainiert. Dabei kommt zur Bestimmung des Gradienten der Backpropagation-Algorithmus zum Einsatz. Dabei geschieht die Veränderung der Gewichte durch Multiplikation des Gradienten mit der Lernrate und Addition des Produkts der vorherigen Änderung des Gewichts und des Momentums gemäß folgender Formel.

$$W_{next} = W + \Delta W$$

$$\Delta W = -learningrate * Gradient + Momentum * \Delta W_{previous}$$

[18]

Tabelle 6.3: Ergebnis: MLP mit Datensatz 1

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	93,7931	6,2069	0,933	0,983	0,980	951,51
P-S	93,8776	6,1224	0,939	0,948	0,948	1009,24

Das MLP brauch nicht nur die längste Zeit, um ein Modell zu bilden, sondern hat auch die höchste Fehlerrate. Insgesamt neun Instanzen (C-V) werden nicht korrekt klassifiziert. Auffällig: Jeweils drei Fehler betreffen das Steelbook (Vor- und Rückseite). Grund hierfür sind womöglich die starken Reflexionseigenschaften des Materials sowie die unebene Oberfläche des Gegenstandes.

6.1.4 Vergleich Untersuchung mit Datensatz 1

Auf dem ersten Datensatz erzielen die SVM und der Random Forest mit der Cross-Validation ähnlich gute Ergebnisse. Die SVM (7 Fehler) klassifiziert ein Objekt mehr falsch als der Random Forest (6 Fehler). Dabei ist auffällig, dass es bei vier Instanzen Überschneidungen gibt, beide Verfahren machen bei diesen den gleichen Fehler. Bei der Zeit, die für die Erstellung der jeweiligen Modelle benötigt wird, liegt die SVM vorn. Das MLP machte im Testlauf mit der Cross-Validation insgesamt neun Fehler und brauchte

mit Abstand die meiste Zeit. Die Cross-Validation erzielt bei der SVM und beim Random Forest ein besseres Ergebnis als das Testen mit einem Percentage-Split. Dies ist mit dem kleinen Datenset zu begründen. Bei der Aufteilung des Datensets wurde nicht auf eine proportionale Verteilung der Klassen geachtet. So variiert die Anzahl der jeweiligen Instanzen im Testset zwischen vier und zehn. Dennoch erreichen beim Percentage-Split alle drei Verfahren ein nahezu gleich gutes Ergebnis. Jedes macht gleich viele Fehlklassifikationen, allerdings an unterschiedlichen Stellen. Hier lässt sich sehen, dass die Verfahren dieselben Daten unterschiedlich interpretieren. Inkorrekte Klassifikationen an unterschiedlichen Instanzen erklären auch die unterschiedlichen Werten des MCC, der ROC Area und der PRC Area. Vor allem bei Letzterem zeigt sich ein deutlicher Unterschied, so weist die SVM nur einen Wert von 0,898 auf. Die PRC Area beim Random Forest und beim MLP sind mit 0,975 bzw. 0,948 höher.

An der ersten Untersuchung lässt sich bereits sehen, dass die zur Verfügung stehende Datenmenge für eine Klassifikation ausreicht. Allerdings sollte der Unterschied zwischen dem Percentage-Split und der Cross-Validation bedacht werden. Es ist anzunehmen, dass ein umfangreicheres Datenset zu einem besseren Ergebnis führt. Da es Überschneidungen in den falsch klassifizierten Instanzen gibt, ist zudem anzunehmen, dass die vorhandenen Daten Veränderungen unterliegen und es für das Modell (unabhängig vom Lernverfahren) uneindeutige oder gar unbekannte Fälle gibt. Um dieses Problem zu minimieren, sollten künftige Experimente mit einer größeren Datenmenge durchgeführt werden.

6.2 Untersuchung mit Datensatz 2

Die Untersuchung mit dem zweiten Datensatz bezieht zusätzlich zu den gemessenen Werten weitere berechnete Features mit ein.

