Hochschule Neu-Ulm
University of Applied Sciences

Bachelorarbeit
im Bachelorstudiengang
Informationsmanagement und Unternehmenskommunikation
an der Hochschule für angewandte Wissenschaften Neu-Ulm

Anomalieerkennung durch Machine Learning zur automatisierten Zustandsüberwachung eines Industrieroboters

Erstkorrektor/-in: Prof. Dr. Jürgen Grinninger
Zweitkorrektor/-in: Prof. Dr. Tobias Engel

Verfasser/-in: Paul Jakob (Matrikel-Nr.: 275386)
Fachsemester: 8

Thema erhalten: 25.04.2023
Arbeit abgegeben: 11.08.2023

Abstract

Die nachfolgende Arbeit untersucht, wie durch Machine Learning die eingestellte Nutzlast eines Industrieroboter-Greifarms auf die Richtigkeit der Einstellung überprüft werden kann und welche Faktoren die Genauigkeit eines solchen Vorhersagemodells beeinflussen. Im Bereich von Industrierobotern findet Machine Learning oft Anwendung im Bereich der Wartung, seltener jedoch geht es um die Echtzeitüberwachung und Korrektur von Robotereinstellungen, sofern diese nicht zu unmittelbaren Ausfällen führen. Um zu überprüfen, wie dies möglich ist und welche Faktoren die Genauigkeit beeinflussen wird ein quantitatives Experiment durchgeführt. Bei diesem werden mithilfe erhobener Daten drei verschiedene Machine Learning-Modelle trainiert. Dabei zeigt sich dass vor allem die Wahl des Machine Learning-Modells den größten Einfluss auf die Genauigkeit eines solchen Vorhersagemodells hat. Neben der Wahl der Modellart spielen aber auch die Wahl der Trainingsdaten, die Feature-Wahl sowie die Modellparameter eine Rolle. Diese Einflussfaktoren sind dabei immer abhängig von dem gewählten Machine Learning-Modell. Für Anwendbarkeit unter Produktionsbedingungen bedarf es jedoch weiterer Versuche im Bereich der Umsetzung unter Realbedingungen.

Keywords: Machine Learning, Industrie 4.0, Condition monitoring, Payload monitoring, Industrial robot

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

kNN ..... k-Nearest Neighbor

1 Einleitung

1.1 Problemstellung

Durch einen kontinuierlichen steigenden und globalen Wettbewerb werden Produktionsunternehmen gezwungen ihre Produktion zu optimieren. In den Abläufen ist dabei immer weniger Platz für potenzielle Fehler und es wird stets nach neuen Wegen gesucht um so nah wie möglich am Idealzustand, ohne jegliche Fehler zu operieren (vgl. Kletti/Rieger 2022, S. 11).

Eine Chance diesem Idealzustand näher zu kommen, ist dabei sicherlich der Trend zur vernetzten Produktion in Form der Industrie 4.0. Zentraler Bestandteil der Industrie 4.0 ist dabei die Verfügbarkeit von allen relevanten Informationen in Echtzeit sowie deren Analyse und Interpretation (vgl. Köhler/Six/Michels 2015, S. 18). Mit der Industrie 4.0 erhalten eine Vielzahl von neuen Technologien Einzug in die Industrie und somit auch in den Produktionsbereich. Einiger diese Technologien sind beispielsweise das Internet of Things, Robotik, Augemented Reality, Virtual Reality, Big Data, sowie Künstliche Intelligenz (vgl. Pérez-Juárez et al. 2022, S. 87). Die Anwendung der Robotik hat dabei mittlerweile in den meisten Produktionsunternehmen Einzug gefunden. Dies wird klar, wenn man sich die weltweiten Absatzzahlen von Industrierobotern anschaut. Im Jahr 2020 wurden 394 000 Industrieroboter verkauft, während sich die Prognose für das Jahr 2025 auf 690 000 Industrieroboter beläuft. Dies entspricht einer jährlichen Absatzsteigerung von 15 % (vgl. IFR 2022, S. 29). Beim Einstellen und Programmieren von Roboterprogrammen handelt es sich um eine Tätigkeit die sich entscheidend auf die spätere Prozessqualität auswirkt (vgl. Meyer 2011, S. 17). Unterläuft hier ein menschlicher Fehler kann dies im schlimmstenfalls zum Stillstand einer Maschine oder einer gesamten Produktionskette führen.

Da es sich beim Menschen um das unzuverlässigste Glied innerhalb eines Produktionsprozesses handelt (vgl. Gołda/Kampa/Paprocka 2018, S. 24), existiert dort eine potenzielle menschliche Schwachstelle für eine auf Industrierobotern basierte Produktion. Gleichzeitig bietet jedoch eine weitere Technologie aus dem Bereich Industrie 4.0 in Form des Machine Learnings, die Möglichkeit diese Schwachstelle zu schließen.

Krauß et al. sehen verschiedene Anwendungsgebiete für Machine Learning in der Produktion. Eines dieser Anwendungsgebiete ist dabei die Anomaly Detection im Bereich Maschinen und Anlagen. Dabei geht es darum Maschinenzustände zu überwachen und Fehler frühzeitig zu identifizieren. (vgl. Krauß et al. 2019, S. 40)

1.2 Zielsetzung

Im Zuge dieser Arbeit soll die Anwendung von Anomaly Detection praxisnah in Form eines Experimentes überprüft werden. Dabei beschränkt sich die Überwachung des Maschinenzustands auf die eingestellte Nutzlast des Industrieroboters. Diese Nutzlast muss vor dem Start eines Roboterprogramms eingestellt werden. Ist die Einstellung dieser Nutzlast falsch und verlässt einen bestimmten Toleranzbereich, so beendet der Roboter aus Selbstschutzgründen das Programm frühzeitig. Es kommt somit zum Produktionsstopp. Um einen solchen Produktionsstopp zu vermeiden, soll eine Art Frühwarnsystem entwickelt werden, welches bereits bei falschen Einstellungen innerhalb des Toleranzbereichs anschlägt. Somit wird verhindert, dass es zu einem späteren Zeitpunkt innerhalb des Programms, beim Verlassen des Toleranzbereichs, zum Stillstand kommt.

Konkret befasst sich diese Arbeit mit der Fragestellung, wie durch Machine Learning die eingestellte Nutzlast eines Industrieroboter-Greifarms auf die Richtigkeit der Einstellung überprüft werden kann und welche Faktoren die Genauigkeit eines solchen Vorhersagemodells beeinflussen.

2 Theoretische Grundlagen und Hintergründe

2.1 Machine Learning

2.1.1 Definition und Klassifikation

Der Begriff Machine Learning wurde hauptsächlich von Arthur Samuel, während seiner Forschung im Jahr 1959 zum maschinellen Lösen eines Damespiels, geprägt (vgl. Joshi 2022, S. 4). Samuels Forschung war eines der frühsten und einflussreichsten Beispiele von Machine Learning und verhalf IBM in den 1950er zu einem der führenden Unternehmen im Bereich KI-Forschung zu werden (vgl. Wiederhold/McCarthy 1992, S. 8).

Grundsätzlich geht es bei Machine Learning um das Gewinnen von Wissen aus Daten. Es handelt sich um ein Gebiet an der Schnittstelle zur Statistik, künstlichen Intelligenz und Informatik. Da es um die Vorhersage von zukünftigen Daten, auf Grundlage von vorhandenen Daten geht wird Machine Learning auch als „prädiktive Analytik“ oder „statistisches Lernen“ bezeichnet (vgl. Müller/Guido 2016, S. 1).

IBM hingegen klassifiziert Machine Learning als Teilgebiet der künstlichen Intelligenz und der Informatik. Es konzentriert sich auf die Nutzung von Daten und Algorithmen, um die Art wie Menschen lernen zu imitieren und so die Genauigkeit schrittweise zu verbessern (IBM o. D.-b).

Joshi definiert den Begriff des Machine Learnings als ein Computerprogramm, welches ein Verhalten zeigen kann, ohne dass dieses Verhalten vom Programmentwickler ausdrücklich programmiert wurde. Weiter noch kann das Programm ein Verhalten entwickeln, dessen sich der Entwickler völlig unbewusst ist. Das entwickelte Verhalten kommt aufgrund von drei Faktoren zustande: erstens den Daten, die vom Programm verwendet und verarbeitet werden, zweitens einer Metrik, oder auch Abstandsfunktion, welche den Fehler oder den Abstand zwischen aktuellem und idealem Verhalten quantifiziert, sowie drittens einem Feedback Mechanismus, welcher die vorherigen Fehler oder Abweichungen vom idealen Verhalten nutzt, um bei zukünftigen Ereignissen ein besseres Verhalten zu erzielen (vgl. Joshi 2022, S. 4).

Machine Learning nutzt also statistische und mathematische Algorithmen, um aus vorhandenen Daten Muster zu lernen und mithilfe dieser erkannten Muster aus historischen Daten auf zukünftige Daten zu schließen (vgl. Verdhan 2020, S. 2-3).

2.1.2 Machine Learning Arten

Machine Learning kann in drei Hauptkategorien unterteilt werden. Das Supervised Learning (überwachtes Lernen), das Unsupervised Learning (unüberwachtes Lernen) und in Reinforcement Learning (verstärkendes Lernen) (vgl. Raschka 2017, S. 24). Neben den drei Hauptkategorien gibt es eine Vielzahl von hybriden Ansätzen, bei welchen es sich meist um eine Mischform von Supervised Learning und Unsupervised Learning handelt (vgl. Sah 2020, S. 1-2).

2.1.2.1 Supervised Learning

Beim Supervised Learning handelt es sich um die wichtigste und am weitesten verbreitete Methode des Machine Learnings (vgl. Cunningham/Cord/Delany 2008, S. 21). Beim Supervised Learning ist das Ziel ein bestimmtes Ergebnis aufgrund von Eingaben vorherzusagen. Dabei wird das Machine Learning Modell von Daten trainiert, welche sowohl die Eingabedaten als auch die gewünschten Ausgabedaten beinhalten (vgl. Müller/Guido 2016, S. 27). Man spricht deshalb von „supervised“ da die Trainingsdaten bereits den gewünschten Zielwert beinhalten und dieser somit die Genauigkeit des Modells überwacht. (vgl. Raschka 2017, S. 24) Ziel des Supervised Machine Learnings ist es also eine mathematische Gleichung zu erstellen, welche die in den Daten vorhandenen Eingangsvariablen mit der Zielvariablen in eine Beziehung setzt, um anschließend von den Eingangsvariablen auf die Zielvariabel schließen zu können. (vgl. Verdhan 2020, S. 25)

Supervised Learning kann in zwei Unterkategorien geteilt werden: Klassifizierung und Regression.
Bei der Klassifizierung werden Trainingsdaten in bestimmte Kategorien eingeordnet, zum Beispiel verschiedene Obstsorten wie Äpfel oder Birnen (vgl. Delua 2021). Dabei ist es nicht wichtig das es sich um eine binäre Unterteilung handelt, auch eine Unterteilung in mehrere Kategorien ist möglich (vgl. Raschka 2017, S. 25). Das heißt, würden neben Äpfeln und Birnen noch weitere Obstsorten, wie Erdbeeren oder Melonen, den Trainingsdaten hinzugefügt werden, welche mithilfe eines Machine Learning Algorithmus kategorisiert werden sollen, so würde es sich weiterhin um eine Klassifizierung handeln.
Bei der Regression hingegen ist das Ziel die Vorhersage einer Zahl (vgl. Müller/Guido 2016, S. 28). Dabei wird auf Grundlage von vorhandenen Datenpunkten eine Funktion erstellt welche versucht die Beziehung zwischen Eingangsvariablen und Zielvariablen darzustellen (vgl. Delua 2021). Ein Beispiel hierfür wäre die Punkteanzahl in einer Prüfung. Sofern es einen Zusammenhang zwischen der Übungszeit und der erreichten Punktzahl gäbe, wäre es mit passenden Trainingsdaten möglich durch Machine Learning von der Anzahl der Übungsstunden auf eine Punktzahl in der Prüfung zu schließen (vgl. Raschka 2017, S. 26).

Zusammenfassend spricht man also von Klassifizierung, wenn eine kategoriale Variabel vorhergesagt werden sollen und von Regression, wenn der Wert einer kontinuierlichen Variabel vorhergesagt werden soll (vgl. Verdhan 2020, S. 28).

2.1.2.2 Unsupervised Learning

Anders als beim Supervised Learning, bei welchem man die richtige Antwort beim Trainieren der Daten bereits kennen, kennt man diese beim Unsupervised Learning nicht. Man hat es mit Daten von unbekannter Struktur zu tun. Unsupervised Learning hilft dabei, diese unbekannten Strukturen zu erkunden und sinnvolle Informationen daraus zu gewinnen (vgl. Raschka 2017, S. 28). Gewonnene Informationen können anschließend in verschiedenen Formen auftreten. So können Strukturen beispielsweise in Form von Ähnlichkeitsbeziehungen ausfindig gemacht werden. Alternativ können auch Untergruppen innerhalb der Daten gebildet werden, bei welchen die Ähnlichkeit innerhalb der jeweiligen Untergruppen hoch ist, während sie von Untergruppe zu Untergruppe gering ist (vgl. Langs/Munro/Wazir 2022, S. 239).
Wie auch beim Supervised Learning werden die Machine Learning Modelle genauer, je mehr Datensätze dem Algorithmus beim Trainieren zur Verfügung stehen. Jedoch ist diese Datenmenge beim Unsupervised Learning noch wichtiger, da der Algorithmus versucht bestehende Muster in Datensätzen ausfindig zu machen (vgl. Naeem et al. 2023, S. 913).

Unsupervised Learning wird in drei Unterkategorien unterteilt: Clustering, Assoziation und Dimensionsreduzierung (vgl. Delua 2021).
Beim Clustering werden Datenpunkte in sogenannten Clustern gruppiert (vgl. Tyagi et al. 2022, S. 2). Innerhalb eines Clusters befinden sich Daten mit gemeinsamen Eigenschaften, wohingegen sich Eigenschaften zwischen zwei oder mehr Clustern unterscheiden (vgl. Qamar/Raza 2023, S. 167).
Bei der Assoziation geht es darum ein Modell zu erstellen, welches Beziehungen und Abhängigkeiten innerhalb von Datensätzen findet und lernt (vgl. Langs/Munro/Wazir 2022, S. 240). In der Praxis wird dies häufig verwendet, um neue Beziehungen oder Abhängigkeiten bei der Warenkorbanalyse oder bei passenden Empfehlungen für Kunden ausfindig zu machen (vgl.Delua 2021).
Bei der Dimensionsreduzierung wird ein Datensatz mit einer Vielzahl von Merkmalen und Dateneinträgen herangezogen und eine neue Darstellung dieses Datensatzes vorgenommen, welche die wesentlichen Merkmale mit weniger Merkmalen zusammenfasst (vgl. Müller/Guido 2016, S. 133). Die Datenmenge wird reduziert, während sich die Datenintegrität nicht ändert. Die Dimensionsreduzierung findet ihre Anwendung oft in der Vorbereitung von Daten (vgl. Delua 2021).

Eine der größten Herausforderungen in der Praxis mit Unsupervised Machine Learning ist die Evaluation. Ziel dabei ist es herauszufinden, ob das Modell etwas Sinnvolles erlernt hat, da man im Vorhinein nicht weiß, wie eine richtige Ausgabe aussehen soll. Daher wird Unsupervised Learning oft in der Datenerforschung oder Datenvorbereitung für Supervised Learning verwendet (vgl. Müller/Guido 2016, S. 134).

2.1.2.3 Reinforcement Learning

Anders als beim Supervised Learning gibt es hier nichts, was das Lernen „überwacht“ und somit als Lehrer für den Algorithmus dient. Vielmehr lernt der Algorithmus beim Reinforcement Learning mithilfe von „Verstärkung“, also einer Art Kritiker, welcher dem Algorithmus nach dem Lösen einer Aufgabe rückmeldet, wie gut oder schlecht dieser die Aufgabe gelöst hat (vgl. Alpaydin 2021, S. 582). Diese Rückmeldung wird auch als Belohnung bezeichnet und in Form einer Belohnungsfunktion angegeben. Der Algorithmus, beim Reinforcement Learning oft auch als Agent bezeichnet, arbeitet dabei mit einfachem Ausprobieren von Abläufen oder Interaktionen mit seiner Umgebung. Durch die Belohnung lernt der Algorithmus, welche Abläufe und Umgebungsinteraktionen gut und welche schlecht funktioniert haben (vgl. Raschka 2017, S. 27). Dabei kann der Algorithmus nicht nur Belohnungen erhalten, sondern auch Bestrafungen, welche für bestimmte Ereignisse oder Verhaltensweisen vordefiniert werden (vgl. Qamar/Raza 2023, S. 47).
Diese Art des Lernens weist viele Parallelen zu biologischen Prozessen auf, durch welche auch Menschen, vor allem in den ersten Jahren ihres Lebens, lernen. Menschen lernen zwar nicht mit einer numerischen Belohnungsfunktion wie ein Machine Learning Algorithmus, aber dennoch folgen sie unterschiedlichen Arten von Belohnungen, beispielsweise dass es ein gutes Gefühl ist zu essen, wenn man hungrig ist (vgl. Langs/Munro/Wazir 2022, S. 242-243).

2.1.3 Machine Learning Modelle

2.1.3.1 Decision Tree

Decision Trees (Entscheidungsbäume) gehören zu den Algorithmen der Klassifizierung. Sie funktionieren mit einer Reihe von Entscheidungen entlang des Baumes und seiner Verzweigungen.

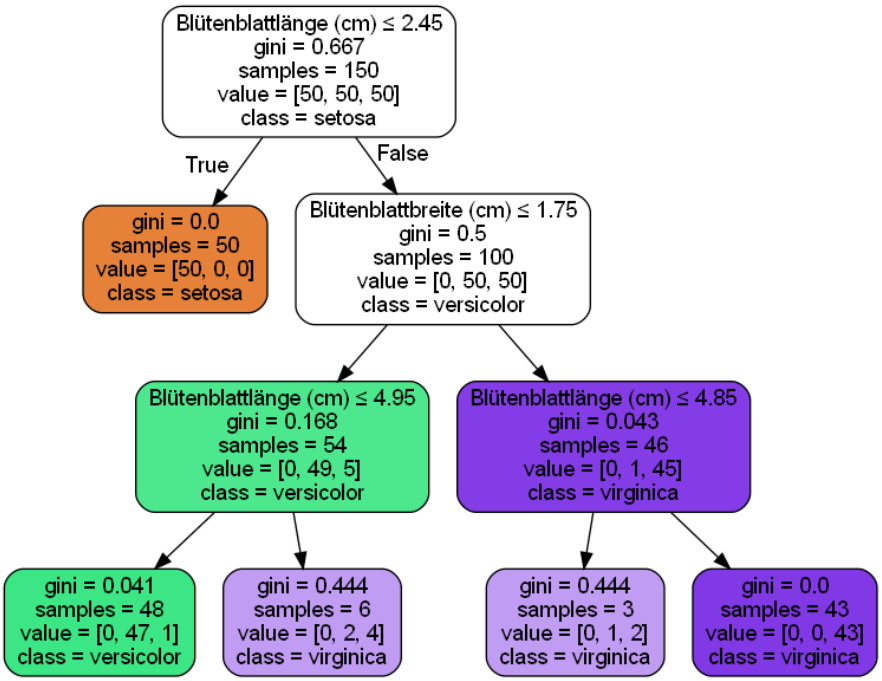

Abbildung 1: Decision Tree zur Klassifizierung von verschiedenen Schwertlilienarten
Quelle: Eigenen Darstellung
Daten-Quelle: (Fisher 1988)

Jedes Blatt entlang des Baumes repräsentiert dabei ein Merkmal, auf welches der zu klassifizierende Datensatz überprüft wird. Nach Überprüfung einer gewissen Anzahl von Merkmalen wird im Decision Tree ein Blatt erreicht, welches das Ende des Decision Trees kennzeichnet und dem zu klassifizierenden Datenpunkt ein Label zuweist (Abbildung 1) (vgl. Langs/Munro/Wazir 2022, S. 228).

Im Gegensatz zu anderen Machine Learning Algorithmen sind Decision Trees einfach zu verstehen und interpretieren. Außerdem ist der Weg einer Vorhersage, im Gegensatz zu anderen Modellen, komplett nachvollziehbar (vgl. Grus 2015, S. 202). Ein weiterer Grund für die Nutzung von Decision Trees ist die unkomplizierte Nutzung bei Daten, welche nicht ausschließlich aus Zahlen bestehen. Während einige Machine Learning Algorithmen nur mit numerischen Daten arbeiten können, können Decision Trees auch kategoriale Daten in ihren Entscheidungsprozess miteinbeziehen, ohne diese vorher in numerische Daten umwandeln zu müssen (vgl. Joshi 2022, S. 73).

Ein Problem von Decision Trees ist jedoch das häufige Overfitting (Überanpassung) von Trainingsdaten. Wird ein Decision Tree in seiner Tiefe nicht beschränkt, erstellt dieser häufig so viele Verzweigungen, bis eine Genauigkeit von 100 % innerhalb der Trainingsdaten erreicht wird. Dabei verlieren Decision Trees die Anwendbarkeit auf neue Daten und ihre Allgemeingültigkeit (vgl. Grunert 2021, S. 231-232).

