

Hochschule für angewandte Wissenschaften Coburg  
Fakultät Elektrotechnik und Informatik

Studiengang: Informatik-Bachelor

Bachelorarbeit

Klassifikation von Verkehrsteilnehmern auf Basis realer Positionszeitreihen mit Verfahren des maschinellen Lernens

Lennart Köpper

Abgabe der Arbeit: 10. 09. 2023

Betreut durch:  
Prof. Dr. Thomas Wieland, Hochschule Coburg

# Abstract

# Inhaltsverzeichnis

Abstract 2

Inhaltsverzeichnis 3

Abbildungsverzeichnis 5

Tabellenverzeichnis 6

Codeverzeichnis 7

Abkürzungsverzeichnis 8

1 Einleitung 9

1.1 Einleitung und Projekthintergrund 9

1.2 Motivation und Zielsetzung 9

1.3 Aufbau der Arbeit 11

2 Theoretischer Hintergrund 12

2.1 Global Navigation Satellite System 12

2.2 Map-Matching 13

2.3 Maschinelles Lernen und Klassifikation 15

2.4 Eingesetzte Klassifikationsverfahren 17

2.4.1 Support Vector Machine 17

2.4.2 Decision-Tree und Random Forest 19

2.5 Klassifikation mit Künstlichen Neuronalen Netzen 20

2.5.1 Aufbau eines künstlichen Neuronalen Netzes 21

2.5.2 Künstliche Neuronen und Aktivierungsfunktionen 22

2.5.3 Training neuronaler Netze 24

2.5.4 Rekurrente Neuronale Netze 24

2.6 Bewertungsmaße für Klassifikatoren 24

2.6.1 Konfusionsmatrizen 24

2.6.2 Accuracy / Genauigkeit 25

2.6.3 Precision / Relevanz 26

2.6.4 Recall / Sensitivität 26

2.6.5 F1-Score 27

2.7 Eingesetzte Technologien und Frameworks 27

2.7.1 Valhalla Map-Matching API 27

2.7.2 Python 28

2.7.3 scikit-learn 28

2.7.4 Keras und TensorFlow 29

3 Verwandte Arbeiten 31

3.1 Vorangegangene Abschlussarbeiten 31

3.2 Vehicle Classification from Low-Frequency GPS Data with Recurrent Neural Networks 32

3.3 Vehicle Classification using GPS Data 33

4 Datengrundlage 35

4.1 Gewinnung der Ausgangsdaten 35

4.2 Beschreibung der Ausgangsdaten 35

4.3 Weiterverarbeitung zu Bewegungsdaten 35

5 Anforderungen und Gesamtkonzept der Klassifikation 36

6 Umsetzung des Map-Matchings 37

6.1 Einrichtung der Valhalla-Engine 37

6.2 Umsetzung und Evaluierung des Road-Snappings 37

6.3 Vorklassifikation zur Bestimmung des korrekten Matching-Modus 37

6.3.1 Erzeugung des Trainingsdatensatzes 37

6.3.2 Auswahl des Klassifikators 37

6.3.3 Evaluierung 37

7 Umsetzung der Klassifikation 38

7.1.1 Erzeugung der Trainingsdatensätze (inkl. Vorverarbeitungen) 38

7.1.2 Training und Optimierung der Modelle 38

7.1.3 Vorstellung der besten Klassifikatoren 38

8 Evaluierung und Diskussion 39

9 Zusammenfassung und Ausblick 40

Quellenverzeichnis 41

Anhang A 1. Test 44

# Abbildungsverzeichnis

[Abb. 1: Ermittlung der GNSS-Position als Schnittpunkt mehrerer Kugeln [5] 12](#_Toc139037433)

[Abb. 2: Map-Matching von aufgenommenen GNSS-Punkten auf das Straßennetz [7] 13](#_Toc139037434)

[Abb. 3: GNSS-Punkte und zugehörige Kandidaten 14](#_Toc139037435)

[Abb. 4: Mögliche Übergänge und optimaler Pfad [7] 15](#_Toc139037436)

[Abb. 5: Lineare Separierung durch suboptimale und optimale Hyperebene [10, S. 156] 17](#_Toc139037437)

[Abb. 6: Effekt des Regularisierungsparameters C bei Soft-Margin-SVMs [10, S. 157] 18](#_Toc139037438)

[Abb. 7: Hinzufügen von Merkmalen, um Datenpunkte separierbar zu machen [10, S. 159] 18](#_Toc139037439)

[Abb. 8: Beispiel für einen Entscheidungsbaum [vgl. 10, S. 178] 19](#_Toc139037440)

[Abb. 9: Aufbau eines Künstlichen Neuronalen Netzes 21](#_Toc139037441)

[Abb. 10: Veranschaulichung eines künstlichen Neurons 22](#_Toc139037442)

[Abb. 11: Aktivierungsfunktionen: tanh(z),und 23](#_Toc139037443)

[Abb. 12: Beispiel für eine zweiklassige Konfusionsmatrix; eigene Abbildung 25](#_Toc139037444)

# Tabellenverzeichnis

# Codeverzeichnis

# Abkürzungsverzeichnis

4G Vierte Generation des Mobilfunks

5G Fünfte Generation des Mobilfunks

API Application Programming Interface

DQLN Deep-Q-Learning-Netz

FFN Feed-Forward-Netz

GNSS Global Navigation Satellite System

HMM Hidden-Markov-Model

HTTP Hypertext Transfer Protocol

IZK Innovations-Zentrum Kronach e.V.