6.2.1 SVM

Tabelle 6.4: Ergebnis: SVM mit Datensatz 2

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	95,1724	4,8276	0,946	0,972	0,915	0,06
P-S	93,8776	6,1224	0,928	0,962	0,896	0,05

Im Vergleich zu dem ersten Datensatz ändert sich die Anzahl der Fehler der SVM im Testverfahren mit Cross-Validation nicht. Ferner macht die SVM weiterhin an den gleichen Stellen falsche Klassifikationen. Hier ist durch die Erweiterung des Datensatzes keine Verbesserung zu erkennen. Im zweiten Testverfahren, dem Percentage-Split, sieht es auf den ersten Blick genauso aus. Allerdings klassifiziert die SVM hier einen anderen Datenpunkt falsch. Sprich durch die neuen Features im Datensatz verschob sich ein Fehler, die anderen beiden Fehler blieben. Aus dieser Fehlerverschiebung resultieren die kleinen Änderungen der Evaluationsparameter. Aufgrund der größeren Datenmenge erhöhte sich auch die Zeit zum Bilden des Classifiers.

6.2.2 Random Forest

Tabelle 6.5: Ergebnis: Random Forest mit Datensatz 2

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	95,1724	4,8276	0,946	0,996	0,983	0,31
P-S	91,8367	8,1633	0,908	0,991	0,968	0,28

Im Vergleich zum Datensatz 1 verschlechtert sich das Ergebnis beim Random Forest durch die zusätzlichen Features in den Daten sogar. Bei beiden Test-Verfahren erhöht sich die Anzahl der Fehlklassifikationen um ein Objekt. In beiden Fällen sind alle Fehler aus dem Datensatz 1 auch im Versuch mit Datensatz 2 vorhanden. Zusätzlich klassifiziert der Algorithmus jeweils ein Objekt, welches er zuvor korrekt klassifizierte falsch. Daran lässt sich erkennen, dass mehr Features nicht uneingeschränkt zu einem besseren Ergebnis führen. Die Gründe dafür sind vielfältig. So könnte der Algorithmus Features für seine Klassifikation wählen, die vielleicht schneller (Bäume mit weniger Knoten) zu einem Ergebnis führen, dafür allerdings nicht eindeutig sind. Hier ist der begrenzte Zahlenraum der Messwerte zu nennen, vor allem beim Minimum und Maximum. Dort könnten diverse Objekte ähnliche Werte aufweisen. Somit ist die Datenoptimierung ein Ansatzpunkt für Verbesserungen. Man sollte die für den Algorithmus wichtigen Features identifizieren und gegebenenfalls Features löschen.

6.2.3 MLP

Tabelle 6.6: Ergebnis: MLP mit Datensatz 2

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	94,4828	5,5172	0,939	0,984	0,975	957,38
P-S	93,8776	6,1224	0,936	0,981	0,983	997,05

Das MLP erreicht durch die zusätzlichen Features im Datensatz 2 mit der Cross-Validation ein besseres Ergebnis als mit dem ersten Datensatz. Hier ist - wie bereits in der Untersuchung mit dem ersten Datensatz - zu sehen, dass die drei maschinellen Lernverfahren dieselben Daten unterschiedlich interpretieren. Der im Vergleich zu den anderen ML-Verfahren hohe Zeitaufwand für das Bilden das Classifiers ändert sich nur minimal.

6.3 Untersuchung mit Datensatz 3

Die Untersuchung mit Datensatz 3 nutzt einen stark reduzierten Datensatz, der nur die berechneten Features enthält.

6.3.1 SVM

Tabelle 6.7: Ergebnis: SVM mit Datensatz 3

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	93,7931	6,2069	0,931	0,965	0,894	0,28
P-S	91,8367	8,1633	0,914	0,955	0,869	0,25

Durch Verwendung des reduzierten Datensatzes verschlechtert sich das Ergebnis der SVM etwas. Damit trainiert, klassifiziert das Verfahren dennoch über 90% der Objekte korrekt. Die immer noch hohe Rate an korrekt klassifizierten Objekten deutet darauf hin, dass für die SVM eine Reduzierung der Features möglich ist. Es ist anzunehmen, dass sich durch eine genauere Auswahl der Features bessere Ergebnisse erzielen lassen. Auffällig ist,

dass es - wie bei den vorherigen Untersuchungen auch - beim dritten Datensatz ebenfalls Überschneidungen der falsch klassifizierten Objekte mit dem Random Forest gibt.