2.1.3.2 k-Nearest Neighbor

k-Nearest Neighbor (k-Nächster Nachbar) oder auch kurz kNN ist eines der weniger komplexen Machine Learning Modelle, da es keine mathematischen Annahmen trifft, um eine Vorhersage zu treffen. Es basiert lediglich auf der Annahme, dass sich nahe Datenpunkte ähnlich sind (vgl. Grus 2015, S. 151). kNN wird primär für die Klassifikation genutzt, kann jedoch auch für Regressionsprobleme implementiert werden (vgl. Singh 2018).

Um einen Datenpunkt mithilfe von kNN zu klassifizieren, muss man lediglich die fünf, bei k=5, nächstgelegenen Nachbarn finden (Abbildung 2) und kann so auf die Zugehörigkeit des Datenpunkts schließen (vgl. Joshi 2022, S. 52-54).

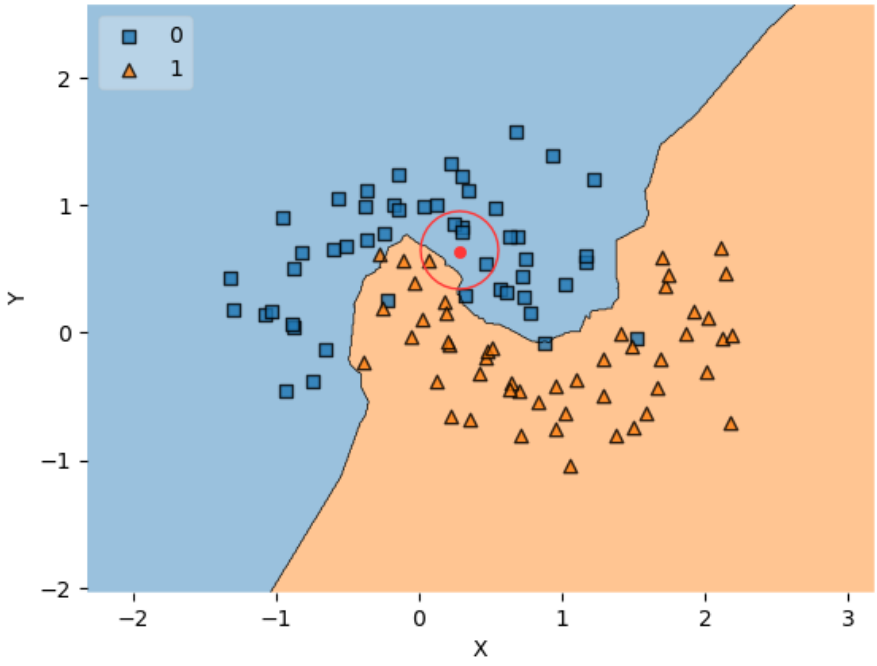

Abbildung 2: kNN mit k=5 zur Klassifizierung eines Datenpunktes
Quelle: In Anlehnung an (Deepthi A R 2019)
Daten-Quelle: (Adityarajora 2019)

Entschieden wird die Zugehörigkeit dabei auf Grundlage einer einfachen Mehrheitsentscheidung. Handelt es sich bei k um eine gerade Anzahl, kann es bei einer Mehrheitsentscheidung zu einer Pattsituation kommen. Tritt dieser Fall auf, werden die unterschiedlichen Nachbarn nach ihrem jeweiligen Abstand gewichtet und entscheiden so über die Zugehörigkeit des Datenpunkts (vgl. Raschka 2017, S. 105-106). Dabei muss sich der Datenpunkt jedoch nicht in einem zweidimensionalen Raum befinden. Der Abstand zu anderen Nachbarn kann auch in einem mehrdimensionalen Raum betrachtet werden. Somit ist kNN auch dann anwendbar, wenn mehr als

zwei Merkmale innerhalb eines Datensatzes betrachtet werden sollen (vgl. Müller/Guido 2016, S. 38).

kNN gehört zu der Klasse der Lazy Learning Algorithmen (träger Lernalgorithmus), da Trainingsdaten einfach gespeichert werden, statt daraus ein Modell zur Vorhersage zu erstellen (vgl. Raschka 2017, S. 104). Da sich die vorhandenen Trainingsdaten im Speicher des Computers befinden und als Beispiele für die Vorhersage genutzt werden, spricht man auch von speicherbasierter Klassifikation oder von beispielbasierter Klassifikation (vgl. Cunningham/Cord/Delany 2008, S. 29-30).

Neben den bereits besprochenen Vorteilen, wie das intuitive Verstehen des Machine Learning Modells oder der schnellen Anpassung an Trainingsdaten durch das Laden dieser in den Arbeitsspeicher, bringt kNN auch einige Nachteile mit sich. So verlangsamt das direkte Laden der Trainingsdaten in den Speicher die Berechnungszeit bei großen Datenmengen. Weiter kann kNN, je nach Wahl der zu betrachtenden Nachbarn, auch anfällig für Ausreißer innerhalb der Daten sein (vgl. Grunert 2021, S. 257).

2.1.3.3 Logistische Regression

Regression oder lineare Regression kommt in der Regel immer dann zum Einsatz, wenn man eine stetige Zielvariabel schätzen möchte (vgl. Hirschle 2020, S. 54). Bei der logistischen Regression ist hingegen das Ziel ein Klassifikationsproblem zu lösen. Die logistische Regression besitzt daher keine metrischen Werte als Zielvariabel, sondern einen binären Wert, null oder eins, wahr oder falsch. (vgl. Grunert 2021, S. 200) Im Grunde handelt es sich bei der logistischen Regression um eine Variante der linearen Regression, bei welcher statt einer Geraden eine Logarithmusfunktion, auch Sigmoidfunktion genannt, angewandt wird (vgl. Joshi 2022, S. 51).

Bei der logistischen Regression befinden sich die Vorhersagen zwischen null und eins, während jede der beiden Zahlen eine Klasse widerspiegelt. Da sich eine logistische Funktion bei negativen Werten auf der X-Achse an null annähert und für positive Werte an eins annähert, können alle Vorhersagen eindeutig der Klasse null oder eins zugewiesen werden (Abbildung 3) (vgl. Grus 2015, S. 189- 192).

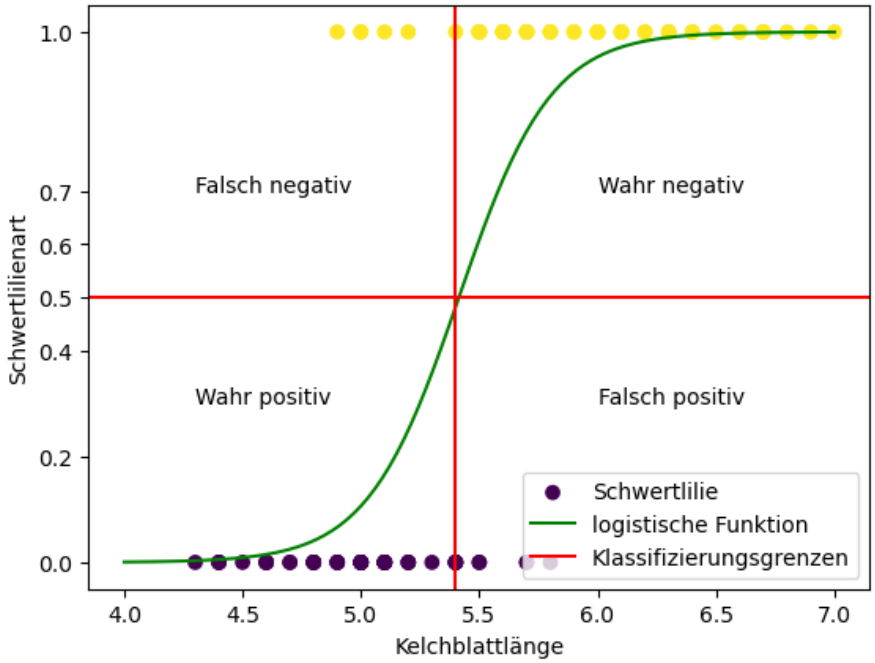

Abbildung 3: logistische Regression zur Klassifizierung zweier Schwertlilienarten
Quelle: In Anlehnung an (Prasad n. D.)
Daten-Quelle: (Fisher 1988)

Anders als bei anderen Machine Learning Arten erhält man nicht nur die Vorhersage zu einem neuen Datenpunkt, sondern zusätzlich die Wahrscheinlichkeit, mit welcher diese Vorhersage korrekt ist. Bei einer Ausgabe von 0,7 so bedeutet dies, dass es sich bei der zu klassifizierenden Eingabe um 70 % um die Klassifizierungsgruppe eins handelt. Bei einer Ausgabe von 0,3 ist die Eingabe zu 30 % Teil der Klassifizierungsgruppe eins, oder auch 1 – 0,3 also zu 70 % Teil der Klassifizierungsgruppe null ist (vgl. Hirschle 2020, S. 54). Deshalb wird die logistische Regression in der Praxis immer dann angewandt, wenn neben der Vorhersage auch eine Wahrscheinlichkeit der Zugehörigkeit angegeben werden soll (vgl. Qamar/Raza 2023, S. 328). Beispiele sind die Regenwahrscheinlichkeit innerhalb einer Wettervorhersage oder auch um im medizinischen Bereich, die Wahrscheinlichkeit für Krankheiten bei Patienten mit bestimmten Symptomen vorherzusagen. (vgl. Raschka 2017, S. 76-77)

2.1.4 Modellevaluation

Es gibt mehrere Möglichkeiten ein Machine Learning-Modell zu bewerten. Eine der gängigsten Methoden für die Bewertung ist die Accuracy (Genauigkeit), sie ist oft jedoch nicht die beste Wahl, wenn es darum geht ein Modell zu bewerten (vgl. Grunert 2021, S. 21) Eine andere Möglichkeit ein Modell zu bewerten ist anhand einer Confusionmatrix (Wahrheitsmatrix oder Konfusionsmatrix).

		Vorhergesagte Klasse	
		Positiv	Negativ
Tatsächliche Klasse	Positiv	Richtig Positive (RP)	Falsch Negative (FN)
	Negativ	Falsch Positive (FP)	Richtig Negative (RN)

Abbildung 4: Beispielhafte Confusionmatrix
Quelle: In Anlehnung an (Raschka 2017, S. 194)

Die Genauigkeit eines Modells wird aus der Confusionmatrix (Abbildung 4) wie folgt berechnet:

$$Accuracy = \frac{RP + RN}{RP + FN + FP + RN}$$

Das Problem an der Accuracy ist, dass diese sehr hoch sein kann, obwohl eine große Zahl an falsch Klassifizierungen dabei ist. Je nach Datenverteilung kann ein Modell mit hoher Accuracy trotzdem schlecht sein. Wird ein Modell zur Vorhersage von Leukämie angewandt, welches 70 Personen richtig positiv klassifiziert (RP), jedoch auch 13 900 Menschen fälschlicherweise positiv klassifiziert (FP), so kann man immer noch bei 981 700 korrekt negativ klassifizierten (RN) Menschen bei einer Stichprobe, die aus 1 Mio. Menschen besteht, auf einen Accuracy Wert von 98 % kommen. Man würde dieses Modell aber trotzdem nicht als zuverlässig in seiner Vorhersage betrachten, da es schließlich lediglich 70 Menschen korrekt ihre Leukämie (RP), aber 13 900 Menschen fälschlicherweise (FP) Leukämie vorhergesagt hat (vgl. Grus 2015, S. 145-146).
Deshalb betrachtet man zusätzliche Metriken zum Evaluieren wie die Precision (Präzision), den Recall (Trefferquote) oder deren kombiniertes Maß den F1-Score (vgl. Raschka 2017, S. 193).
Diese werden dabei wie folgt berechnet:

$$Precision = \frac{RP}{RP + FP}$$

Die Precision gibt dabei Aufschluss darüber, wie gut das Verhältnis der richtigerweise positiv klassifizierten Beobachtungen im Vergleich zu allen positiv klassifizierten Beobachtungen ist (vgl. Alpaydin 2021, S. 635).

$$Recall = \frac{RP}{RP + FN}$$

Der Recall gibt Aufschluss über die Trefferquote eines Modells. Das bedeutet dieser gibt den Anteil der richtig positiv klassifizierten Beobachtungen zu allen Beobachtungen, die tatsächlich der positiven Klasse zugehören, an. Also sowohl zu den richtig positiv klassifizierten, als auch zu den fälschlicherweise negativ klassifizierten, welche ebenfalls positiv sein müssten (vgl. Grunert 2021, S. 21).

$$F1 - Score = \frac{2\ (Recall\ x\ Precision)}{(Recall\ +\ Precision)}$$

Beim F1-Score handelt es sich um einen gewichteten Mittelwert aus Precision und Recall (vgl. Grunert 2021, S. 21). Da es sich hierbei um eine Kombination aus Precision und Recall handelt, welches beide wichtige Evaluationsmetriken sind, findet er oft Anwendung als Vergleichsmaß in der Praxis (vgl. Raschka 2017, S. 196).

2.2 Condition Monitoring

2.2.1 Definition und Klassifikation

Unter dem Begriff Condition Monitoring (Zustandsüberwachung) versteht man das Erfassen von Daten, welche Aufschluss über den Maschinenzustand oder über den Zustand einzelner Maschinenkomponenten geben (vgl. Brecher/Klein/Lindner 2009, S. 118).
Laut dem Frauenhofer-Institut kann Condition Monitoring dabei vor allem entscheidende Informationen für Instandhaltungsstrategien liefern. Und somit eine bessere Prozessoptimierung und Prozesssteuerung ermöglichen (vgl. Fraunhofer-Institut für Keramische Technologien und Systeme o. D., S. 1). Die einfachste Form der Zustandsüberwachung ist dabei die Zustandsüberwachung von akustischen Signalen durch einen geschulten Menschen, welcher in der Lage ist, sogar kleine Änderungen im akustischen Signal festzustellen (vgl. Maytas 2016, S. 128). Deutlich verbreiteter ist heutzutage jedoch das Ermitteln des technischen Zustands einer Maschine mit Sensoren, welche Schwingungen, Vibrationen, Feuchtigkeit (vgl. Lambertz 2023), Temperaturen oder Überstrom erfassen (vgl. Irfan 2019, S. xvi).
Auch wenn Condition Monitoring oft im Zusammenhang mit Condition-Based Maintenance (Zustandsbasierter Instandhaltung) steht, so ist Condition Monitoring in erster Linie das Erfassen von Daten einer Maschine. Diese Daten müssen nicht zwangsläufig im Bereich des

Instandhaltungsmanagements genutzt werden (vgl. Mobley 1998, S. 38). Mögliche weitere Anwendungsgebiete sind nach Kolerus und Wassermann:

- Der Schutz vor Schäden an Maschine, Umwelt oder Mensch,
- Schutz vor Ausfällen,
- Sicherung der Produktion sowie,
- die Qualitätskontrolle
  (vgl. Kolerus/Wassermann 2017, S. 2).

2.2.2 Condition-Based Maintenance und Predictive Maintenance

Condition-Based Maintenance und Predictive Maintenance sind Teil des Instandhaltungsmanagements (vgl. Mobley 1998, S. 37). Ziel der Instandhaltung ist dabei, prinzipiell die Funktion und Leistungsfähigkeit einer Maschine oder Anlage zu gewährleisten. (vgl. Maytas 2016, S. 27) Die Instandhaltung kann dabei laut Nagel in drei Arten unterteilt werden:

- Corrective Maintenance (Korrigierende Wartung / Instandsetzung): die Maschine wird gewartet, wenn eine Instandsetzung notwendig ist, also ein Ausfall auftritt.
- Preventiv Maintenance (Vorbeugende Wartung / präventive Instandhaltung): die Maschine wird regelmäßig gewartet, auch wenn noch kein Ausfall auftritt, es wird präventiv gearbeitet.
- Condition-Based Maintenance / Predictive Maintenance: die Maschine wird ihrem aktuellen Zustand entsprechend gewartet
  (vgl. Nagel 2018, S. 114).

Dabei wird bei Condition-Based Maintenance die Lebensdauer von Maschinen verlängert, indem Veränderungen im Maschinenzustand, welche auf einen sich entwickelnden Fehler hinweisen, frühzeitig erkannt werden, und somit zum richtigen Zeitpunkt kurz vor einem Ausfall gehandelt werden kann (vgl. Maheswari/Gunasekharan 2019, S. 152). Im Gegensatz zur Preventiv Maintenance hat Condition-Based Maintenance den Vorteil, dass man sich nicht auf Statistiken über die durchschnittlichen Lebensdauern oder Ausfälle der jeweiligen Maschine verlassen muss (vgl. Mobley 1998, S. 37). Die Wartungsressourcen können dementsprechend besser priorisiert und optimiert werden ohne Gefahr von Ausfällen oder Ressourcenverschwendung (vgl. Nagel 2018, S. 114).
Obwohl sich Condition-Based Maintenance und Predictive Maintenance sehr ähnlich sind, da sie beide auf Condition Monitoring beruhen, gibt es Unterschiede. Grundsätzlich verfolgen beide

Ansätze dasselbe Ziel: die Reduzierung der Ausfallzeiten und die Optimierung von Ressourcen (vgl. Eisner 2022). Beim Condition-Based Monitoring werden jedoch aus dem erfassten Maschinenzustand lediglich gewisse Kennwerte und Grenzwerte abgeleitet, bei deren Erreichung Instandhaltungsmaßnahmen initiiert werden (vgl. Mühlnickel et al. 2018, S. 354). Bei Predicitve Maintenance hingegen geht es nicht nur darum mit Daten gewisse Grenzwerte festzulegen, sondern aus dem aktuellen Maschinenzustand auf zukünftig auftretende Fehler und Ausfälle zu schließen (vgl. Sullivan et al. 2010, S. 5.4).

2.2.3 Anomaly Detection

Anomaly Detection (Anomalieerkennung) gehört zu den wichtigsten Praktiken in der Industrie (vgl. Gamal et al. 2021, S. 1). Als Anomalien bezeichnet man Ausreißer, Ereignisse deren auftreten selten oder unwahrscheinlich ist, oder auch Abweichungen von einer Norm oder innerhalb einer Datenreihe (vgl. Morales-Forero/Bassetto 2019, S. 1).
Das Erkennen von Anomalien beschäftigt sich damit, Datenpunkte innerhalb von Daten ausfindig zu machen die nicht in das erwartete Muster der Daten passen (vgl. Chandola/Banerjee/Kumar 2009, S. 1). Im Rahmen der modernen Industrie bekommt die Anomalieerkennung eine immer wichtigere Schlüsselrolle (vgl. Pittino et al. 2020, S. 1). Dabei sammeln Unternehmen an einer Vielzahl von Stellen innerhalb des Produktionsprozesses Informationen und Daten, um in diesen Anomalien zu finden, die sich negativ auf den Produktionsprozess auswirken (vgl. Morales-Forero/Bassetto 2019, S. 1). Laut Nyguen et al. sind die vier wichtigsten Anwendungsbereiche der Anomalieerkennung in der modernen Produktion:

- Condition-Based Maintenance / Predictive Maintenance,
- Wearable Technology für Personal (Smartwatch, Smart clothing, etc.),
- Produktionsüberwachung und
- Echtzeit-Cybersicherheit
  (vgl. Nguyen et al. 2021, S. 6).

Die Anomalieerkennung ist mit dem Rauschen innerhalb von Daten verwandt. Anders als bei Datenrauschen sind die Abweichungen bei Anomalien interessant für die Analytiker der Daten und es geht nicht darum, diese aus den Daten zu entfernen, sondern diese ausfindig zu machen (vgl. Chandola/Banerjee/Kumar 2009, S. 2-3).

Generell gibt es zwei Arten der Anomalieerkennung, eine manuelle und eine automatische Erkennung. Da es sich bei der manuellen Erkennung durch Personal um eine repetitive und fehleranfällige Aufgabe handelt, hat die Entwicklung von automatischen Systemen zur Anomalieerkennung innerhalb der Produktion deutlich an Popularität gewonnen (vgl. Gamal et al. 2021, S. 1-2). Deren Ansatz besteht dabei darin, einen Normalbereich zu definieren. Fällt eine Beobachtung dann nicht in den Normalbereich, wird sie als Anomalie deklariert. In der Praxis bereitet jedoch gerade das Definieren dieses normalen Bereichs große Schwierigkeiten und gerade im Grenzbereich zwischen Normalbereich und Anomaliebereich kommt es leicht zu Falschdeklarationen (vgl. Chandola/Banerjee/Kumar 2009, S. 3).

In der Lösung dieses Problems der automatisierten Anomalieerkennung haben sich vor allem Technologien aus dem Bereich des Machine Learnings in diversen Forschungsarbeiten durchgesetzt (vgl. Nguyen et al. 2021, S. 13). Dabei kann Supervised Learning, Unsupervised Learning, als auch Semi-Supervised Learning als Mischform zum Einsatz kommen. Wobei in der Praxis am häufigsten Unsupervised Learning zum Einsatz kommt, was dem einfachen Fakt geschuldet ist, dass oft nur Daten aus dem Normalbereich vorliegen (vgl. Chandola/Banerjee/Kumar 2009, S. 9-10). Da im Zuge dieser Arbeit der Zugriff auf Daten aus dem Normalbereich sowie aus dem Anomaliebereich zur Verfügung standen, wird innerhalb des empirischen Teils der Arbeit mit Modellen aus dem Bereich des Supervised Learnings gearbeitet.