6.3.2 Random Forest

Tabelle 6.8: Ergebnis: Random Forest mit Datensatz 3

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	95,1724	4,8276	0,945	0,991	0,963	0,05
P-S	91,8367	8,1633	0,906	0,994	0,972	0,05

Der Random Forest macht, trainiert mit Datensatz 3 genauso viele Fehler wie mit dem Datensatz 2, allerdings nicht exakt die gleichen. Hier ist zu sehen, dass eine Reduzierung der Features möglich ist. Wahrscheinlich lassen sich die Ergebnisse durch mehr Datensamples und eine genauere Feature-Auswahl weiter verbessern.

6.3.3 MLP

Tabelle 6.9: Ergebnis: MLP mit Datensatz 3

Test	correct (%)	incorrect (%)	MCC	ROC Area	PRC Area	t(s)
C-V	89,6552	10,3448	0,883	0,988	0,931	0,34
P-S	79,5918	20,4082	0,794	0,990	0,959	0,33

Das MLP kommt mit Datensatz 3 nicht so gut wie die anderen beiden Verfahren zurecht. Eine Anpassung der Lernrate führte zu besseren Ergebnissen. Anzumerken ist hier der deutlich reduzierte Zeitaufwand durch die kleinere Anzahl Inputneuronen.

6.4 Zusammenfassung der Ergebnisse

Die Ergebnisse aller Datensets zeigen, dass die Objektklassifizierung mittels Radardaten prinzipiell möglich ist. Für die erstellten Datensets eignen sich die SVM und der Random

Forest besser als ein MLP. Dies ist in erster Linie durch das verwendete MLP-Netz zu erklären, da die Anzahl der Eingabewerte sehr groß ist. Dadurch besteht das verwendete Netz aus sehr vielen Eingabeneuronen.

Die SVM und der Random Forest interpretieren die Daten recht ähnlich, dies wird durch Fehlklassifikation der teilweise gleichen Objekte sichtbar. Beide Verfahren scheinen ähnliche Merkmale in den Daten für ihre Entscheidungen zu wählen. Dabei ist die Genauigkeit der Klassifikationen beim Random Forest etwas höher. Dafür bildet die SVM schneller ein Modell als die anderen beiden Verfahren. Je nach Verfahren bietet die Datenoptimierung eine Möglichkeit, die Ergebnisse zu verbessern. Dies wird durch die Untersuchungen mit dem Datensatz 2 und Datensatz 3 deutlich. Wie erwartet führen zusätzliche Features in den Daten zu anderen Ergebnissen. Durch eine geeignete Wahl der Features und eventuell der Reihenfolge der Features sind bessere Ergebnisse zu erwarten. Dabei spielt auch die Menge der Samples, also die Anzahl der Instanzen pro Objekt eine wichtige Rolle. Für verlässlichere Aussagen sollte die Anzahl erhöht werden. Dies wird durch den Vergleich zwischen den Testverfahren klar. So erreichten die Lernverfahren beim Test mit der Cross-Validation in der Regel ein besseres Ergebnis. Da bei der Cross-Validation mehr Daten fürs Training genutzt werden als beim Percentage-Split. Aus mehr Daten konnten die Verfahren mehr Lernen.

7 Zusammenfassung

7.1 Fazit

In dieser Arbeit wurden maschinelle Lernverfahren zur Objektklassifizierung mittels Radardaten eingesetzt und untersucht. Dazu wurden die Grundlagen der Radartechnik erläutert, um eine Einordnung des verwendeten Sensors zu ermöglichen. Ebenso gab es einen kurzen Überblick über die verwendeten maschinellen Lernverfahren sowie Parameter zur Bewertung dieser.

Zur Untersuchung wurden eigene Datensätze für das Training und Testen der Lernverfahren erstellt. Hierzu wurden Messungen mit dem in Kapitel 5 beschriebenen Sensor durchgeführt und die Messwerte anschließend in das ARFF-Format überführt. Neben dem Datenset mit reinen Rohdaten wurden weitere Datensets mit zusätzlichen berechneten Attributen erstellt.

Es wurde gezeigt, dass verschiedene maschinelle Lernverfahren in der Lage sind, anhand von Radardaten Objekte zu klassifizieren. Dabei ist anzunehmen, dass sich die Erkenntnisse dieser Arbeit auch auf ähnlich Fälle übertragen lassen. Dies gilt insbesondere für Untersuchungen mit ähnlichen Radarsensoren und Untersuchungen mit anderen als den hier verwendeten Messobjekten.

Vermutlich lassen sich die Ergebnisse durch den Einsatz anderer maschineller Lernverfahren verbessern. So wäre beispielsweise der Einsatz eines Convolutional Neural Network (CNN) denkbar. Ebenso ist eine Leistungssteigerung durch eine genauere Aufbereitung der Messwerte und Parameter-Tuning der Lernverfahren anzunehmen. Hier ist auch der Umfang der Datensätze zu nennen, denn durch mehr (unterschiedliche) Trainingsdaten lernen die meisten Algorithmen in der Regel mehr. In dieser Arbeit wurde allerdings bewusst ein sehr kleines Datenset verwendet, um sich der Grenze der Objekterkennung mit Radardaten zu nähern.

7.2 Ausblick

In dieser Arbeit wurde die Objektklassifikation mittels Radardaten nur für starre Objekte Untersucht und ausschließlich offline durchgeführt, d. h. die Aufnahme der Testdaten und die Klassifikation fand getrennt voneinander statt. Künftig wäre der Einsatz in einem Echtzeitsystem denkbar, wie es auch im Paper Radarcat beschrieben ist. Mögliche Einsatzgebiete wären eine Sortieranlage, die beispielsweise Müll (Glas, Metall, Plastik und Holz) trennt oder Bars, die entsprechende Radarsensoren in ihre Tische integrieren, um Füllstände von Gläsern zu bestimmen.

Künftige Arbeiten sollten die Objektklassifikation mit weiteren Radarsensoren untersuchen. Dabei sollte man mit einer kleineren Anzahl an Kanälen, weniger Features und oder einer anderen Anzahl an Datenpunkten testen, um die Grenzen der Objekterkennung mittels Radardaten genauer zu erforschen. Ebenso sollte die Art der untersuchten Objekte variiert werden. So könnten Materialien mit starken Reflexionseigenschaften in verschiedenen Dimensionierungen betrachtet werden. Zusätzlich ließe sich die Anwendbarkeit in einem bewegten Umfeld untersuchen.

Literaturverzeichnis

[1] Aberham, Jana ; Kuruc, Fabrizio: *Support Vector Machine*. S. 95–103. In: Kersting, Kristian (Hrsg.) ; Lampert, Christoph (Hrsg.) ; Rothkopf, Constantin (Hrsg.): *Wie Maschinen lernen: Künstliche Intelligenz verständlich erklärt*. Wiesbaden : Springer Fachmedien Wiesbaden, 2019. – URL https://doi.org/10.1007/978-3-658-26763-6_13. – ISBN 978-3-658-26763-6

[2] Andreas Haderer, Inras: *Radarlog*. – URL http://www.inras.at/uploads/media/RDL-77G-TX2RX16.pdf. – Eingesehen am 30.8.2020

[3] Brownlee, Jason: *rocc*. – URL https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/. – Eingesehen am 30.8.2020

[4] Draelos, Rachel: *splitf*. – URL https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/. – Eingesehen am 30.8.2020

[5] Ertel, Wolfgang: *Maschinelles Lernen und Data Mining*. S. 191–264. In: *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*. Wiesbaden : Springer Fachmedien Wiesbaden, 2016. – URL https://doi.org/10.1007/978-3-658-13549-2_8. – ISBN 978-3-658-13549-2

[6] Ertel, Wolfgang: *Neuronale Netze*. S. 265–311. In: *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*. Wiesbaden : Springer Fachmedien Wiesbaden, 2016. – URL https://doi.org/10.1007/978-3-658-13549-2_9. – ISBN 978-3-658-13549-2

[7] Frochte, J.: *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. Carl Hanser Verlag GmbH & Company KG, 2019. – URL https://books.google.de/books?id=gMmCDwAAQBAJ. – ISBN 9783446459977

[8] Göbel, J.: *Radartechnik: Grundlagen und Anwendungen*. VDE-Verlag, 2001. – URL https://books.google.de/books?id=551lAAAACAAJ. – ISBN 9783800725823

[9] Knüppel, Moritz: *Evaluation von Data-Mining-Algorithmen am Beispiel einer Anwendung zur Parkplatzsuche*. 2017. – URL http://edoc.sub.uni-hamburg.de/haw/volltexte/2018/4150/