3 Empirische Untersuchung

3.1 Forschungsdesign

Im Zuge dieser Arbeit sollen die Fragen, wie durch Machine Learning die eingestellte Nutzlast eines Industrieroboters auf die Richtigkeit der Einstellung überprüft werden kann und welche Faktoren eine solche Vorhersage beeinflussen, mithilfe eines quantitativen Experiments beantwortet werden.
Für dieses Experiment werden Daten des sechsachsigen Industrieroboters UR10 der Firma „Universal Robots“ erhoben und anschließend für das Training von Machine Learing-Modellen und deren Evaluation genutzt. Die gewonnen Daten werden genutzt, um drei verschiedene Supervised Machine-Learning-Modelle für die Klassifizierung von Daten zu Trainieren und zu vergleichen. Bei den drei unterschiedlichen Modellarten handelt es sich um den Decision Tree, das k-Nearest Neighbor-Modell, und die logistische Regression. Dabei durchläuft jede Modellart dieselben Abläufe zur Ermittlung der Genauigkeitsfaktoren. So kann eine Aussage über die Einflussfaktoren, über alle Modelarten hinweg, getroffen werden.
Die komplette Arbeit mit den Daten findet dabei in der Programmiersprache Python statt. Für die Anwendung der Machine Learning-Modelle wurde die Open Source Bibliothek „scikit-learn“ verwendet, welche Python um die Machine Learning-Funktionalitäten erweitert.
Im ersten Schritt der Experimentauswertung wird stets ein Referenzmodell geschaffen, um festhalten zu können, welche Faktoren wie viel Genauigkeitsanstieg zum Referenzmodell bringen. Im nächsten Schritt wird überprüft, wie sich die Genauigkeitswerte der Modelle mit unterschiedlichen Features verhalten. Im dritten Schritt wird das Modell beim Training mit einigen Parametern getestet, welche dem Feintuning der Modelle dienen.

3.1.1 Datenerhebung

Bei der Datenerhebung werden ausschließlich vom Roboter aufgezeichnete Wert erhoben, es kommt dabei keine externe Sensorik zum Einsatz. Die Erhebung erfolgt mithilfe der von Universal Robots bereitgestellter Log-Software, welche mittels direkter Verbindung zum Roboter, über ein LAN-Kabel, die Daten aufzeichnet. Die von der Log-Software aufgezeichneten Daten können aus dem Programm im csv-Dateiformat exportiert werden.
Die Datenerhebung findet zu zwei unterschiedlichen Terminen statt, wobei beim ersten Termin zwei Datensätze erhoben werden. Für die Aufzeichnungen werden vom Roboter unterschiedliche Fahrtszenarien mit einer unterschiedlichen Anzahl an Punkten in einer Schleife abgefahren. Beim ersten Termin entstehen zwei Datensätze. Beim zweiten Termin entstehen sieben Datensätze. Für die

Versuche entstehen so insgesamt neun Datensätze mit einer insgesamten Anzahl von etwa 1,4 Mio. Datenzeilen.

3.1.2 Datenvorbereitung

Um mit Supervised Learning-Modellen arbeiten zu können, benötigen die Modelle eine Zielvariabel. Deshalb besteht jeder Datensatz aus fünf Subdatensätzen, bei welchen vier eine falsch eingestellte Payload-Einstellung besitzen und einer die richtige Payload-Einstellung besitzt. Die Aufteilung der fünf Subdatensätze sieht wie folgt aus:

- Korrekt eingestellte Payload-Einstellung
- Payload-Einstellung 1500g zu niedrig eingestellt
- Payload-Einstellung 750g zu niedrig eingestellt
- Payload-Einstellung 1500g zu hoch eingestellt
- Payload-Einstellung 750g zu hoch eingestellt

Die fünf Subdatensätze bekommen in Python eine weitere Datenspalte mit dem Namen „payload_setting“. Diese Spalte wird für den Subdatensatz mit der richtigen Einstellung mit dem Wert „true“ aufgefüllt und für die vier Subdatensätze mit der falschen Einstellung mit dem Wert „false“. Anschließend werden die vier Subdatensätze zu einem gemeinsamen Datensatz verbunden. Dieses Vorgehen wird für alle neun Datensätze wiederholt. So besitzt jeder Datensatz das binäre Merkmal „payload_setting“, was die Machine Learning-Modelle zum Trainieren nutzen und für die Evaluation eines Modells benötigt wird.

3.1.3 Datenauswertung Modellauswertung

Für die Beurteilung der Machine Learning Models wird der „Classification report“ von scikit-learn herangezogen. Es handelt sich dabei um eine tabellarische Darstellung der Precision, dem Recall und dem F1-Score. Alle drei Metriken werden dabei pro Klasse und über beide Klassen hinweg als Durchschnitt angegeben. Als Durchschnitt werden die Metriken jeweils als Macro average (Makro-Durchschnitt) und als Weighted average (gewichteter Durchschnitt) angegeben. Bei den Weighted average Metriken wird also der Score pro Klasse nach der Anzahl der Beobachtungen gewichtet. In den erhobenen Daten herrscht jedoch ein Ungleichgewicht, da wie in Kapitel 3.1.2 beschrieben, jeder Datensatz aus einem Subdatensatz mit korrekter Payload-Einstellung und vier mit falscher Payload-Einstellung besteht. Ziel ist jedoch ein Modell, welches beide Klassen zuverlässig vorhersagen kann.

Somit werden primär die Makrometriken genutzt, bei welchen der Durchschnitt nicht nach Anzahl der Beobachtungen gewichtet ist.

3.2 Auswertung und Ergebnisse

3.2.1 Datenbetrachtung und Feature-Selection

Zur Datenbetrachtung gehört auch das Auseinandersetzen mit den erhobenen Daten. Dabei spielt ein grundlegendes Verständnis für alle erhobenen Daten und ihre Bedeutung im weiteren Verlauf eine wichtige Rolle. Insbesondere bei der Auswahl der Merkmale, im Machine learning auch oft Features genannt, kann ein Datenverständnis hilfreich sein. Jede csv-Datei beinhaltet mindestens die Folgenden 112 Spalten (Tabelle 1):

Tabelle 1: Auflistung aller Aufgezeichneten Merkmale inklusive ihrer Bedeutung
Quelle: Eigene Darstellung
Quelle-Bedeutung: (Universal Robots Support 2023)

Spaltenname	Bedeutung
timestamp	Die vergangene Zeit seit Beginn der Log-Aufzeichnung
Timestamp_Realtime	Zeitstempel der Realzeit innerhalb der aktuellen Zeitzone inklusive Datum.
script_control_line	Zeigt an, ob ein anderer Prozess die Handlungsfähigkeit des Roboters blockiert.
target_q_*(0-5)*	Angestrebte Gelenkposition *(Gelenk 0-5)*
actual_q_*(0-5)*	Tatsächliche Gelenkposition *(Gelenk 0-5)*
target_qd_*(0-5)*	Zielgeschwindigkeit des Gelenks *(Gelenk 0-5)*
actual_qd_*(0-5)*	Geschwindigkeit des Gelenks *(Gelenk 0-5)*
target_qdd_*(0-5)*	Zielbeschleunigung des Gelenks *(Gelenk 0-5)*
target_current_*(0-5)*	Ziel-Stromstärke des Gelenkmotors *(Gelenk 0-5)*
actual_current_*(0-5)*	Stromstärke des Gelenkmotors *(Gelenk 0-5)*
actual_current__window_*(0-5)*	Fenster, in welchem sich die Stromstärke bewegen darf, ohne dass das Programm abbricht. Ausgehend von der Ziel-Stromstärke die Hälfte des Wertes in die positive Richtung und in die negative Richtung.
joint_control_output_*(0-5)*	Steuerstrom des Gelenks *(Gelenk 0-5)*
target_moment_*(0-5)*	Zielmoment (Drehmoment) des Gelenks *(Gelenk 0-5)*
target_TCP_pose_*(0-5)*	Zielkoordinate des Werkzeugs *(0-5 entspricht: X, Y, Z, rotationX, rotationY, rotationZ)*
actual_TCP_pose_*(0-5)*	Koordinate des Werkzeugs *(0-5 entspricht: X, Y, Z, rotationX, rotationY, rotationZ)*

target_TCP_speed_*(0-5)*	Zielgeschwindigkeit des Werkzeugs auf den Achsen *(0-5 entspricht: X, Y, Z, rotationX, rotationY, rotationZ)*
actual_TCP_speed_*(0-5)*	Geschwindigkeit des Werkzeugs auf den Achsen *(0-5 entspricht: X, Y, Z, rotationX, rotationY, rotationZ)*
actual_TCP_force_*(0-5)*	Verallgemeinerte Kräfte des Werkzeugs auf den Achsen *(0-5 entspricht: X, Y, Z, rotationX, rotationY, rotationZ)*
ft_raw_wrench_*(0-5)*	Rohe Kraft- und Drehmomentmessung im Werkzeugflansch/Gelenkausgangs des Gelenks. *(Gelenk 0-5)*
joint_temperatures_*(0-5)*	Temperatur innerhalb des Gelenks. *(Gelenk 0-5)*
speed_scaling	Geschwindigkeitsskalierung. Auftretende Werte null und eins, null tritt auf, wenn der Roboter noch kein laufendes Programm abarbeitet.
elbow_position_*(x, y, z)*	Position des Roboter-Ellbogens auf der Achse *(x, y, z)*
elbow_velocity_*(x, y, z)*	Geschwindigkeit des Roboter-Ellbogens auf der Achse *(x, y, z)*

Einige Daten aus der zweiten Datenerhebung mit dem UR10 beinhalten zusätzliche Spalten mit weiteren Informationen. Diese werden jedoch entfernt und nicht beachtet, um über alle Daten hinweg einen übereinstimmenden Datenstand zu besitzen. Zunächst werden alle dieser Merkmale in grafischer Form dargestellt, um mögliche Abhängigkeiten, und damit geeignete Features für das Trainieren eines Machine Learning Models bereits bei grafischer Darstellung, zu erkennen.

3.2.1.1 Visualisierung

Bei der Visualisierung in Diagramform stellten sich, im Bereich der Stromstärke und der Temperatur, einige vielversprechende Merkmale heraus.

Ist die Payload zu niedrig eingestellt (Abbildung 5), fällt die tatsächliche Stromstärke, die der Roboter innerhalb der Elektromotoren aufbringen muss, deutlich höher aus als in seinen Berechnungen, also der erwarteten Stromstärke. Dies ist in der Abbildung deutlich durch den gelb gekennzeichneten Differenzbereich zu erkennen.

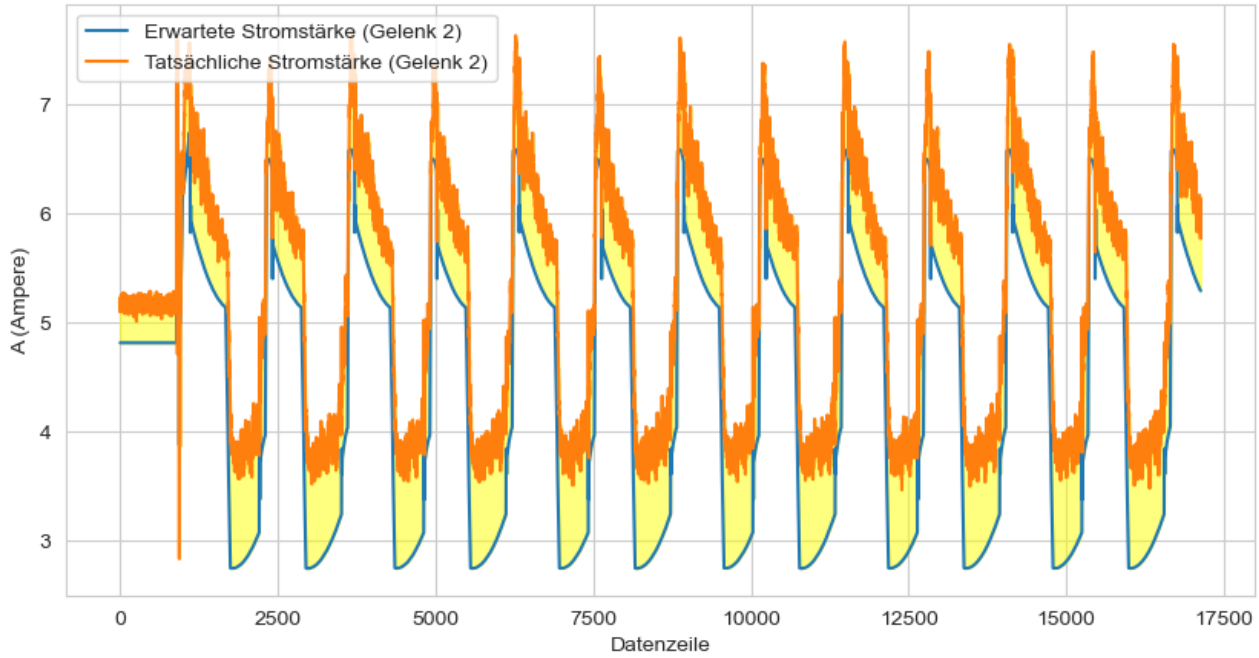

Abbildung 5: Erwartete und tatsächliche Stromstärke in Gelenk 2 mit 1,5kg zu niedrig eingestellter Payload des ersten Datensatz
Quelle: Eigene Darstellung

Ist die Payload richtig eingestellt (Abbildung 6), fällt der gelbe Bereich, und damit die Differenz zwischen erwarteter und tatsächlicher Stromstärke deutlich geringer aus. Lediglich zu Beginn der Aufzeichnung existiert ebenfalls eine recht hohe Differenz.

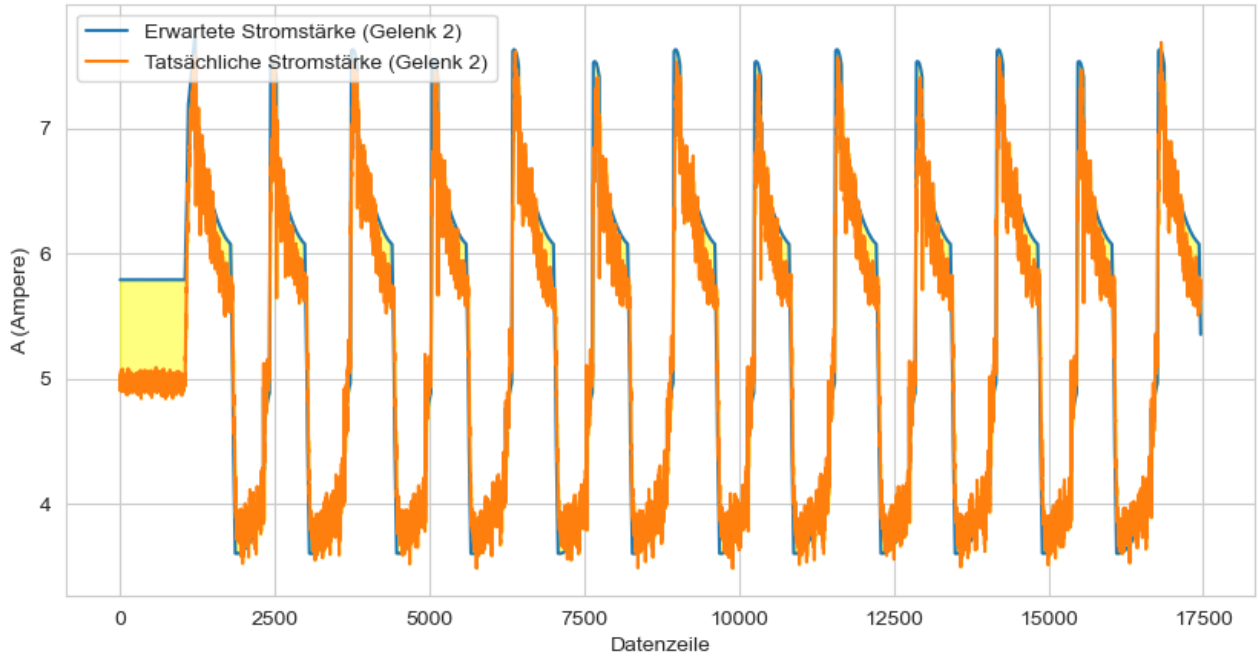

Abbildung 6: Erwartete und tatsächliche Stromstärke in Gelenk 2 mit richtig eingestellter Payload des ersten Datensatz
Quelle: Eigene Darstellung

Ist die Payload zu hoch eingestellt (Abbildung 7), fällt der Differenzbereich wieder deutlich größer aus. Anders als bei der zu niedrig eingestellten Payload, liegt jetzt jedoch die vom Roboter erwartete Stromstärke über der tatsächlichen Stromstärke.

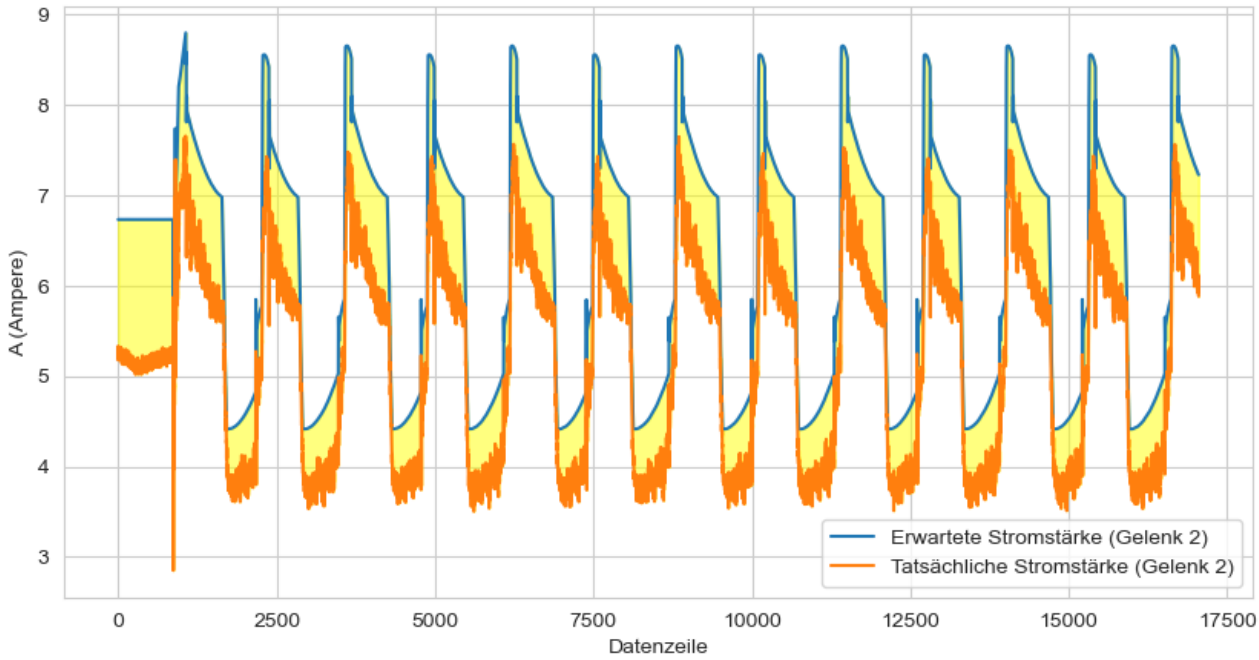

Abbildung 7: Erwartete und tatsächliche Stromstärke in Gelenk 2 mit 1,5kg zu hoch eingestellter Payload des ersten Datensatz
Quelle: Eigene Darstellung

Die Temperaturen innerhalb der Gelenke lassen in der visuellen Vorbetrachtung (Abbildung 8) zunächst auch auf einen Zusammenhang zwischen eingestellter Payload und Gelenktemperatur schließen. Es scheint so, als würde mit einer höher eingestellten Payload auch eine stetig höhere Temperatur in den Gelenken herrschen.