[10] Kossen, Jannik ; Müller, Maike E.: *Verzerrung-Varianz-Dilemma*. S. 119–123. In: Kersting, Kristian (Hrsg.) ; Lampert, Christoph (Hrsg.) ; Rothkopf, Constantin (Hrsg.): *Wie Maschinen lernen: Künstliche Intelligenz verständlich erklärt*. Wiesbaden : Springer Fachmedien Wiesbaden, 2019. – URL https://doi.org/10.1007/978-3-658-26763-6_16. – ISBN 978-3-658-26763-6

[11] Kossen, Jannik ; Müller, Maike E. ; Ruckriegel, Max: *Entscheidungsbäume*. S. 111–118. In: Kersting, Kristian (Hrsg.) ; Lampert, Christoph (Hrsg.) ; Rothkopf, Constantin (Hrsg.): *Wie Maschinen lernen: Künstliche Intelligenz verständlich erklärt*. Wiesbaden : Springer Fachmedien Wiesbaden, 2019. – URL https://doi.org/10.1007/978-3-658-26763-6_15. – ISBN 978-3-658-26763-6

[12] Krause, Michael ; Natterer, Elena: *Maschinelles Lernen*. S. 21–27. In: Kersting, Kristian (Hrsg.) ; Lampert, Christoph (Hrsg.) ; Rothkopf, Constantin (Hrsg.): *Wie Maschinen lernen: Künstliche Intelligenz verständlich erklärt*. Wiesbaden : Springer Fachmedien Wiesbaden, 2019. – URL https://doi.org/10.1007/978-3-658-26763-6_3. – ISBN 978-3-658-26763-6

[13] Paynter, Gordon: *arf*. – URL https://www.cs.waikato.ac.nz/~ml/weka/arff.html. – Eingesehen am 30.8.2020

[14] Sabri Boughorbel, Mohammed El-Anbari3: *MCC*. – URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5456046/. – Eingesehen am 30.8.2020

[15] School, Data: *verteilung*. – URL https://www.youtube.com/watch?v=OAl6eAyP-yo. – Eingesehen am 30.8.2020

[16] Tharwat, Alaa: Classification assessment methods. In: *Applied Computing and Informatics* (2018). – URL http://www.sciencedirect.com/science/article/pii/S2210832718301546. – ISSN 2210-8327

[17] Witten, Ian H.: *Cross-Validation*. – URL https://www.cs.waikato.ac.nz/ml/weka/mooc/dataminingwithweka/slides/Class2-DataMiningWithWeka-2013.pdf. – Eingesehen am 30.8.2020

[18] Witten, Ian H.: *Cross-Validation*. – URL https://www.cs.waikato.ac.nz/ml/weka/mooc/moredataminingwithweka/slides/Class5-MoreDataMiningWithWeka-2014.pdf. – Eingesehen am 30.8.2020

[19] Witten, Ian H.: *Cross-Validations*. – URL https://www.futurelearn.com/courses/data-mining-with-weka/0/steps/25384. – Eingesehen am 30.8.2020

[20] Wolff, Christian: *Dauerstrichradargeräte (CW-Radar)*. – URL https://www.radartutorial.eu/02.basics/Dauerstrichradarger%C3%A4te.de.html. – Eingesehen am 30.8.2020

[21] Wolff, Christian: *Dopplerfrequenz*. – URL https://www.radartutorial.eu/11.coherent/co06.de.html. – Eingesehen am 30.8.2020

[22] Wolff, Christian: *Dopplerfrequenz*. – URL https://www.radartutorial.eu/02.basics/Frequenzmodulierte%20Dauerstrichradarger%C3%A4te.de.html. – Eingesehen am 30.8.2020

[23] Yeo, Hui-Shyong ; Flamich, Gergely ; Schrempf, Patrick ; Harris-Birtill, David ; Quigley, Aaron: RadarCat: Radar Categorization for Input & Interaction. In: Rekimoto, Jun (Hrsg.) ; Igarashi, Takeo (Hrsg.) ; Wobbrock, Jacob O. (Hrsg.) ; Avrahami, Daniel (Hrsg.): *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016*, ACM, 2016, S. 833–841. – URL https://doi.org/10.1145/2984511.2984515