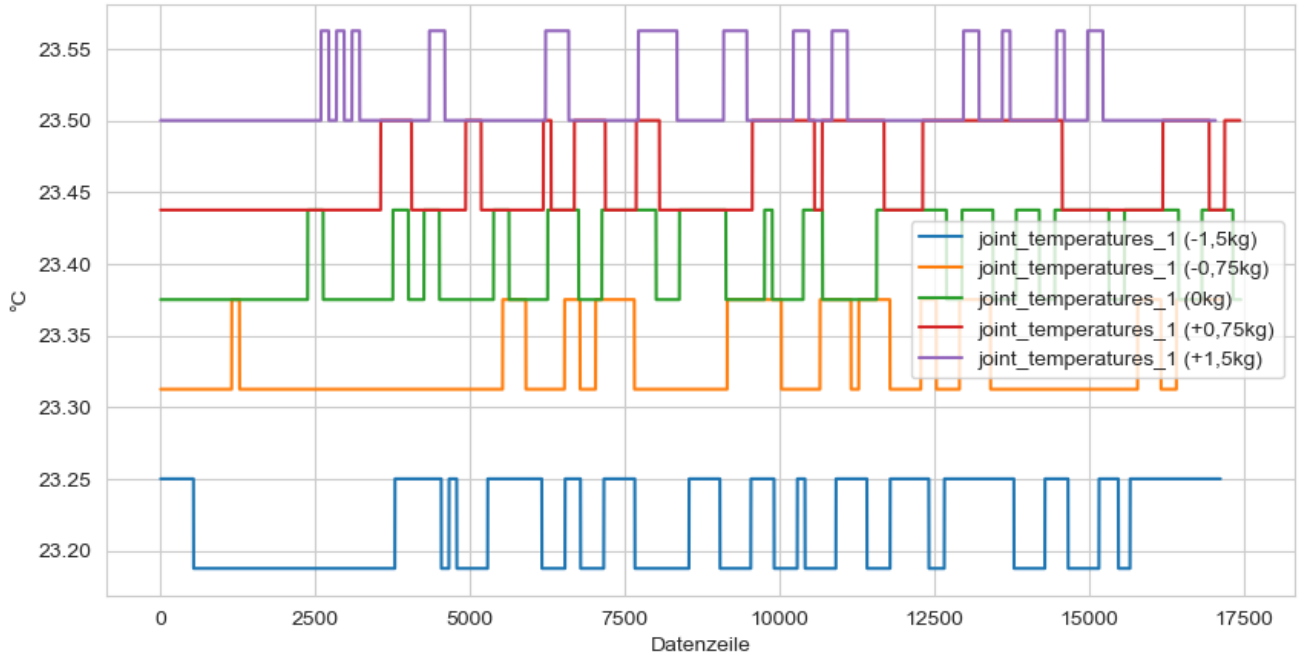

Abbildung 8: Temperaturen in Gelenk 2 bei verschiedenen Payload-Einstellungen des ersten Datensatzes
Quelle: Eigene Darstellung

Bei einer weiteren Datenerhebung wird jedoch klar, dass die Temperatur keine verlässliche Auskunft zu geben scheint. Bei der ersten Datenerhebung (Abbildung 8) steigt die Temperatur zwar mit der Payload Einstellung kontinuierlich, sie steigt jedoch auch, was auf den ersten Blick nicht erkenntlich ist, mit der Reihenfolge der Datenerhebung und somit der Nutzungsdauer. Die Vermutung, dass die Nutzungsdauer einen Einfluss auf die Temperatur hat, erhärtet sich ebenfalls bei der zweiten Datenerhebung (Abbildung 9). Auch hier steigt die Temperatur mit der Reihenfolge der Datenerhebung. Anders als bei der ersten Datenerhebung wurde jedoch die Aufzeichnungsreihenfolge geändert und somit passt der Temperaturanstieg nichtmehr zur höheren Payload-Einstellung. Es herrscht nun bei der Temperaturaufzeichnung mit der richtig eingestellten Payload die niedrigste Temperatur und bei der Temperaturaufzeichnung mit der zu niedrig eingestellten Payload, die mittlere Temperatur der drei Kurven.

Weiter lässt sich in der zweiten Datenerhebung ebenfalls ein Temperaturanstieg innerhalb der einzelnen Kurven beobachten. Dieser Anstieg innerhalb der einzelnen Aufzeichnungen lässt sich in der ersten, circa sieben Mal kürzeren, Datenerhebung, nicht erkennen.

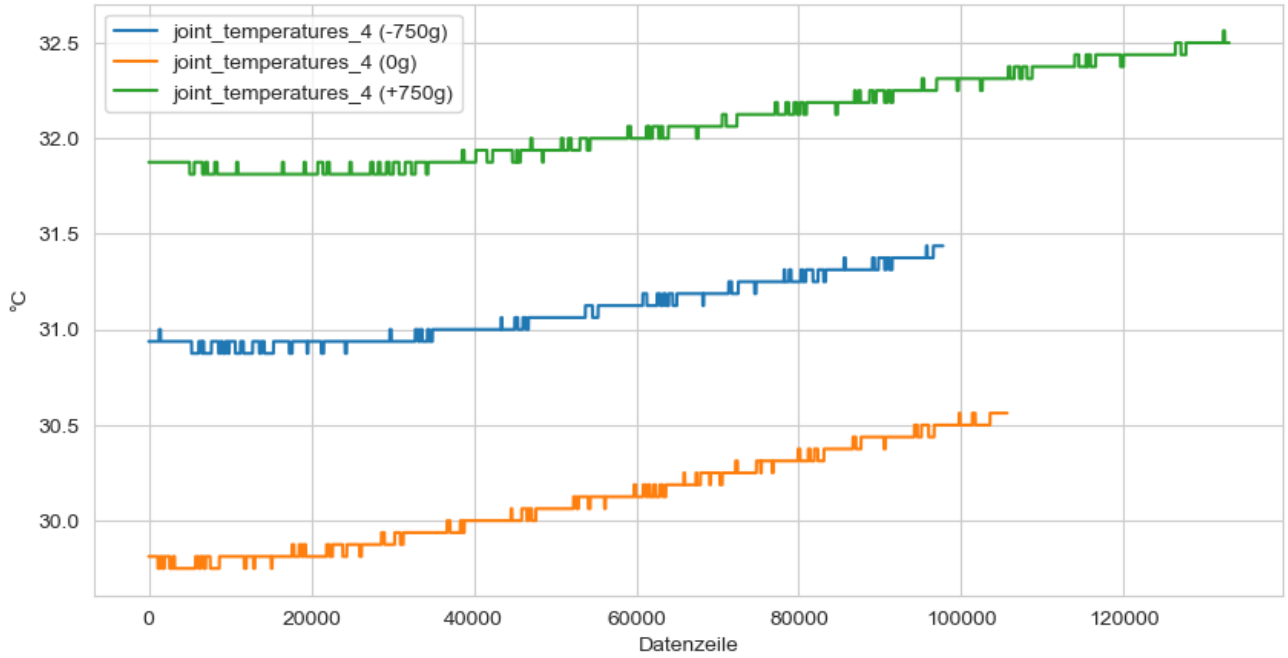

Abbildung 9: Temperaturen in Gelenk 5 über eine längere Zeitperiode des neunten Datensatzes
Quelle: Eigene Darstellung

Alle weiteren Daten scheinen in der visuellen Vorbetrachtung zunächst unauffällig. Sie werden deshalb jedoch nicht von vorne hinein ausgeschlossen, um den Modellen zu Beginn alle Möglichkeiten der Merkmalsauswahl zu überlassen und diese dann sukzessiv zu verfeinern.

3.2.1.2 Feature-Engineering

Bevor jedoch eine Auswahl von Features getroffen wird, werden den Daten erst neue Features hinzugefügt. Überall wo es einen erwarteten Wert des Roboters und einen tatsächlichen Wert gibt, welchen der Roboter aufzeichnet, bietet es sich an, eine Differenz zwischen Erwartungswert und tatsächlichem Wert zu bilden. Es werden für die folgenden Merkmale Differenzen gebildet:

- current_*(0-5)*
- q_*(0-5)*
- qd_*(0-5)*
- TCP_pose_*(0-5)*
- TCP_speed_*(0-5)*

Neben den Differenzen pro Gelenk wird auch der Mittelwert der Differenzen über alle Gelenke hinweg gebildet:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Da jedoch je nach Programm des Roboters die Differenz in unterschiedlichen Gelenken höher ausfallen kann, wird neben dem Mittelwert auch die Standardabweichung vom Mittelwert berechnet, um Ausreißer in einzelnen Gelenken besser identifizieren zu können:

$$\sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})}$$

3.2.1.3 Feature-Selection

Neben der visuellen Vorbetrachtung bietet Scikit-learn auch eine Reihe von Funktionen, die dabei helfen kann, die besten Features für Machine Learning im Vorhinein zu berechnen und auszuwählen. Diese Funktionen geben jedoch nur eine grundlegende Richtung und Idee vor. Die optimale Feature Auswahl unterscheidet sich von Machine Learning Algorithmus zu Machine Learning Algorithmus und kann meistens nur durch praktisches Testen ermittelt werden. Lediglich Erfahrung und Logik wären zudem eine weiter Option.

Bevor die besten Features ausgewählt werden, sollen zunächst Features die konstante Werte enthalten und Features ohne Informationsgehalt herausgefiltert werden. Mithilfe einer Funktion, die uns einen Schwellenwert für Abweichung innerhalb jedes Feature festlegen lässt, können wir alle Features, die lediglich einen konstanten Wert abbilden, herausfiltern, indem wir diesen Schwellenwert auf null setzen. Es fallen zunächst vier Features heraus, inklusive „actual_current_window_4“ und „actual_current_window_5“. Deshalb wird ebenfalls „actual_current_window_0-3“ entfernt, da sie nahezu konstante Werte besitzen und zur selben Klasse gehören. Zudem werden alle Temperaturfeatures entfernt. Diese stehen zwar in einem starken Zusammenhang mit der Payload. Jedoch wurde dieser Zusammenhang, wie in Kapitel 3.2.1.1 der Visualisierung erläutert, fälschlicherweise bei der Datenerhebung erzeugt. Ebenfalls entfernt werden die Zeitfeatures „timestamp“ und „Timestamp_Realtime“. Diese Features könnten bei echten Daten aus einer Fabrik zwar im Zusammenhang mit der richtigen Payload-Einstellung stehen. Bei den hier, unter Laborbedingungen, erhobenen Daten, stehen die Uhrzeit und der Zeitstempel aber nicht im Zusammenhang mit der Richtigkeit der Payload-Einstellung.

Nach dem Entfernen dieser Features handelt es sich jedoch immer noch um eine recht hohe Anzahl von Features. Um diese Anzahl weiter zu reduzieren, werden zwei Funktionen von Scikit-learn, zur

Berechnung der optimalen Features, genutzt. Zum einen wird „mutual_info_classif“ genutzt, welche eine Punktzahl für den gemeinsamen Informationsgehalt zwischen Feature und Target bei Klassifikation berechnet (vgl. Holbrook/Cook o. D.). Zum anderen wird „f_classif“, welche einen F-Score durch Bestimmen des Verhältnisses von erklärter zu ungeklärter Varianz bildet, genutzt (vgl. IBM o. D.-a). Beide Funktionen bekommen den Parameter k=20 um die 20 Features mit der höchsten Punktzahl auszugeben.

Beide Funktionen liefern größtenteils unterschiedliche Features (Tabelle 2), es gibt jedoch auch vier Features die von beiden Funktionen als gute Feature identifiziert werden. Diese übereinstimmenden Features sind in der Tabelle grün gekennzeichnet.

Tabelle 2: Die besten 20 Features nach „mutual_info_classif“- und „f_classif“-Funktion und ihre Überschneidungen
Quelle: Eigene Darstellung

mutual_info_classif	f_classif
target_current_1	target_q_3
target_current_2	actual_q_3
target_current_3	target_moment_2
joint_control_output_1	target_TCP_pose_0
joint_control_output_2	target_TCP_pose_2
joint_control_output_3	target_TCP_pose_3
target_moment_1	target_TCP_pose_4
target_moment_2	actual_TCP_pose_0
target_moment_3	actual_TCP_pose_2
target_moment_4	actual_TCP_pose_3
target_moment_5	actual_TCP_pose_4
actual_TCP_force_3	actual_TCP_force_2
actual_TCP_force_4	actual_TCP_force_3
ft_raw_wrench_2	actual_TCP_force_4
speed_scaling	actual_TCP_force_5
current_diff_1	ft_raw_wrench_0
current_diff_2	ft_raw_wrench_2
current_diff_3	ft_raw_wrench_3
current_all_diff_mean	ft_raw_wrench_4
current_all_diff_S_deviation	current_all_diff_S_deviation

Insgesamt liefern sowohl die Feature-Selection als auch die visuelle Vorbetrachtung zwar nicht die optimalen Features, sie geben jedoch eine grobe Idee, welche Features im weiteren Verlauf nützlich

sein können und potenziell gute Ansatzpunkte für die Verfeinerung der Machine Learning Modelle bieten könnten.

3.2.2 Decision Tree

3.2.2.1 Referenzmodell und Trainingsdaten

Zunächst wird ein Decision Tree ohne jegliche Parameterauswahl oder Einschränkung der Features trainiert, um einen Referenzwert für spätere Verfeinerungen des Models zu schaffen. Der erste Decision Tree wird dabei mit dem ersten Datensatz trainiert. Bereits die Größe des Baumes und die Anzahl der Verzweigungen (Abbildung 10) lässt auf eine Überanpassung des Modells schließen.

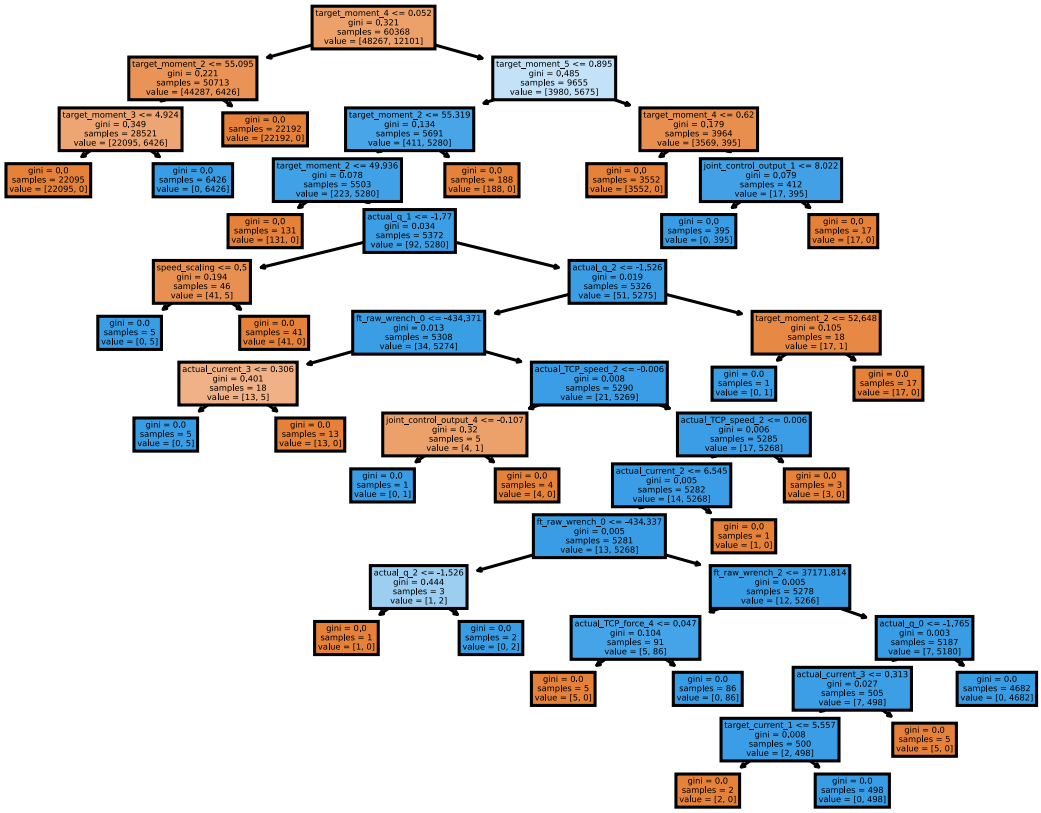

Abbildung 10: Decision Tree des Referenzmodell mit allen Features
Quelle: Eigene Darstellung

Auch der Klassifikationsreport mit den Testdaten bestätigt diese Annahme, da alle Kennzahlen einen Wert von 100 % aufweisen. Bei der Gegenprüfung mit den Validierungsdaten (Tabelle 3) nimmt die Genauigkeit auf 73 % ab. Deutlich nehmen jedoch vor allem Präzision und Recall ab, besonders beim Erkennen der richtigen Payload Einstellung.

Tabelle 3: Klassifikationsreport des Decision Tree-Referenzmodells mit den Validierungsdaten
Quelle: Eigene Darstellung

	precision	recall	f1-score
false	0.75	0.93	0.83
true	0.55	0.22	0.32
accuracy			0.73
macro avg	0.65	0.58	0.58
weighted avg	0.70	0.73	0.69

Insgesamt scheint das Modell nun immer noch akzeptabel Datenpunkte mit falschen Payload-Einstellungen zu identifizieren. Die Identifizierung von Daten mit richtig eingestellter Payload ist mit weniger als 40 % im F1-Score jedoch kein sehr gutes Ergebnis.

Fehlende Präzision und Recall für die Klasse „true“ lassen vermuten, dass die Trainingsdaten nicht genug Informationen liefern, um die Klasse besser vorhersagen zu können. Um dies zu überprüfen, werden dem ersten Datensatz einige Datenzeilen, mit der Klasse „true“, aus dem achten Datensatz hinzugefügt, um ein annäherndes Verhältnis von eins zu eins zwischen den Ausprägungen „true“ und „false“ zu schaffen. Erstaunlicherweise stellt sich bei der Validierung (Tabelle 4) dieses neu trainierten Modells heraus, dass sich die Genauigkeit massiv auf gerade einmal 31% Vorhersagengenauigkeit verschlechtert. So nimmt zwar die Präzision für die Klasse „false“ etwas zu und der Recall der Klasse „true“ steigt enorm, jedoch geschieht dies zu Lasten der Präzision. Betrachtet man die F1-Scores beider Klassen so stellt man fest, dass das Model deutlich an Vorhersagequalität verliert.

Tabelle 4: Klassifikationsreport des Decision Tree-Referenzmodells, trainiert mit ausgeglichener Anzahl an Ausprägungen des Features „payload_setting“. Überprüft mit den Validierungsdaten
Quelle: Eigene Darstellung

	precision	recall	f1-score
false	0.90	0.10	0.17
true	0.30	0.97	0.45
accuracy			0.34
macro avg	0.60	0.53	0.31
weighted avg	0.73	0.34	0.25

Eventuell war nicht nur die Anzahl der Informationen ein Problem, sondern vielmehr auch die Qualität dieser Informationen, also die Wahl der Features. Eine Möglichkeit besteht darin, dass das Model mit allen Features überfordert ist und aus der Masse der Informationen es nicht schafft, die brauchbarsten herauszufiltern. Da durch einen Ausgleich der Anzahl von „true“ und „false“ des Features „payload_setting“ keine Verbesserung des Referenzmodels erreicht werden konnte, bleibt der erste Datensatz als Trainingsdatensatz für zukünftige Bäume bestehen.

3.2.2.2 Berechnete Feature-Selection

Mit der zweiten Reihe an Decision Trees wird überprüft, ob und wie viel uns die Feature-Selection aus Kapitel 3.2.1.3 im Vergleich zum ersten Decision Tree an Genauigkeit in der Vorhersage bringt. Man testet hierfür beide Listen an Features aus den unterschiedlichen Funktionen, „mutual_info_classif“ und „f_classif“, nacheinander, um ebenfalls einen Unterschied zwischen den beiden Feature-Selection Methoden feststellen zu können.

Der Decision Tree mit der „mutual_info_classif“ Featureliste konzentriert sich vor allem auf das Zieldrehmomente (target_moment) der unterschiedlichen Gelenke und besitzt deutlich mehr Verzweigungen als der Baum mit der „f_classif“ Featureliste.

Der Decision Tree mit der „f_classif“ Featureliste arbeitet vor allem mit den angestrebten und tatsächlichen Positionen (target_q, actual_q) der Gelenke als Klassifizierungsfeatures.

Ein Modell kann sich im Vergleich zum Modell mit allen Features in der Genauigkeit verbessern (Abbildung 11). Das Modell aus der „mutual_info_classif“ Featureliste verbessert sich um drei Prozentpunkte auf 75 % Genauigkeit. Es kann sich außerdem in den F1-Scores beider Klassen steigern, in dem der Klasse „true“ sogar beachtlich von 36 % auf 42 %, was zu einer gesamten Steigerung des F1-Scores (makro) auf 63 % führt.

Das Modell aus der „f_classif“ Featureliste hingegen kann sich in der reinen Genauigkeit nicht verbessern. Zusätzlich nehmen Recall und F1-Score (makro) ab und machen dieses Modell somit weniger zuverlässig als das Referenzmodell, mit allen Features. Lediglich die Precision steigt auf 77 %.