A Anhang

A.1 Testergebnisse als Confusion Matrix

Hier eine Übersicht der einzelne Confusion Matrizen zu den durchgeführten Untersuchungen. Dabei stehen folgende Bezeichnungen für die Klassen:

iPhone 6 Rückseite = iphone

iPhone 6 Vorderseite = vip

Steelbook Vorderseite = spiel

Steelbook Rückseite = nhl

Tisch = Tisch

Redmi Note 7 Vorderseite = Vandy

Redmi Note 7 Rückseite = Handy

Buch = Buch

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 15  0  0  1  0  0  1  0 |  a = iphone
  0 18  0  0  0  0  0  0 |  b = vip
  0  0 17  0  0  0  1  0 |  c = spiel
  0  0  0 17  0  0  1  0 |  d = nhl
  0  0  0  0 20  0  0  0 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  1  1  0  0  0  0 16  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.1: Confusion Matrix: Random Forest, Cross-Validation, Datensatz 1

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 14  0  0  1  0  0  2  0 |  a = iphone
  0 18  0  0  0  0  0  0 |  b = vip
  0  0 17  0  0  0  1  0 |  c = spiel
  0  0  0 17  0  0  1  0 |  d = nhl
  0  0  0  0 20  0  0  0 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  1  1  0  0  0  0 16  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.2: Confusion Matrix: Random Forest, Cross-Validation, Datensatz 2

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 15  0  1  0  0  0  1  0 |  a = iphone
  0 17  0  0  0  1  0  0 |  b = vip
  1  0 17  0  0  0  0  0 |  c = spiel
  0  0  0 17  0  0  1  0 |  d = nhl
  0  0  0  0 19  0  0  1 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  0  0  0  1  0  0 17  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.3: Confusion Matrix: Random Forest, Cross-Validation, Datensatz 3

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 13  0  0  1  0  0  2  1 |  a = iphone
  0 18  0  0  0  0  0  0 |  b = vip
  0  0 18  0  0  0  0  0 |  c = spiel
  0  0  0 16  0  0  1  1 |  d = nhl
  0  0  0  0 20  0  0  0 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  0  1  0  0  0  0 17  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.4: Confusion Matrix: SVM, Cross-Validation, Datensatz 1

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 13  0  0  1  0  0  3  0 |  a = iphone
  0 18  0  0  0  0  0  0 |  b = vip
  0  0 18  0  0  0  0  0 |  c = spiel
  0  0  0 16  0  0  0  2 |  d = nhl
  0  0  0  0 20  0  0  0 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  0  1  0  0  0  0 17  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.5: Confusion Matrix: SVM, Cross-Validation, Datensatz 2

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 14  0  2  0  0  0  1  0 |  a = iphone
  0 18  0  0  0  0  0  0 |  b = vip
  0  0 16  1  0  0  1  0 |  c = spiel
  0  0  2 16  0  0  0  0 |  d = nhl
  0  0  0  0 20  0  0  0 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  0  0  0  2  0  0 16  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.6: Confusion Matrix: SVM, Cross-Validation, Datensatz 3

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 15  0  0  0  0  0  1  1 |  a = iphone
  0 18  0  0  0  0  0  0 |  b = vip
  0  0 15  0  0  0  1  2 |  c = spiel
  0  0  0 15  0  0  1  2 |  d = nhl
  0  0  0  0 20  0  0  0 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  0  1  0  0  0  0 17  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.7: Confusion Matrix: MLP, Cross-Validation, Datensatz 1

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 14  0  0  1  0  0  1  1 |  a = iphone
  0 18  0  0  0  0  0  0 |  b = vip
  0  0 16  0  0  0  1  1 |  c = spiel
  0  0  0 16  0  0  1  1 |  d = nhl
  0  0  0  0 20  0  0  0 |  e = Tisch
  0  0  0  0  0 18  0  0 |  f = Vandy
  0  1  0  0  0  0 17  0 |  g = Handy
  0  0  0  0  0  0  0 18 |  h = Buch
```

Abbildung A.8: Confusion Matrix: MLP, Cross-Validation, Datensatz 2

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
 15  0  1  0  0  0  1  0 |  a = iphone
  1 16  0  0  1  0  0  0 |  b = vip
  1  0 16  1  0  0  0  0 |  c = spiel
  0  0  2 16  0  0  0  0 |  d = nhl
  0  1  0  0 18  0  0  1 |  e = Tisch
  0  2  0  0  0 16  0  0 |  f = Vandy
  0  0  0  0  0  0 18  0 |  g = Handy
  0  0  0  0  3  0  0 15 |  h = Buch
```