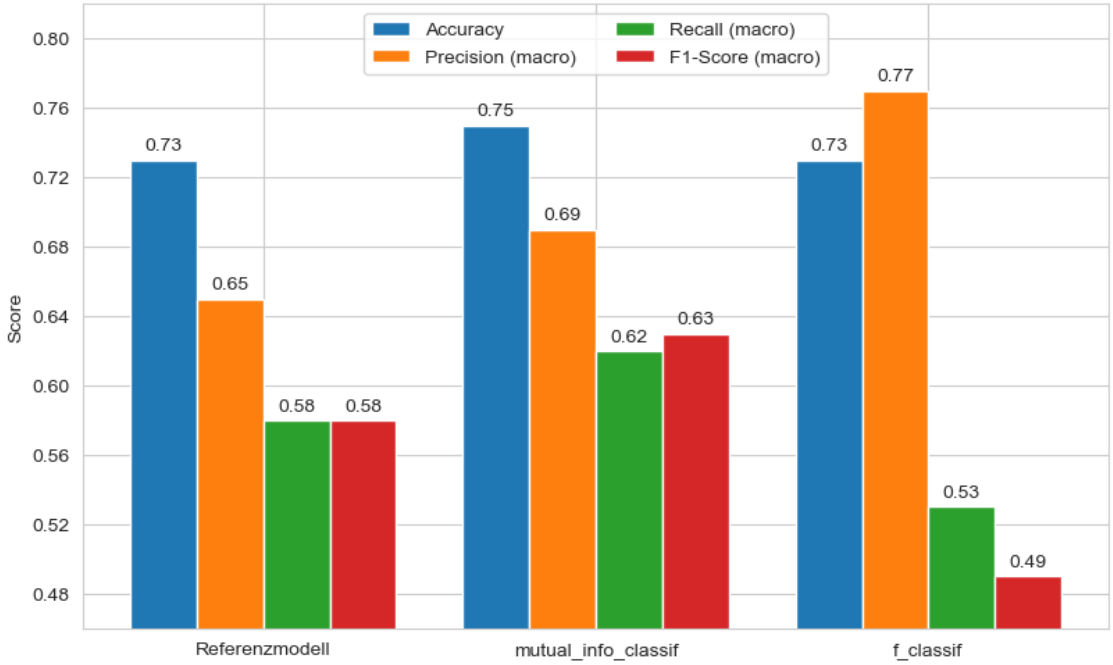

Abbildung 11: Evaluation der berechneten Feature-Selection Modelle im Vergleich zum Referenzmodell (Decision Tree)
Quelle: Eigene Darstellung

3.2.2.3 Manuelle Feature-Selection

Da die berechnete Feature-Selection zwar merklich bessere Ergebnisse geliefert hat als ein Modell mit allen vorhandenen Features, aber gerade der F1-Score noch ausbaufähig ist, wird im nächsten Schritt versucht durch eigene Featurezusammenstellungen noch bessere Ergebnisse zu erzielen. Die eigens zusammengestellten Feature-Sets basieren dabei auf der Logik, dass im Idealfall anhand von Differenzen aller Art Unterschiede erkennbar werden können. Dies wurde bereits in der visuellen Vorbetrachtung in Kapitel 3.2.1.1 anhand der Stromstärkedifferenzen erkennbar.
Auf Grundlage dessen sollen die nachfolgenden fünf Feature-Sets genutzt werden, um ein Modell zu trainieren und mit den Validierungsdaten zu evaluieren:

- Set 1 bestehend aus den Stromstärkedifferenzen zwischen erwartet und tatsächlich, sowie der daraus resultierenden Durchschnittsdifferenz.
- Set 2 bestehend aus den Stromstärkedifferenzen zwischen erwartet und tatsächlich, sowie der Standardabweichung der Durchschnittsdifferenz.
- Set 3 bestehend aus den Stromstärkedifferenzen zwischen erwartet und tatsächlich, sowie der daraus resultierenden Durchschnittsdifferenz und deren Standardabweichung.
- Set 4 bestehend aus allen in Kapitel 3.2.1.2, durch Feature-Engineering erschaffenen Durchschnittsdifferenzen von Soll und Ist, sowie deren Standardabweichungen

- Set 5 bestehend aus allen in Kapitel 3.2.1.2 durch Feature-Engineering erschaffenen Durchschnittsdifferenzen von Soll und Ist, sowie deren Standardabweichungen. Ausgenommen der Durchschnittsdifferenz und Standardabweichung der Stromstärke.

Set 5 liefert jedoch mit einer Genauigkeit von 63 %, Präzision und Rückruf von 50 %, sowie einem F1-Score von 49 %, ernüchternde Ergebnisse und schneidet damit sogar schlechter ab als eines der Modelle mit den berechneten Feature-Sets. Es fällt somit als potenzielles finales Feature-Set raus. Somit bleiben noch die Feature-Sets eins bis vier (Abbildung 12). Diese schaffen es jedoch in allen Evaluationsmetriken die berechneten Feature-Sets zu übertreffen.

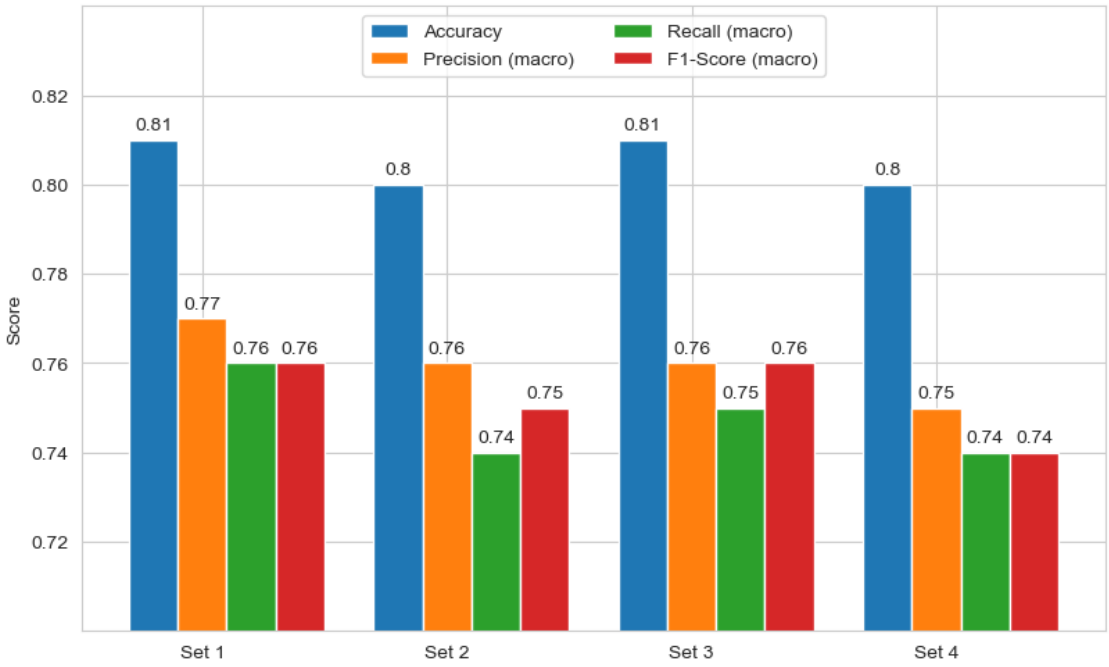

Abbildung 12: Evaluation verschiedener Decision Tree-Modelle auf Basis verschiedener Feature-Sets
Quelle: Eigene Darstellung

Erstaunlich ist, dass das Set 1 mit der normalen Durschnittsdifferenz der Stromstärke sogar minimal besser abschneidet als Set 2 mit der Standardabweichung der Stromstärkendifferenz. Insgesamt scheint es, über alle Sets hinweg, dass sobald eines der beiden Features Durchschnittsdifferenz der Stromstärke oder deren Standardabweichung als Feature beim Trainieren des Models genutzt wird, dieses recht zuverlässig in der Vorhersage ist.
Da Set 1 (Stromstärkedifferenz über alle Gelenke hinweg und die Durchschnittsdifferenz aus allen Gelenken) das beste Ergebnis liefert, dient dieses nun als Ausgangslage für den weiteren Verlauf.

3.2.2.4 Hyperparametertuning

Jeder Machine Learning Algorithmus besitzt beim Trainieren einige Übergabeparameter, welche dem Modell übergeben werden können. Diese werden im Machine Learning auch Hyperparameter genannt. In der Regel wählen die Modelle entweder automatisch den optimalen Wert oder sie besitzen einen Ausgangswert, der in den meisten Fällen gute Ergebnisse liefert.
Im Nachfolgenden wird versucht einige der optimalen Hyperparameter herauszufinden, um so das Modell in seiner Vorhersagegenauigkeit noch weiter zu verbessern.
Zu Beginn wird mit dem Parameter „criterion“, welcher die Werte „gini“ oder „entropy“ akzeptiert, gestartet. Es handelt sich dabei um zwei unterschiedliche Berechnungsarten, die berechnen wann ein Baum eine Abzweigung erhält. Außerdem wird im selben Zug die maximale Baumtiefe als Parameter betrachtet. Dafür werden Accuracy, Precision und Recall als Evaluationsmetriken herangezogen.

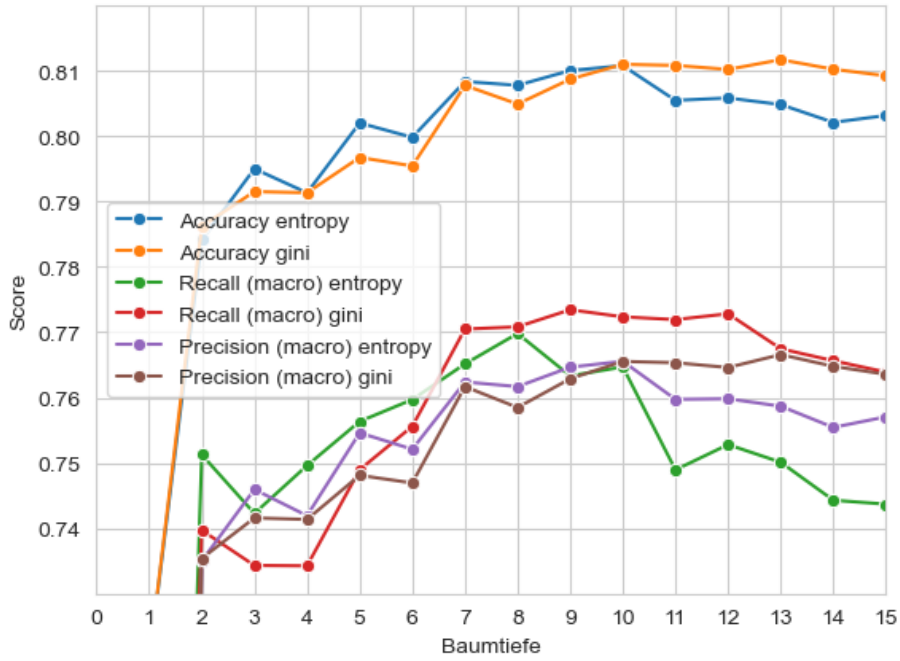

Abbildung 13: Evaluation verschiedener Baumtiefen für beide „criterion“ Parameter des Decision Tree-Modells
Quelle: Eigene Darstellung

Im Diagramm (Abbildung 13) lässt sich direkt erkennen, dass alle aus den Entropy-Modellen stammenden Kurven fast durchgängig besser performen als die Kurven, welche sich aus den Gini-Modellen ergeben. Die maximale Accuracy des Entropy-Modells liegt bei einer Baumtiefe von zehn vor, mit einem Wert von 81,1 %. Die höchste Precision erreicht das Entropy-Modell ebenfalls bei einer Baumtiefe von zehn, mit 76,6 %. Den besten Recall liefert das Modell bei einer Baumtiefe von acht, mit 76,9 %.

Der nächste Parameter der betrachtet wird, ist „min_samples_leaf“. Dieser bestimmt wie viele Stichproben sich mindestens in einem Blatt des Decison Trees befinden müssen. Neben ganzen Zahlen können hier auch Bruchteile der Gesamtmenge der Daten angegeben werden.

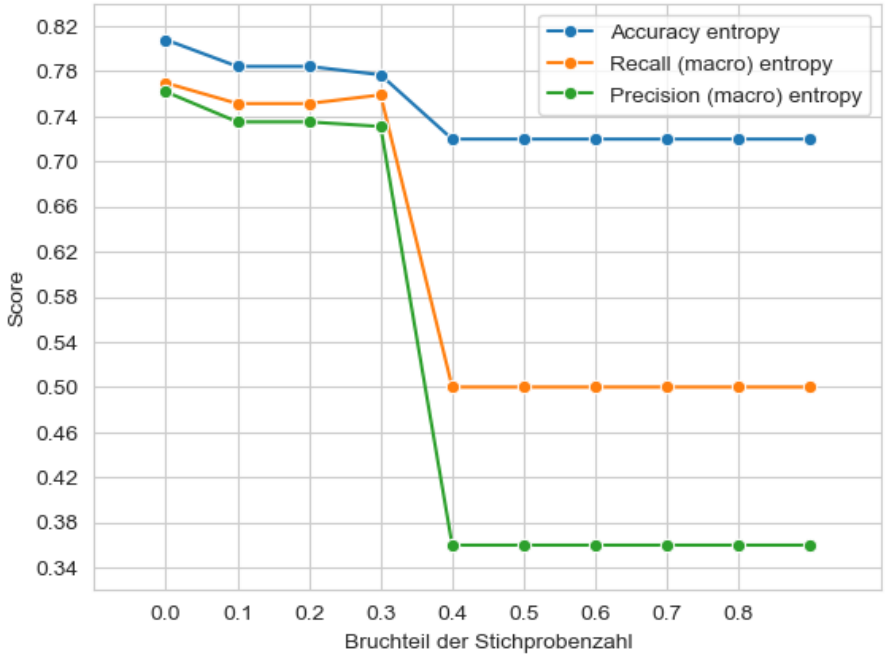

Abbildung 14: Evaluation verschiedener Bruchteile für den Parameter „min_samples_leaf“ des Decision Tree-Modells
Quelle: Eigene Darstellung

Im dazugehörigen Diagramm (Abbildung 14) lässt sich jedoch direkt erkennen, dass sich alle Evaluationsmetriken bei manuellem Einstellen lediglich verschlechtern. Der Standardwert, welcher hier bei einer Minimalanzahl von einer benötigten Stichprobe pro Blatt liegt und hier von dem Bruchteil null repräsentiert wird, liefert das beste Modell.
Ähnlich sieht das Ganze beim dritten und letzten Parameter „min_samples_split“ aus. Auch hier geht es um die Minimalanzahl von Stichproben, allerdings um die Minimalanzahl an Stichproben, die benötigt werden um eine neue Abzweigung zuzulassen.

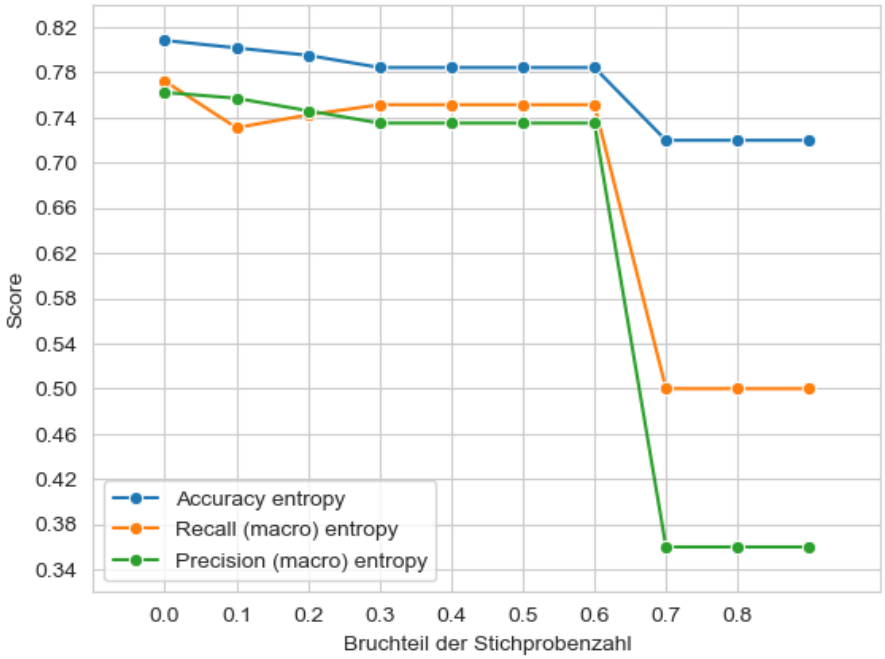

Abbildung 15: Evaluation verschiedener Bruchteile für den Parameter „min_samples_split“ des Decision Tree-Modells
Quelle: Eigene Darstellung

Auch hier zeigt das Diagramm (Abbildung 15), dass sich die Evaluationsmetriken beim Erhöhen der benötigten Stichprobenzahl lediglich verschlechtern. Genauso wie beim letzten Parameter ist der Standardparameter von zwei, welcher hier vom Bruchteil null repräsentiert wird, die beste Wahl und ein manuelles Ändern des Wertes verschlechtert das Modell schlimmstenfalls.
Insgesamt scheint das Hyperparametertuning in diesem Anwendungsfall keinen großen Nutzen zu haben. Einige Parameter schaffen es zwar das Modell zu verbessern, jedoch nur im messbaren Nachkommabereich. Die Frage, ob dieses Feintuning in der Praxis einen Mehrwert bietet, bleibt dabei offen.

3.2.2.5 Finales Modell

Das finale und beste Modell des Decision Trees basiert auf den nachträglich errechneten Features Stromstärkedifferenz, für die Gelenke eins bis fünf, sowie die daraus resultierende Durchschnittsdifferenz aller Gelenke. Zusätzlich werden beim Trainieren die Parameter „criterion=‘entropy‘“ und „max_depth=8“ manuell befüllt, um das theoretisch genauste Modell zu erhalten.

Tabelle 5: Klassifikationsreport des finalen Decision Tree-Modells
Quelle: Eigene Darstellung

	precision	recall	f1-score
false	0.87	0.86	0.86
true	0.65	0.68	0.67
accuracy			0.81
macro avg	0.76	0.77	0.76
weighted avg	0.81	0.81	0.81

Das finale Modell kann sich somit in alle Metriken (Tabelle 5) merklich verbessern. Die Genauigkeit steigt insgesamt von 72 % auf 81 % an. Aber auch alle anderen Evaluationsmetriken steigen an. Die Präzision (makro) von 63 % auf 76 %, der Rückruf (makro) von 58 % auf 77 % und somit auch der F1-Score (makro) von 59 % auf 76 %.

3.2.3 k-Nearest Neighbor

3.2.3.1 Referenzmodell und Trainingsdaten

Auch beim kNN-Modell wird zunächst ein Referenzmodell geschaffen, welches mit allen verfügbaren Features trainiert wird. Wie auch beim Decision Tree scheint sich dieses jedoch nicht zu verbessern, wenn die Anzahl der beiden Klassen ausgeglichen wird. Somit dient auch beim kNN-Modell der erste Datensatz als Trainingsdatensatz. Die restlichen Datensätze, ausgenommen des ersten Datensatzes, dienen als Validierungsdaten um die trainierten Modelle zu evaluieren.

Tabelle 6: Klassifikationsreport des kNN-Referenzmodells mit den Validierungsdaten
Quelle: Eigene Darstellung

	precision	recall	f1-score
false	0.80	0.83	0.82
true	0.52	0.47	0.49
accuracy			0.73
macro avg	0.65	0.65	0.65
weighted avg	0.72	0.73	0.73

Dieses Referenzmodell (Tabelle 6) startet bereits mit besseren Werten als das Referenzmodell des Decision Trees. Die Genauigkeit ist mit 73 % zwar ähnlich hoch, die Präzision, der Recall und der F1-Score liefern jedoch bereits zu Beginn alle ein um einige Prozentpunkte besseres Ergebnis ab.

3.2.3.2 Berechnete Feature-Selection

Im nächsten Schritt werden erneut die berechneten Featurelisten aus Kapitel 3.2.1.3 überprüft. Dabei zeichnet sich bei der Evaluation wieder ein ähnliches Bild ab wie beim Decision Tree.

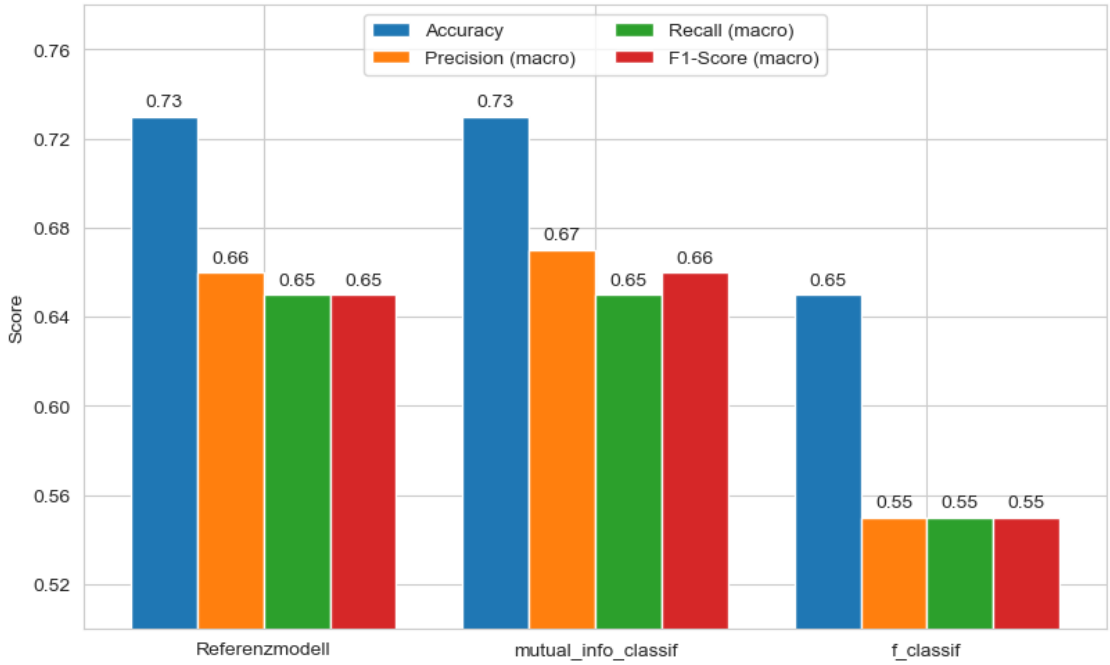

Abbildung 16: Evaluation der berechneten Feature-Selection Modelle im Vergleich zum Referenzmodell (kNN)
Quelle: Eigene Darstellung

Es lässt sich erkennen (Abbildung 16), dass die mit der „mutual_info_classif"-Funktion berechneten Features minimal besser abschneiden als das Referenzmodell, welchem freie Hand gelassen wurde. Dabei kann sich jedoch lediglich die Präzision (makro) der Vorhersagen um einen Protzentpunkt von 66 % auf 67 % verbessern, und damit auch der daraus resultierende F1-Score (makro) von 65 % auf 66 %. Das Modell welches mithilfe der „f_classif"-Funktion berechnet wurde, schneidet deutlich schlechter ab und verliert im F1-Score (makro) ganze 11 % zum Referenzmodell.

3.2.3.3 Manuelle Feature-Selection

Als Hilfsmittel für die manuelle Feature-Selection werden Streudiagramm-Matrizen eingesetzt. Diese erlauben es, bestimmte Features in verschiedenen Paarkonstellationen zu visualisieren. Dort werden verschiedene Kombinationen der Differenzdurchschnitte aus dem Feature-Engineering betrachtet.

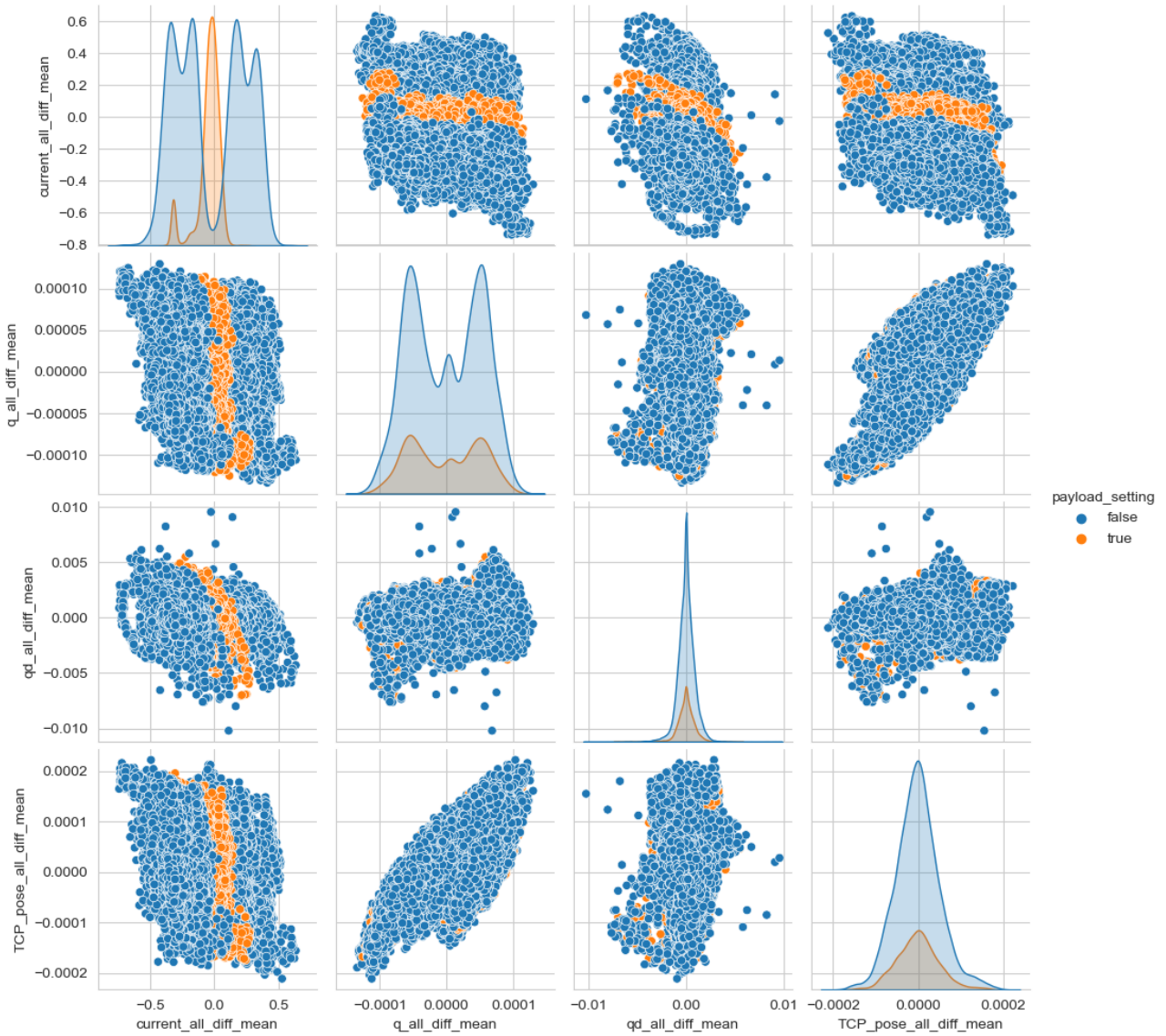

Abbildung 17: Streumatrix vorgefilterter Differenzdurchschnitte, eingeteilt nach Klassezugehörigkeit der Payload-Einstellung
Quelle: Eigene Darstellung

Die erste Matrix (Abbildung 17) wurde bereits vorgefiltert und einige nichts sagende Durchschnittsdifferenzen wurden bereits entfernt. Es lässt sich auch hier wieder klar erkennen, dass alle Streudiagramme, welche die Durchschnittsdifferenz der Stromstärke enthalten, eine relativ klare Grenze zwischen den beiden Klassen besitzen. Dies ist am orangenen Streifen aus Datenpunkten (Klasse „true“) erkennbar, welcher sich durch die blauen Datenpunkte (Klasse „false“) zieht. Ist dieses Feature nicht Teil des Streudiagramms, handelt es sich eher um blaue Punktewolken mit einzelnen orangenen Datenpunkten, ohne visuell erkennbare Grenzen zwischen den Farbbereichen.
Die zweite Streudiagramm-Matrix zeigt vorgefilterte Features in Form der Standardabweichungen der Differenzdurchschnitte (Abbildung 18).

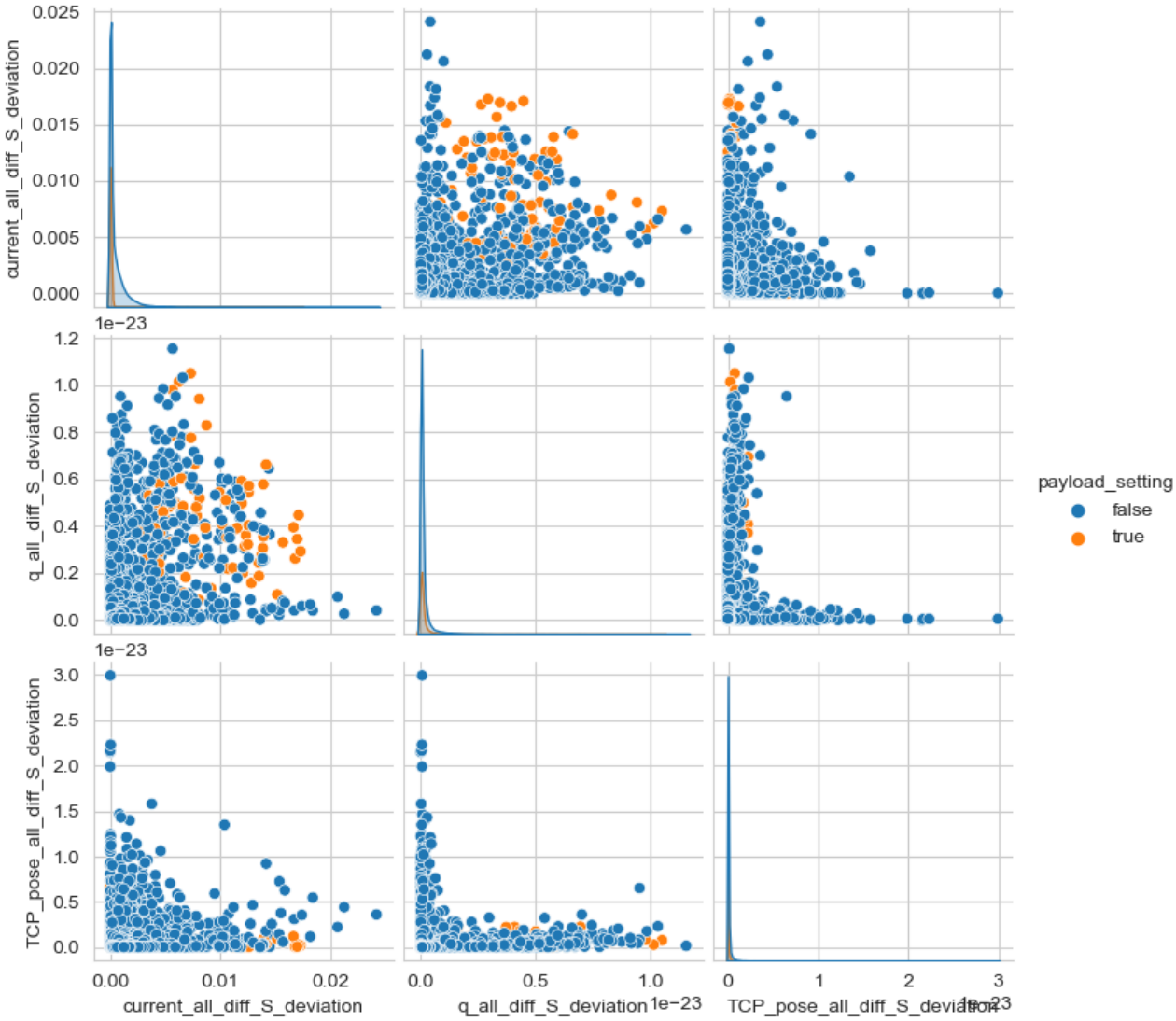

Abbildung 18: Streumatrix vorgefilterter Standardabweichungen der Differenzdurchschnitte, eingeteilt nach Klassezugehörigkeit der Payload-Einstellung
Quelle: Eigene Darstellung

Auch hier lässt sich bei der Kombination, Standardabweichung der Stromstärkedurchschnittsdifferenz (current_all_diff_S_deviation) und Standardabweichung der durchschnittlichen Gelenkpositionsdifferenz (q_all_diff_S_deviation) eine visuelle Grenze erkennen. Diese Grenze ist aber in der Deutlichkeit nicht mit den Grenzen aus der ersten Streudiagramm-Matrix zu vergleichen. Die Abgrenzung der orangenen Datenpunkte verläuft deutlich unklarer und ist mit einer größeren Anzahl an blauen Datenpunkten „verunreinigt“.

Aus den beiden Streudiagramm-Matrizen und den Vorerfahrungen aus dem Decision Tree-Modell ergeben sich die vier nachfolgenden Feature-Sets, welche getestet werden sollen:

- Set 1 bestehend aus den Stromstärkedifferenzen aller Gelenke, sowie der daraus resultierenden Durchschnittsdifferenz. Es handelt sich dabei um die finalen Features des Decision Tree-Modells.
- Set 2 bestehend aus der Stromstärkedurchschnittsdifferenz und der durchschnittlichen Gelenkpositionsdifferenz. Hier lässt sich in der ersten Streudiagramm-Matrix eine der besten visuellen Grenzen zwischen den Klassen ziehen.
- Set 3 bestehend aus der durchschnittlichen Gelenkpositionsdifferenz und der durchschnittlichen TCP-Positionsdifferenz. Dieses dient als Referenzset ohne die durchschnittliche Stromstärkedifferenz.
- Set 4 bestehend aus denselben Features wie Set 1 plus die Features, Durchschnittsdifferenzen der Gelenkposition, der TCP-Position und der TCP-Geschwindigkeit.

Alle vier Feature-Sets werden genutzt, um ein kNN-Modell zu trainieren und dieses mit den bekannten Evaluationsmetriken zu bewerten und zu vergleichen.

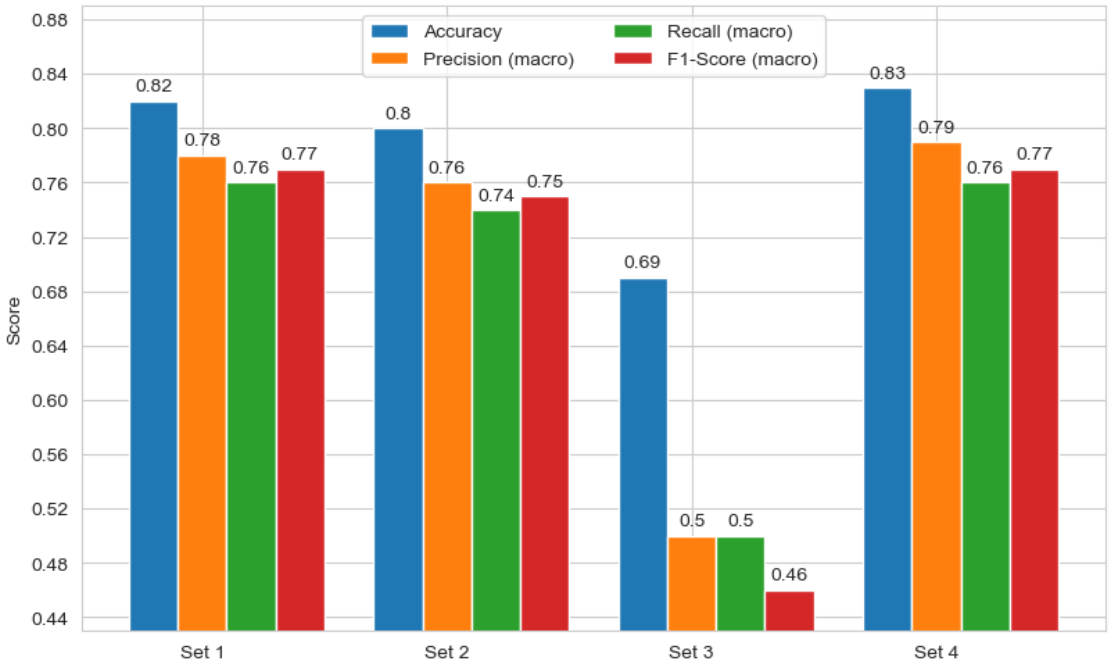

Abbildung 19: Evaluation verschiedener kNN-Modelle auf Basis verschiedener Feature-Sets
Quelle: Eigene Darstellung

Im Balkendiagramm (Abbildung 19) zeigt sich an Set 3 wieder, wie wichtig die durchschnittliche Stromstärkedifferenz für die Modelle ist. Set 3 ist das einzige Feature-Set ohne die durchschnittliche Stromstärkedifferenz und liegt in allen Evaluationsmetriken deutlich hinter den anderen Feature-Sets.

Set 2 performt recht gut, obwohl es lediglich aus zwei Features insgesamt besteht. Besser performen lediglich Set 1 welches sich bereits beim Decision Tree-Modell bewährt hat, und Set 4.
Set 4, besteht dabei aus denselben Features wie Set 1, sowie allen zusätzlichen Durchschnittsdifferenzen. Es liefert die besten Evaluationsmetriken mit einer Genauigkeit von 83 %, einer Präzision (makro) in den Vorhersagen von 79 %, einem Rückruf (makro) von 76 %, und einem F1-Score (makro) von 77 %.
Da es sich bei Set 4 um das Feature-Set mit den besten Werten handelt, bildet es die Ausgangslage für das Hyperparametertuning.

3.2.3.4 Hyperparametertuning

Startpunkt beim Hyperparametertuning des kNN-Modells ist der Parameter „n_neighbors“. Dieser Parameter bestimmt wie viele Nachbarpunkte für die Klassifizierung eines Punktes betrachtet werden sollen.

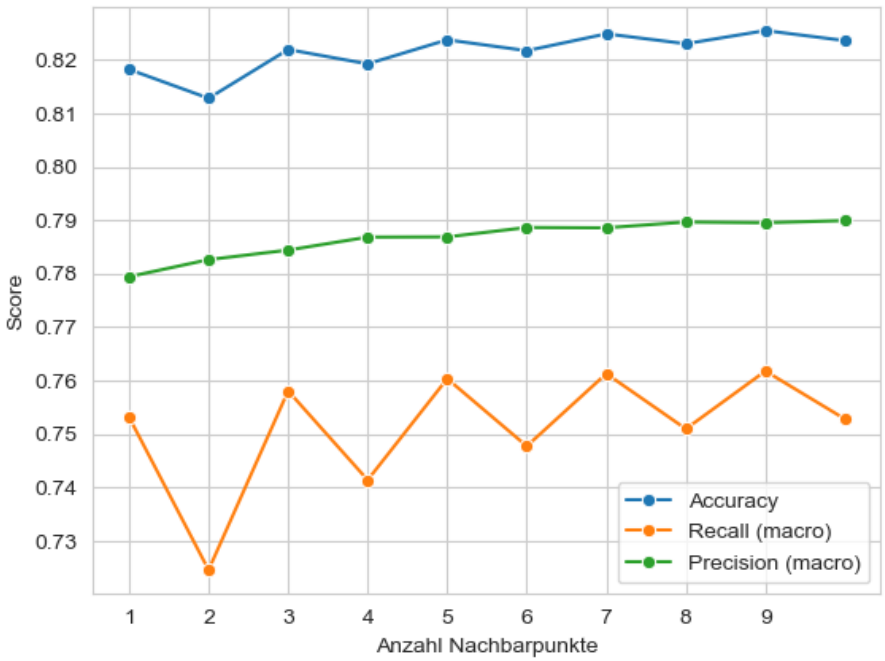

Abbildung 20: Evaluation verschiedener Anzahlen von Nachbarpunkten des kNN-Modells
Quelle: Eigene Darstellung

Gut zu erkennen ist dabei (Abbildung 20), wie für eine ungerade Zahl von Nachbarn die Metriken Precision und Recall stets einen merklich höheren Wert erreichen als die nachfolgende gerade Zahl. Auch ist zu erkennen, dass sich alle Metriken über den Erhebungsrahmen mit steigender Anzahl an Nachbarn auch stetig leicht zu verbessern scheinen. Da jedoch die Kurven bereits nach den ersten

drei bis fünf Nachbarn abflachen und es sich bei kNN-Modellen um sehr rechenintensive und zeitaufwendige Modelle handelt, wird für den fortlaufenden Verlauf eine Anzahl der Nachbarn von fünf gewählt.

Der zweite Parameter „metric“ ändert die Art der Berechnung, welche für den Abstand zu den Nachbarpunkten gewählt wird. Dabei werden die drei gängigsten kNN-Berechnungsarten „euclidian“, „manhattan“ und „minkowski“ getestet.

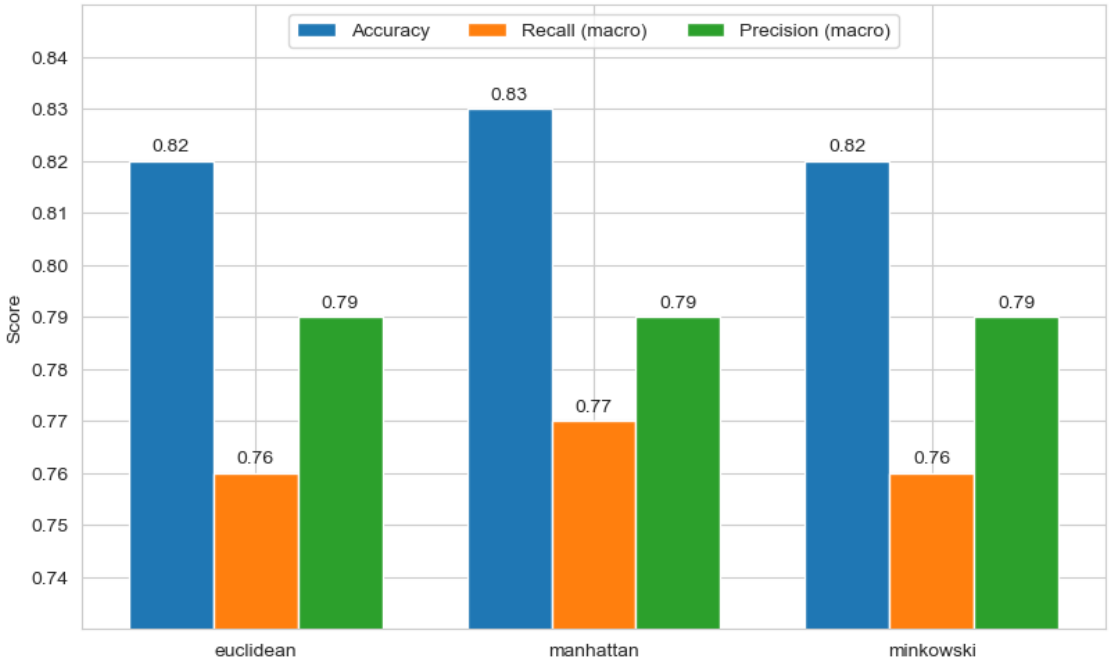

Abbildung 21: Evaluation verschiedener Berechnungsarten für den Parameter „metric“ des kNN-Modells
Quelle: Eigene Darstellung

Erkennbar ist (Abbildung 21), dass alle drei Berechnungsarten sehr ähnlich abschneiden. Dennoch erreicht die Berechnungsart „manhattan“ im Recall ein, um einen Prozentpunkt, besseres Ergebnis als „euclidean“ und „minkowski“. Sie wird somit als finale Berechnungsart ausgewählt.

Der dritte zu betrachtende Parameter bei dem kNN-Modell ist „weight“, die Gewichtung der Nachbarpunkte. Diese Gewichtung ist standardmäßig „uniform“. Dies bedeutet sie ist für jeden der betrachteten Nachbarn gleich.