Abbildung A.9: Confusion Matrix: MLP, Cross-Validation, Datensatz 3

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  5  1  0  1  0  0  0  0 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  6  0  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  1  0  0  0  0  0  8  0 |  g = Handy
  0  0  0  0  0  0  0  4 |  h = Buch
```

Abbildung A.10: Confusion Matrix: Random Forest, Percentage Split, Datensatz 1

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  4  2  0  1  0  0  0  0 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  6  0  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  1  0  0  0  0  0  8  0 |  g = Handy
  0  0  0  0  0  0  0  4 |  h = Buch
```

Abbildung A.11: Confusion Matrix: Random Forest, Percentage Split, Datensatz 2

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  4  0  1  0  0  0  2  0 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  6  0  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  0  0  0  1  0  0  8  0 |  g = Handy
  0  0  0  0  0  0  0  4 |  h = Buch
```

Abbildung A.12: Confusion Matrix: Random Forest, Percentage Split, Datensatz 3

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  4  0  0  1  0  0  2  0 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  6  0  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  0  0  0  0  0  0  9  0 |  g = Handy
  0  0  0  0  0  0  0  4 |  h = Buch
```

Abbildung A.13: Confusion Matrix: SVM, Percentage Split, Datensatz 1

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  4  0  0  0  0  1  2  0 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  6  0  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  0  0  0  0  0  0  9  0 |  g = Handy
  0  0  0  0  0  0  0  4 |  h = Buch
```

Abbildung A.14: Confusion Matrix: SVM, Percentage Split, Datensatz 2

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  6  0  1  0  0  0  0  0 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  5  1  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  0  0  0  1  0  0  8  0 |  g = Handy
  0  0  0  0  1  0  0  3 |  h = Buch
```

Abbildung A.15: Confusion Matrix: SVM, Percentage Split, Datensatz 3

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  4  0  0  0  0  0  0  3 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  6  0  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  0  0  0  0  0  0  9  0 |  g = Handy
  0  0  0  0  0  0  0  4 |  h = Buch
```

Abbildung A.16: Confusion Matrix: MLP, Percentage Split, Datensatz 1

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h   <-- classified as
  4  0  0  1  0  0  0  2 |  a = iphone
  0  5  0  0  0  0  0  0 |  b = vip
  0  0  6  0  0  0  0  0 |  c = spiel
  0  0  0  4  0  0  0  0 |  d = nhl
  0  0  0  0  4  0  0  0 |  e = Tisch
  0  0  0  0  0 10  0  0 |  f = Vandy
  0  0  0  0  0  0  9  0 |  g = Handy
  0  0  0  0  0  0  0  4 |  h = Buch
```

Abbildung A.17: Confusion Matrix: MLP, Percentage Split, Datensatz 2

```
=== Confusion Matrix ===

 a b c d e f g h   <-- classified as
 4 2 1 0 0 0 0 0 | a = iphone
 0 5 0 0 0 0 0 0 | b = vip
 0 0 5 1 0 0 0 0 | c = spiel
 0 0 0 4 0 0 0 0 | d = nhl
 0 0 0 0 4 0 0 0 | e = Tisch
 0 3 0 0 0 7 0 0 | f = Vandy
 0 0 0 1 0 0 8 0 | g = Handy
 0 0 0 0 2 0 0 2 | h = Buch
```

Abbildung A.18: Confusion Matrix: MLP, Percentage Split, Datensatz 3

A.2 Hinweise zur Inbetriebnahme des RadarLog-Sensors

Der Sensor lässt sich per Matlab oder Python steuern. Im Folgenden wird nur die Nutzung mittels Matlab behandelt. Nach Installation der zugehörigen Treiber auf dem Computer, sollte der Sensor automatisch erkannt werden. Zur Inbetriebnahme muss zuerst die Stromversorgung hergestellt werden und der RadarLog anschließend per USB-Kabel mit dem Rechner verbunden werden. Wenn das Board erfolgreich gebooted ist, blinkt die

Status-LED. In Matlab müssen vor der Nutzung alle mitgelieferten Klassen zum Pfad hinzugefügt werden. Dann lässt sich mit dem ersten Beispiel die Verbindung zum Sensor testen und Statuswerte wie die Temperatur auslesen. Zur Ansteuerung wurde in dieser Arbeit das von Inras mitgelieferte Matlab-Skript für FMCW-Messungen verwendet. Dabei wurden vor jeder Session einige Messungen als "Warm-up" für den Sensor durchgeführt, da die Temperatur des Sensors zu beginn niedriger ist als nach einigen Messungen. Ebenso sollten die Messobjekte ständig durchgetauscht werden, damit sich der Sensor nicht an etwas "gewöhnt" und möglichst unterschiedliche Daten ausgibt.