Das könnte auch die Einbrüche der Metriken bei einer geraden Anzahl von Nachbarpunkten beim Parameter „n_neighbors“ (Abbildung 20) erklären.

Alternativ kann dieser Parameter den Übergabewert „distance“ bekommen. Bei diesem haben die Nachbarpunkte nicht alle denselben Einfluss auf die Klassifizierung, sondern der Einfluss der Nachbarpunkte wird nach deren Abstand gewichtet.

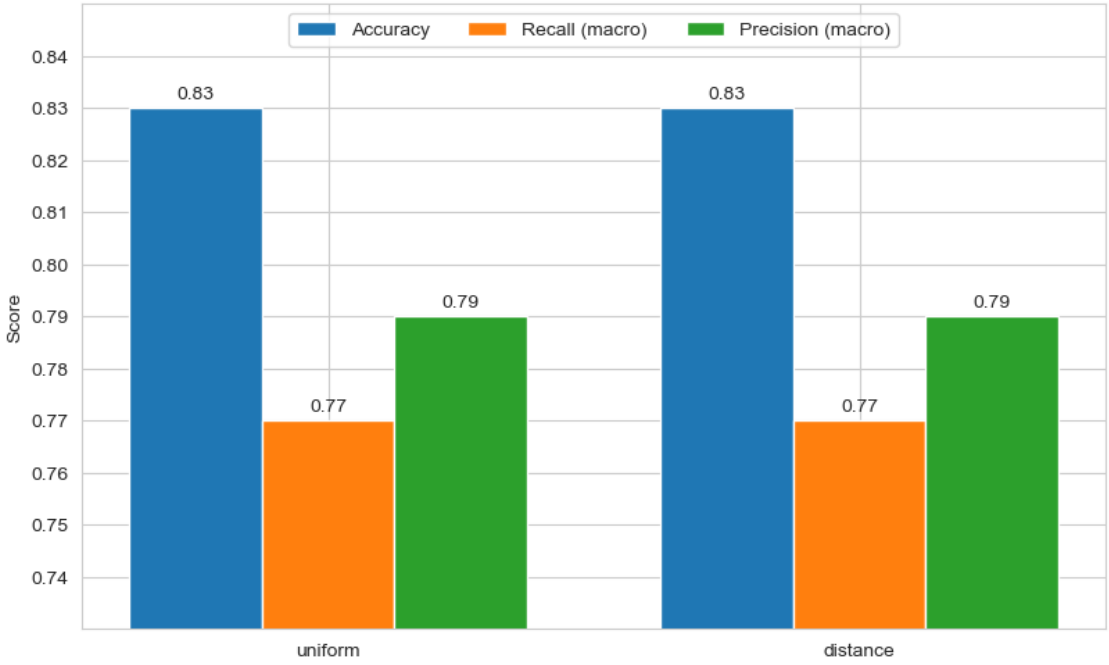

Abbildung 22: Evaluation verschiedener Gewichtungsarten für den Parameter „weight“ des kNN-Modells
Quelle: Eigene Darstellung

Im Schaubild (Abbildung 22) erkennt man, dass die Gewichtung keinen Einfluss auf die Evaluationsmetriken hat. Das könnte auch damit zusammenhängen, dass sich zuvor bereits für eine ungerade Anzahl von zu betrachtenden Nachbarpunkten entschieden wurde und somit die Gewichtung eine untergeordnete Rolle spielt. Grund dafür ist, dass bei einer ungeraden Anzahl an Nachbarn immer eine Klassifizierung durch die Mehrheit der Nachbarpunkte erfolgen kann, unabhängig von der Gewichtung.

3.2.3.5 Finales Modell

Das finale und beste kNN-Modell basiert ebenfalls, wie das Decision Tree-Modell, auf den nachträglich errechneten Features Stromstärkedifferenz für die Gelenke eins bis fünf, sowie auf der daraus resultierende Durchschnittsdifferenz über alle Gelenke. Zusätzlich kommen hier jedoch noch die Features Durchschnittsdifferenz der Gelenkpositionen (q_all_diff_mean), Durchschnittsdifferenz der Zielgeschwindigkeiten (q_all_diff_mean), Durchschnittsdifferenz der TCP-Positionen (TCP_pose_all_diff_mean), sowie Durchschnittsdifferenz der TCP-Geschwindigkeiten (TCP_speed_all_diff_mean), hinzu. Beim Trainieren des Modells werden außerdem die Parameter „n_neighbors=5“ und „metric=manhattan“ übergeben.

Tabelle 7: Klassifikationsreport des finalen kNN-Modells
Quelle: Eigene Darstellung

	precision	recall	f1-score
false	0.87	0.90	0.88
true	0.72	0.64	0.68
accuracy			0.83
macro avg	0.79	0.77	0.78
weighted avg	0.83	0.83	0.83

Die finalen Werte (Tabelle 7) verbessern sich in allen Bereichen gegenüber dem Referenzmodell. Die gesamte Genauigkeit verbessert sich von 73 % auf 83 %. Besonders beachtlich sind die Steigerungen der Präzision, des Rückrufs und des daraus resultierenden F1-Scores für die Klasse „true“. Hier kann sich das Modell von 49 % im F1-Score des Referenzmodells auf ganze 68 % im finalen Modell verbessern.
Neben der Verbesserung zum Referenzmodell kann sich das kNN-Modell auch um ein bis zwei Prozentpunkte pro Evaluationsmetrik im Vergleich zum finalen Decison Tree-Modell verbessern.

3.2.4 Logistische Regression

3.2.4.1 Referenzmodell und Trainingsdaten

Wie bei den beiden vorangegangenen Machine Learning-Modellen wird auch beim Modell auf Grundlage der logistischen Regression zunächst ein Referenzmodel erstellt, um nachfolgende Modelle vergleichen zu können. Anders als bei den zwei vorherigen Modellen ist der erste Datensatz als Trainingsdatensatz nicht geeignet. Sowohl bei der Überprüfung mit einem Teil dieser Daten als Testdaten, als auch mit dem Validierungsdatensatz ist es einem damit trainierten Modell nicht möglich auch nur einen Datenpunkt der Klasse „true“ zu erkennen. Es handelt sich somit um ein unbrauchbares Modell.
Das Referenzmodell wird also mit Trainingsdaten trainiert, die grundlegend aus dem ersten Datensatz bestehen. Zusätzlich wird dieser Datensatz mit Datenpunkten der Klasse „true“ aufgefüllt, um ein annäherndes Verhältnis von eins zu eins innerhalb der Trainingsdaten zwischen den beiden Klassen „true“ und „false“ zu schaffen. Mit diesen Trainingsdaten gelingt es, ein funktionales Modell zu trainieren. Dieser Trainingsdatensatz wird auch für alle zukünftigen logistischen Regressionsmodelle genutzt.

Tabelle 8: Klassifikationsreport des logistischen Regressions-Referenzmodells mit den Validierungsdaten
Quelle: Eigene Darstellung

	precision	recall	f1-score
false	0.75	0.45	0.56
true	0.31	0.62	0.41
accuracy			0.50
macro avg	0.53	0.53	0.49
weighted avg	0.63	0.50	0.52

Die Ausgangswerte (Tabelle 8) des logistischen Regressionsmodells sind deutlich niedriger als die der beiden vorangegangenen Modelle. Die Genauigkeit des Referenzmodells beträgt gerade einmal 50 % und es liegt damit über 20 % hinter den anderen beiden Referenzmodellen. Auch die makro Werte für die Präzision und den Rückruf liegen mit jeweils 53 % nur knapp über der 50 % Marke und somit auch hier deutlich an dritter Stelle der Referenzmodelle.

3.2.4.2 Berechnete Feature-Selection

Anschließend geht es an die Wahl der besten Features zum Trainieren des Modells. Zwischen „mutual_info_classif“ und „f_classif“ zeichnet sich ein ähnliches Bild wie die Male zuvor ab.

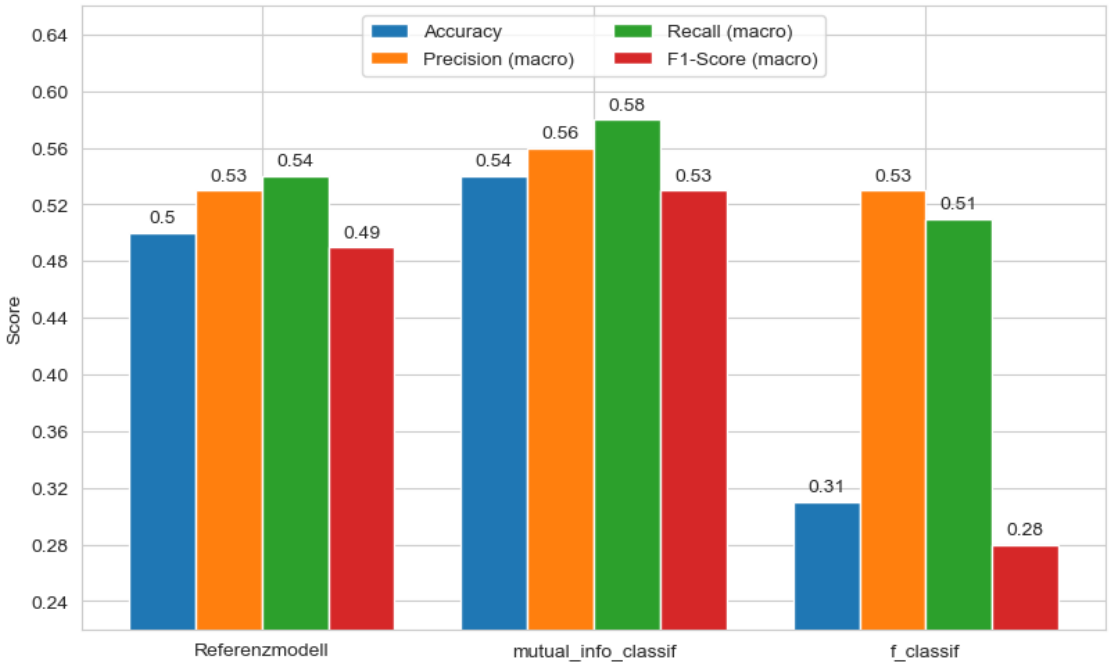

Abbildung 23: Evaluation der berechneten Feature-Selection Modelle im Vergleich zum Referenzmodell (Logistische Regression)
Quelle: Eigene Darstellung

Der Zuwachs (Abbildung 23) vom Referenzmodell zu dem Modell der „mutual_info_classif“ Features ist der höchste absolute Zuwachs, in Accuracy und F1-Score (makro), über alle drei Machine Learning Arten hinweg. Allerdings sind alle Evaluationsmetriken auch mit den berechneten besten Features immer noch unter 60 %. Neben dem höchsten absoluten Zuwachs von Referenzmodell zum Modell von „mutual_info_classif“, erfährt jedoch das Modell auf Basis der „f_classif“ Featurewahl auch die höchsten absoluten Einbußen von allen drei Machine Learning Arten.

3.2.4.3 Manuelle Feature-Selection

Auf Grundlage der beiden vorangegangen Machine Learning Modelle und deren Feature-Sets werden ähnliche Feature-Sets für die logistische Regression getestet. Die nachfolgenden Feature-Sets werden genutzt, um ein logistisches Regressionsmodell zu trainieren und dieses mit den Validierungsdaten zu testen:

- Set 1 bestehend aus dem durch „mutual_info_classif“ berechneten Feature-Sets als Referenzwert.
- Set 2 bestehend aus den Stromstärkedifferenzen aller Gelenke, sowie der daraus resultierenden Durchschnittsdifferenz. Es handelt sich dabei um die finalen Features des Decision Tree-Modells.
- Set 3 bestehend aus den Stromstärkedifferenzen aller Gelenke, sowie der daraus resultierenden Durchschnittsdifferenz und der Durchschnittsdifferenzen der Gelenkposition, der TCP-Position und der TCP-Geschwindigkeit. Es handelt sich dabei um die finalen Features des kNN-Modells.
- Set 4 bestehend aus einer Teilmenge des durch „mutual_info_classif“ berechneten Feature-Sets.

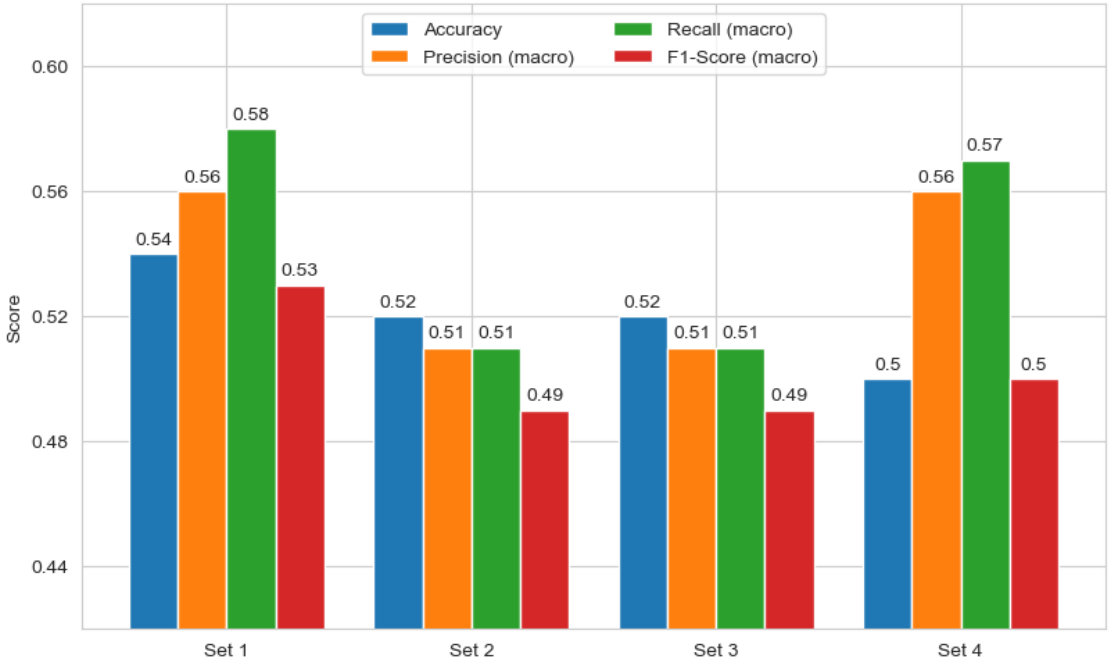

Abbildung 24: Evaluation verschiedener logistischer Regressionsmodelle auf Basis verschiedener Feature-Sets
Quelle: Eigene Darstellung

Im Evaluationsdiagramm (Abbildung 24) zeigt sich, dass keines der vorherigen Sets es schafft Set 1, mit den von „mutual_info_classif“ berechneten Features, zu übertreffen.
Auch Set 2 und Set 3, welche die besten Features beim Decision Tree und beim kNN-Modell waren, schaffen dies nicht. Generell scheint das Feature der Stromstärkedurchschnittsdifferenz hier bei weitem nicht den gleichen Informationsgewinn für die Klassifizierung zu schaffen wie bei den beiden vorangegangenen Modellen. Es konnte keine Feature-Set gefunden werden, welches bessere Werte als Set 1, mit den berechneten Features, liefert. Set 1 dient somit als Ausgangslage für das Hyperparametertuning.

3.2.4.4 Hyperparametertuning

Der erste Parameter „solver“ besitzt sechs mögliche Werte, welche darüber entscheiden, was für einen Algorithmus das Modell für die interne Berechnung nutzt.

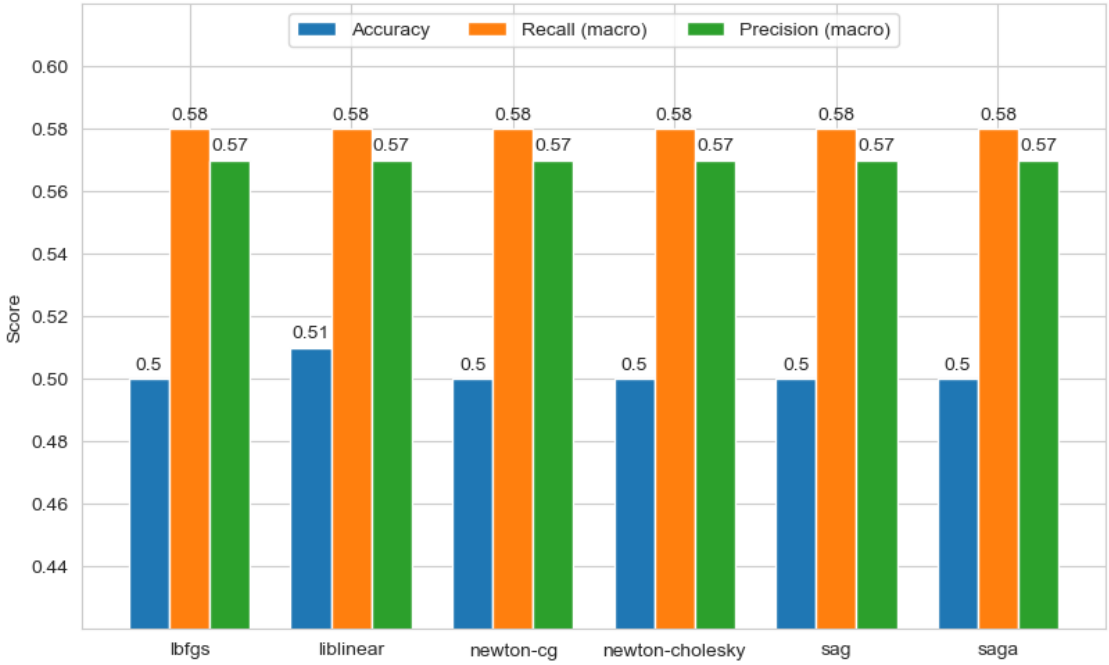

Abbildung 25: Evaluation verschiedener Werte für den Parameter „solver“ des logistischen Regressionsmodells
Quelle: Eigene Darstellung

Alle sechs Parameteroptionen (Abbildung 25) unterscheiden sich dabei in den Ergebnissen kaum. Lediglich der standardmäßige Parameter „lbfgs“ und der Parameter „liblinear“ führen zu einem Modell, welches in der Genauigkeit mit 51 % einen Prozentpunkt vor den anderen Modellen liegt. Die Ergebnisse für die Precision, mit 57 %, und den Recall, mit 58 %, sind dabei mit allen Parametern über alle Modelle hinweg gleich. Der Parameter wird deshalb nicht verändert, beziehungsweise beim standardmäßigen Parameter „lbfgs“ belassen.

Der zweite Parameter „penalty“ bestimmt die Art und Weise, wie das logistische Regressionsmodell beim Trainieren bestraft wird, wenn es zu viele Features beim Trainieren verwendet. So versucht das Modell selbst eine Überanpassung auf die Trainingsdaten zu verhindern. Dieser Parameter besitzt zwei verschiedene Arten von Strafen, sowie einen Wert bei welchem das Modell für eine große Anzahl an Features nicht bestraft wird. Im direkten Zusammenhang zu diesem Parameter steht der

dritte Parameter „C“, welcher über die Härte der Strafe entscheidet. Kleinere Werte bedeuten dabei ein höheres Strafmaß als größere.

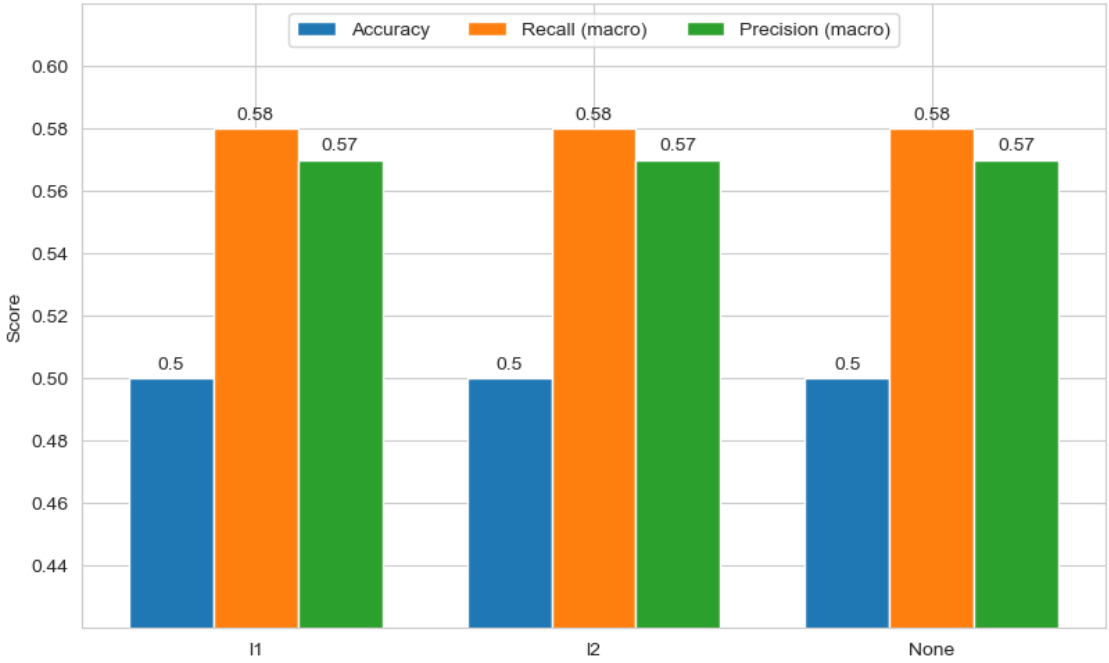

Abbildung 26: Evaluation verschiedener Werte für den Parameter „penalty“ des logistischen Regressionsmodells
Quelle: Eigene Darstellung

Bei den Modellen mit den unterschiedlichen Werten für den zweiten Parameter „penalty“ (Abbildung 26) gibt es zwischen den Modellen keinen Unterschied. Sie sind über alle drei Werte, Genauigkeit, Precision und Recall identisch. Deshalb wird hier auf den Standardparameter für den weitere Verlauf zurückgegriffen.

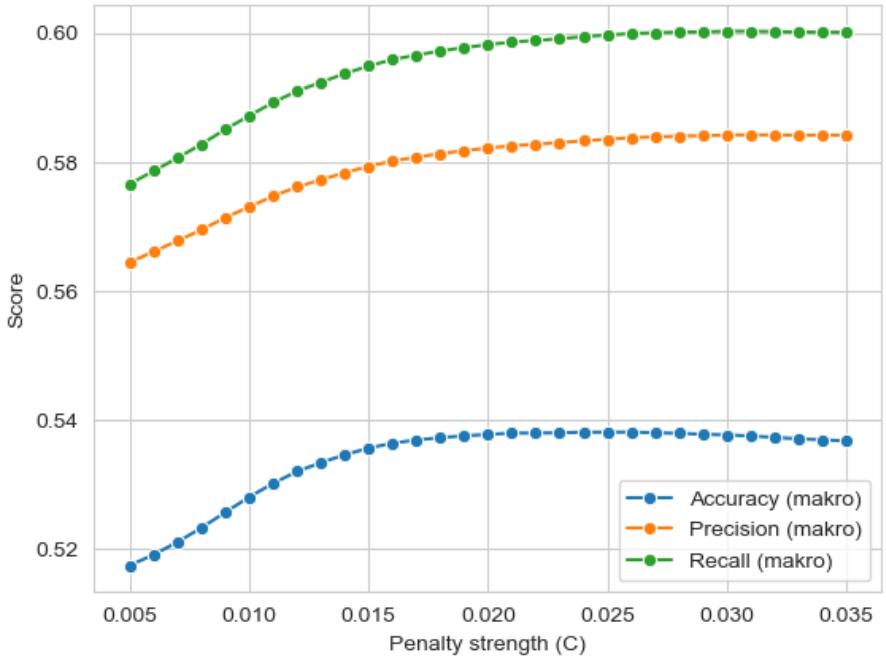

Abbildung 27: Evaluation verschiedener Strafmaße für den Parameter „C“ des logistischen Regressionsmodells
Quelle: Eigene Darstellung

Der dritte Parameter „C“ legt das Strafmaß fest, mit welchem der Algorithmus bestraft wird, wenn er eine hohe Anzahl von Features in die Berechnung mit einfließen lässt. Die Kurven zeigen (Abbildung 27), dass bei zu engem Strafmaß, also kleinen Zahlen, die Genauigkeit, die Precision und der Recall abnehmen. Gleichzeitig nehmen Genauigkeit, Precision und Recall aber auch ab, wenn ein zu geringes Strafmaß vorliegt, also der Wert für C zu groß wird. Der beste Wert der Genauigkeit liegt mit 53,8 % bei einem C-Wert von 0,026 vor. Die besten Werte für die Precision und den Recall liegen bei einem C-Wert von 0,031, mit 58,4 % für die Precision und 60,0 % für den Recall vor. Für das finale Modell wird der Mittelwert von 0,028 für den C-Wert, genommen.

3.2.4.5 Finales Modell

Das finale logistische Regressionsmodell basiert als einziges der drei Machine Learning-Modellen auf dem berechneten Feature-Set aus Kapitel 3.2.1.3 mit der Funktion „mutual_info_classif“. Zusätzlich besitzt es ein spezifiziertes Strafmaß in Form des Parameters „C=0,028“.

Tabelle 9: Klassifikationsreport des finalen logistischen Regressionsmodells
Quelle: Eigene Darstellung

	precision	recall	f1-score
false	0.82	0.46	0.59
true	0.31	0.62	0.47
accuracy			0.54
macro avg	0.58	0.60	0.53
weighted avg	0.69	0.54	0.56

Das finale logistische Regressionsmodell (Tabelle 9) kann sich im Vergleich zum Referenzmodell kaum verbessern. Bei der Precision ist lediglich eine Steigerung um vier Prozentpunkte von 50 % auf 54 % möglich. Aber auch in den Bereichen Precision und Recall verbessert sich das finale Modell nur minimal. Der aus diesen Werten resultierende F1-Score (makro) steigt lediglich von 49 % beim Referenzmodell auf 53 % beim finalen Modell.
Damit ist das logistische Regressionsmodell mit Abstand das unzuverlässigste Modell der drei betrachteten Modellarten, wenn es darum geht die Payload-Einstellung vorherzusagen.

3.3 Übersicht Einflussfaktoren der Modelle

Während der Untersuchungen haben sich eine Vielzahl von Faktoren als mehr oder weniger einflussreich auf die Genauigkeit der Machine Learning-Modelle herausgestellt (Tabelle 10). Dabei stellt sich heraus, dass vor allem die Wahl des Machine Learning-Modells den größten Einfluss auf die Evaluationsmetriken hat. So scheint die logistische Regression ein, im Vergleich zum Decision Tree oder dem kNN-Modell, weniger brauchbares Modell zu sein, wenn es darum geht die Payload-Einstellung eines Industrieroboters vorherzusagen.
Innerhalb eines einzelnen Machine Learning-Modells lässt sich beobachten, dass primär die Wahl der richtigen Features Einfluss auf die Steigerung der Modellqualität hat. So schafft die berechnete Feature-Selection eine durchschnittliche Steigerung von 2,3 % in der Accuracy gegenüber der Referenzmodelle und 3 % im F1-Score (makro). Noch besser performen hingegen die Modelle, bei welchen die Features manuell selektiert werden. Hier werden die Features auf Grundlage der visuellen Vorbetrachtung, logischer Schlussfolgerungen und Vorerfahrungen ausgewählt. Durch die manuelle Feature-Selection lassen sich die Modelle im Vergleich zur berechneten Feature-Selection noch einmal um durchschnittlich 5,3 % in der Accuracy und um 8 % im F1-Score (makro) verbessern. Den messbar geringsten Einfluss hat das Hyperparametertuning, hier kann sich kein Modell in der

Accuarcy erneut steigern. Auch beim F1-Score (makro) können sich zwei von drei Modellen nicht steigern. Lediglich beim kNN-Modell ist eine Steigerung um einen Prozentpunkt möglich.

Tabelle 10: Accuracy und F1-Score (makro) aller drei Modellarten über alle Bearbeitungsschritte hinweg, inklusive Zuwachs zum vorangegangen Bearbeitungsschritt
Quelle: Eigene Darstellung

Machine Learning Modellart	Referenzmodell		Berechnete Feature-Selection		Manuelle Feature-Selection		Parametertuning	
	Accuracy	F1-Score (makro)	Accuracy	F1-Score (makro)	Accuracy	F1-Score (makro)	Accuracy	F1-Score (makro)
Decision Tree	72 %	59 %	75 % (+3 %)	63 % (+4 %)	81 % (+6%)	76 % (+13 %)	81 % (+0 %)	76 % (+0 %)
kNN	73 %	65 %	73 % (+0 %)	66 % (+1 %)	83 % (+10 %)	77 % (+11 %)	83 % (+0 %)	78 % (+1 %)
Logistische Regression	50 %	49 %	54 % (+4 %)	53 % (+4 %)	54 % (+0 %)	53 % (+0 %)	54 % (+0 %)	53 % (+0 %)
Durchschnittlicher Zuwachs			+2,3 %	+3 %	+5,3 %	+8 %	+0 %	+0,3 %

4 Fazit und Ausblick

4.1 Fazit

Ziel dieser Arbeit war es herauszufinden, wie mithilfe von Machine Learning ermittelt werden kann, ob die aktuell eingestellte Payload eines Industrieroboter-Greifarms richtig eingestellt ist und welche Faktoren die Genauigkeit eines solchen Vorhersagemodells beeinflussen.

Es zeigt sich eine solche Vorhersage kann mit den vom Roboter aufgezeichneten Daten getroffen werden, ohne das Einsetzen von externen Sensoren für die Aufzeichnung zusätzlicher Daten. Wichtig ist dafür, dass Teile der aufgezeichneten Werte in einem gewissen Zusammenhang mit dem Zielmerkmal stehen. Außerdem ist es notwendig sich mit den erhobenen Daten zu beschäftigen, um logische Zusammenhänge, die für den Trainingsprozess erforderlich sind, zu erkennen. Je nach Daten kann es nötig oder hilfreich sein, bestimmte neue Features aus den vorhandenen Features abzuleiten, um diese so auch für Machine Learning Modelle quantifizierbar zu machen. Im Falle des Industrieroboters UR10 waren das vor allem Differenzen zwischen Erwartungswerten des Roboters und tatsächlich aufgezeichneten Werten des Roboters, welche den Machine Learning-Modellen einen Informationszuwachs eingebracht haben.

Eine Vielzahl von Faktoren haben auf die Genauigkeit eines solchen Machine Learning-Modells Einfluss. Einer der wichtigsten Faktoren ist dabei eine repräsentative Datenerhebung. Der Einfluss zeigt sich dabei jedoch erst im Realbetrieb eines Modells und ist deshalb im Rahmen dieser Arbeit nicht quantifizierbar. Unter Laborbedingungen kann es jedoch vorkommen, dass ungewollte Zusammenhänge zwischen Datenmerkmalen und Zielmerkmal geschaffen werden, wie sich in dieser Arbeit an den Temperaturmerkmalen offenbart. Zu den quantifizierbaren Einflussfaktoren, die sich im Lauf der Arbeit gezeigt haben, gehören die Wahl des Machine Learning-Modells, die Wahl der Trainingsdaten für das Modell, die Auswahl der Features und die Parameterwahl des jeweiligen Machine Learning Modells.

Der größte Genauigkeitszuwachs kann hier mit der Wahl des richtigen Machine-Learning Modells erreicht werden. Zwischen dem in diesem Fall ungenauesten (Logistische Regression) und dem genauesten (k-Nearest Neighbor) Ausgangsmodell liegt eine Spanne von 23 Prozentpunkten in der Accuracy und 16 Prozentpunkten im Makrowert F1-Score. Die Wahl von passenden Features schafft eine durchschnittliche Verbesserung über alle drei Machine Learning-Arten hinweg von 5,3 Prozentpunkten in der Accuracy und 8 Prozentpunkten im Makrowert des F1-Scores. Die Hyperparameteranpassung der Modelle schafft es nicht in der Accuracy eine Verbesserung zu erzielen und erreicht im Makrowert des F1-Scores lediglich eine durchschnittliche Verbesserung von

0,3 Prozentpunkten. Zu beachten ist dabei immer, dass sich alle Einflussfaktoren abhängig von dem gewählten Machine Learning-Modell unterscheiden und je nach Modellart unterschiedlichen Genauigkeitszuwachs einbringen.

4.2 Ausblick

Aufgrund der Rahmenbedingungen dieser Arbeit beziehen sich alle Ergebnisse auf Laborbedingungen. Ob die Faktoren und ihr Einfluss auf die Genauigkeit unter Realbedingungen dieselben wären, kann in dieser Arbeit nicht geklärt werden. Ein mögliches Folgeexperiment wäre die Modelle unter Realbedingungen zu testen.

Für die Übertragbarkeit und Anwendung in der Praxis, bei der idealerweise präventive Maßnahmen einen möglichen Produktionsstopp verhindern, reicht die nachträgliche Analyse von Log-Dateien nicht aus. Die Daten des Roboters müssten in Echtzeit über eine Schnittstelle an ein mögliches Machine Learning-Modell geliefert werden, um so Maschinenstopps aufgrund falsch eingestellter Nutzlasten verhindern zu können.

Offen bleibt auch, ob die vom Roboter aufgezeichneten Daten die beste Wahl für die Vorhersage über die Nutzlast-Einstellung sind, oder ob Aufzeichnungen durch externe Sensorik bessere Ergebnisse erzielen würde. Denkbare Sensorik für weitergehende Untersuchungen wären Vibrationssensoren oder Ultraschallsensoren. Interessant könnten auch die Ergebnisse bei einer Kombination aus Vibrationssensorik, Ultraschallsensorik und den Roboterintern aufgezeichneten Daten, sein.

Quellenverzeichnis

Adityarajora (2019): Complete-KNN-visualization. URL: https://github.com/Adityarajora/Complete-KNN-visualization (27.06.2023).

Alpaydin, Ethem (2021): Maschinelles Lernen. Berlin: De Gruyter.

Brecher, Christian/Klein, Wieland Hermann/Lindner, Florian (2009): Condition Monitoring von Werkzeugmaschinen. In: Lernen & lehren: Elektrotechnik - Informatik, Metalltechnik, 95. Jg., S. 117-122.

Chandola, Varun /Banerjee, Arindam/Kumar, Vipin (2009): Anomaly detection: A survey. In: ACM Comput. Surv., 41. Jg. (3).

Cunningham, Pádraig/Cord, Matthieu/Delany, Sarah Jane (2008): Supervised Learning. In: Cord, Matthieu/Cunningham, Pádraig (Hrsg.): Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval. Berlin, Heidelberg: Springer, S. 21-49.

Deepthi A R (2019): KNN visualization in just 13 lines of code. URL: https://towardsdatascience.com/knn-visualization-in-just-13-lines-of-code-32820d72c6b6 (27.06.2023).

Delua, Julianna (2021): Supervised vs. Unsupervised Learning: What's the Difference? URL: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning (14.06.2023).

Eisner, Caroline (2022): What's the Difference between Condition-Based and Predictive Maintenance? URL: https://www.getmaintainx.com/learning-center/condition-based-vs-predictive-maintenance/ (20.07.2023).

Fisher, Ronald A. (1988). Iris. UCI Machine Learning Repository.

Fraunhofer-Institut für Keramische Technologien und Systeme (o. D.): Zustandsüberwachung von Prozessen, Anlagen und Komponenten. URL: https://www.ikts.fraunhofer.de/content/dam/ikts/downloads/profile/IKTS_Industrieloesungen_Zustandsueberwachung.pdf (17.07.2023).

Gamal, Mohamed et al. (2021): Anomalies Detection in Smart Manufacturing Using Machine Learning and Deep Learning Algorithms.

Gołda, Grzegorz/Kampa, Adrian/Paprocka, Iwona (2018): Analysis of human operators and industrial robots performance and reliability. In: Management and Production Engineering Review, 9. Jg., S. 24-33.

Grunert, Philipp (2021): Machine Learning und Neuronale Netze: Der verständliche Einstieg mit Python. Landshut: BMU Media.

Grus, Joel (2015): Data Science from Scratch. Sebastopol: O'Reilly Media.

Hirschle, Jochen (2020): Machine Learning für Zeitreihen. Carl Hanser.

Holbrook, Ryan/Cook, Alexis (o. D.): Mutual Information. URL: https://www.kaggle.com/code/ryanholbrook/mutual-information/tutorial (27.07.2023).

IBM (o. D.-a): F value. URL: https://www.ibm.com/docs/en/cognos-analytics/11.1.0?topic=terms-f-value (27.07.2023).

IBM (o. D.-b): What is machine learning? URL: https://www.ibm.com/topics/machine-learning (13.06.2023).

IFR (2022): World Robotics 2022.

Irfan, Muhammad (Hrsg.) (2019): Advanced Condition Monitoring and Fault Diagnosis of Electric Machines. Hershey: IGI Global.

Joshi, Ameet V. (2022): Machine Learning and Artifical Intelligence. Springer Cham.

Kletti, Jürgen/Rieger, Jürgen (2022): Die neuen Anforderungen der Smart Factory. In: Kletti, Jürgen/Rieger, Jürgen (Hrsg.): Die perfekte Produktion: Manufacturing Excellence in der Smart Factory. Wiesbaden: Springer Fachmedien Wiesbaden, S. 11-20.

Köhler, Peter/Six, Björn/Michels, Jan Stefan (2015): Industrie 4.0: Ein Überblick. In: Köhler-Schute, Christiana (Hrsg.): Industrie 4.0: Ein praxisorientierter Ansatz. Berlin: KS-Energy, S. 17-39.
Kolerus, Josef/Wassermann, Johann (2017): Zustandsüberwachung von Maschinen: Das Lehr- und Arbeitsbuch für den Praktiker. Renningen: expert.
Krauß, Jonathan et al. (2019): Maschinelles Lernen in der Produktion: Anwendungsgebiete und frei verfügbare Datensätze. In: Industrie 4.0 Management, 35. Jg., S. 39-42.
Lambertz, Björn (2023): Condition Monitoring. URL: https://maint-care.de/knowhow/condition-monitoring/#was-ist-condition-monitoring (17.07.2023).
Langs, Georg/Munro, Katherine/Wazir, Rania (2022): Maschinelles Lernen. Handbuch Data Science und KI. Carl Hanser Verlag GmbH & Co. KG, S. 220-261.
Maheswari, M./Gunasekharan, S. (2019): Types of Faults and Condition Monitoring Methods of Induction Motors. In: Irfan, Muhammad (Hrsg.): Advanced Condition Monitoring and Fault Diagnosis of Electric Machines. Hershey: IGI Global, S. 131-162.
Maytas, Kurt (2016): Instandhaltungslogistik: Qualität und Produktivität steigern. München: Hanser.
Meyer, Christian (2011): Aufnahme und Nachbearbeitung von Bahnen bei der Programmierung durch Vormachen von Industrierobotern (Dissertation; Konstruktions-, Produktions- und Fahrzeugtechnik). Heimsheim: Jost Jetter.
Mobley, R. Keith (1998): Condition based maintenance. In: Davies, A. (Hrsg.): Handbook of Condition Monitoring: Techniques and Methodology. Dordrecht: Springer Netherlands, S. 35-53.
Morales-Forero, Andres/Bassetto, Samuel. (2019). Case Study: A Semi-Supervised Methodology for Anomaly Detection and Diagnosis. 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM).
Mühlnickel, Helmut et al. (2018): Smart Maintenance. In: Reichel, Jens/Müller, Gerhard/Haeffs, Jean (Hrsg.): Betriebliche Instandhaltung. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 349-360.
Müller, Andreas C./Guido, Sarah (2016): Introduction to Machine Learning with Python : A Guide for Data Scientists. Sebastopol: O'Reilly Media.
Naeem, Samreen et al. (2023): An Unsupervised Machine Learning Algorithms: Comprehensive Review. 13. Jg., S. 911-921.
Nagel, Matthias (2018): Predictive Maintenance: Zukunftsweisender Ansatz für mehr Effektivität und Effizienz in der Instandhaltung. In: Klein, Andreas (Hrsg.): Modernes Produktionscontrolling für die Industrie 4.0: Konzepte, Instrumente und Kennzahlen. Freiburg: Haufe Lexware.
Nguyen, Quoc-Thông et al. (2021): Decision Support Systems for Anomaly Detection with the Applications in Smart Manufacturing: A Survey and Perspective. In: Tran, Kim Phuc (Hrsg.): Machine Learning and Probabilistic Graphical Models for Decision Support Systems. Boca Raton: CRC Press.
Pérez-Juárez, María A. et al. (2022): How Artificial Intelligence Can Enhance Predictive Maintenance in Smart Factories. In: Ahmad, Muneer/Zaman, Noor (Hrsg.): Empowering Sustainable Industrial 4.0 Systems With Machine Intelligence. Hershey: IGI Global, S. 86-100.
Pittino, Federico et al. (2020): Automatic Anomaly Detection on In-Production Manufacturing Machines Using Statistical Learning Methods. In: Sensors, 20. Jg. (8), S. 2344.
Qamar, Usman/Raza, Muhammad Summair (2023): Data Science Programming Language. In: Qamar, Usman/Raza, Muhammad Summair (Hrsg.): Data Science Concepts and Techniques with Applications. Cham: Springer International Publishing, S. 353-391.

Raschka, Sebastian (2017): Machine Learning mit Python: Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning. Frechen: mitp.
Sah, Shagan (2020): Machine Learning: A Review of Learning Types. Preprints.
Singh, Aishwarya (2018): KNN algorithm: Introduction to K-Nearest Neighbors Algorithm for Regression. URL: https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/ (28.06.2023).
Sullivan, GP et al. (2010): Operations & Maintenance Best Practices – A Guide to Achieving Operational Efficiency (Release 3.0). Richland: Pacific Northwest National Laboratory.
Tyagi, Kanishka et al. (2022): Unsupervised learning. In: Pandey, Rajiv et al. (Hrsg.): Artificial Intelligence and Machine Learning for EDGE Computing. London: Academic Press, S. 33-52.
Universal Robots Support (2023): Real-Time Data Exchange (RTDE) Guide. URL: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/ (02.07.2023).
Verdhan, Vaibhav (2020): Supervised Learning with Python: Concepts and Practical
Implementation Using Python. Limerick: Apress.
Wiederhold, Gio/McCarthy, John (1992): Arthur Samuel: Pioneer in Machine Learning. In: IBM Journal of Research and Development, 36. Jg., S. 329-331.