Während der Untersuchungen hat sich gezeigt, dass Objekte mit glatten Oberflächen hoch charakteristische Daten liefern, dennoch sollten äußere Einflüsse bedacht werden. So könnten Bewegungen im Hintergrund zu weiteren Reflexionen führen und sollten vermieden werden. In dieser Arbeit wurde der Sensor deshalb stets auf einem Tisch liegend verwendet.

Beim Betrachten der visualisierten Messdaten in Matlab ist verschiedenes zu beachten. Die vom Sensor gelieferten Daten sind begrenzt, sodass die Werte nicht über 2047 bzw. -2048 hinaus gehen. Tauchen diese Werte auf, sollte der Signalverlauf genauer betrachtet werden, um eventuelle Messfehler festzustellen. Ebenso sollte das Augenmerk nicht nur auf dem Signalverlauf, sondern auch auf den Farben der einzelnen Channels im Matlab-Plot liegen.

Glossar

Bagging Methode zum Kombinieren von Daten aus verschiedenen Klassifikationsmodellen.

Blatt Endpunkt eines Entscheidungsbaums, entspricht der Klassifikation.

Dauerstrich-Radar Radar das ununterbrochen Signale sendet und empfängt.

Doppler-Radar Radar, das den Doppler-Effekt nutzt.

Dopplereffekt ist die Änderung eines Signals (zeitliche Stauchung oder Dehnung), wenn sich der Abstand zwischen Sender und Empfänger während der Dauer des Signals ändert.

Epochen Parameter für Neuronale Netze, beschreibt die Anzahl der Trainingsdurchläufe.

Features Hier ist ein Feature die Kombination aus einem Attribut und zugehörigem Wert. In Excel vergleichbar mit einer Zelle.

FMCW-Radar Ein Dauerstrichradar, das das Sendesignal in der Frequenz moduliert.

Frequenz Anzahl der Schwingungen pro Sekunde in der Einheit Hertz (Hz).

Gradientverfahren Methode zur Optimierung des Lernverhaltens.

GUI = Graphical User Interface.

Hauptdiagonale Die Diagonale einer Matrix, die von links oben nach rechts unten verläuft.

Header Zusatzinformationen (Metadaten) als Ergänzung zu den Nutzdaten.

Kernel-Trick Verfahren um einen linearen Klassifikator auf nicht linear separierbare Daten anzuwenden.

Large Margin Classifier Klassifikatorr, der einen möglichst breiten Bereich um die Klassengrenzen sucht. Beispiel: SVM.

Lernrate Parameter für Neuronale Netze. Sie bestimmt die Größe der Schritte bei der Berechnung des Gradienten.

Libsvm Bibliothek für SVMs.

MIMO steht für Multiple Input Multiple Output und beschreibt hier einen Radarsensor mit mehreren Empfangs- und Sendeantennen.

Momentum Parameter zur Berechnung der Gewichte für Neuronale Netze.

Overfitting Auch Überanpassung, sprich die zu Starke Anpassung an Trainingsdaten.

Puls-Radar Radar, das kurze Signale aussendet und in den Sendepausen die Echosignale empfängt.

Resampling Verfahren, die wiederholt Stichproben aus einem Datensatz ziehen, beispielsweise Bagging.

Rohdaten Gemessene Daten ohne weitere Verarbeitung durch beispielsweise Filter.

Threshold Schwellenwert, ab dem sich die Bewertung der Daten ändert.

Underfitting Auch Unteranpassung, bedeutet ein Modell kann aus den Trainingsdaten nicht genug lernen.

Varianz Fehler ausgehend von der Empfindlichkeit auf kleine Änderungen in den Trainingsdaten, eine hohe Varianz verursacht Overfitting.

Verzerrung Fehler ausgehend von falschen Annahmen des Lernalgorithmus, hohe Verzerrung führt zu Underfitting.

Wurzel Anfang eines Entscheidungsbaums, entspricht der Gesamtheit der Daten.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Untersuchung von radardatenbasierter Objekterkennung mithilfe maschineller Lernverfahren

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original