JSON JavaScript Object Notation

LIDAR Light Detection and Ranging

LKW Lastkraftwagen

LSTM Long-Short-Term-Memory

MM Map-Matching

PKW Personenkraftwagen

RADAR Radio Detection and Ranging

ReLU Rectified Linear Unit

RNN Rekurrentes Neuronales Netz

SVM Support Vector Machine

# Einleitung

## Einleitung und Projekthintergrund

Schone heute besitzen Mobilität und Konnektivität einen großen Einfluss auf den Alltag und die Lebensqualität vieler Menschen. Zwei bedeutende Trends, die im Zusammenhang mit diesen beiden Begriffen immer wieder Erwähnung finden, sind hierbei das *autonome Fahren* und *5G*. Dass diese beiden Trends eng miteinander in Verbindung stehen, legt ein Bericht des Beratungsunternehmens Gartner nahe, nach welchem der Ausbau von 5G-Netzwerken von großer Bedeutung für die weiteren Entwicklungen des autonomen Fahrens ist. Der wichtigste Grund hierfür ist demnach die immer weiter steigende Menge von Fahrzeug- und Sensordaten, welche durch zunehmend autonomisierte Fahrzeuge nicht nur verarbeitet, sondern auch im Zuge von *Car-2-X-Kommunikation* zeitkritisch mit anderen Akteuren geteilt werden muss. Hierdurch ist davon auszugehen, dass 5G durch die bis zu zehnfach höheren Übertragungsraten (im Vergleich mit dem heutigen 4G-LTE) in Zukunft nicht nur Einfluss auf die allgemeine Konnektivität und das Infotainment von Fahrzeugen nehmen, sondern auch ganz konkret zur weiteren Steigerung der Sicherheit im autonomen Fahren beitragen wird [vgl. 1].

In diesem Kontext wurde auch das Projekt *5GKC* ins Leben gerufen. Das Ziel dieses Projektes besteht darin in der oberfränkischen Stadt Kronach ein 5G-basiertes Testfeld für das autonome Fahren zu schaffen, in welchem verschiedene Anwendungen von 5G im autonomen Fahren erforscht, entwickelt und im öffentlichen Verkehr erprobt werden können. Der wesentliche Fokus liegt dabei auf den Bereichen Steuerung, Überwachung, Mensch-Fahrzeug-Interaktion und Kommunikation. Kooperationspartner des Projektes ist dabei neben dem Fraunhofer-Institut für integrierte Schaltungen, dem Innovations-Zentrum Region Kronach e.V., und Valeo, dem Weltmarktführer für Fahrerassistenzsysteme, auch die Hochschule für angewandte Wissenschaften Coburg. Letztere dient als akademisches Zentrum der Region, als wichtiger Impulsgeber und Partner bei verschiedenen Forschungsprojekten. Zu diesen zählt auch ein Projekt zur *erweiterten Umfeldwahrnehmung*, in welchem auch die vorliegende Arbeit zu verorten ist[vgl. 2].

## Motivation und Zielsetzung

Im Bereich des autonomen Fahrens und der Fahrerassistenzsysteme spielt die Wahrnehmung des Fahrzeugumfeldes, zu welchem neben der unbelebten Umgebung insbesondere auch andere Verkehrsteilnehmer gehören, eine entscheidende Rolle. Üblicherweise kommen hierfür verschiedenste Sensorsysteme, wie *Ultraschall*-, *RADAR*- und *LIDAR*-Sensoren, sowie kamerabasierte Verfahren, wie das Stereo- und das Maschinelle-Sehen zum Einsatz, welche allesamt spezifische Vor- und Nachteile mit sich bringen. Die Vorteile können hierbei durch die sogenannte *Sensordatenfusion*, also durch die Kombination von Informationen aus verschieden Sensorsystemen, oft gut zusammengeführt werden, was auch häufig zum Ausgleich spezifischer Nachteile einzelner Sensoren führt [vgl. 3, S. 440]. Ein wesentlicher Nachteil bleibt bei den heute verbreiteten Systemen jedoch in jedem Fall bestehen, denn alle beschränken sich auf ein stark limitiertes Umfeld des Fahrzeuges, welches sich im Fall von kamerabasierten Verfahren bspw. auf das Sichtfeld der Kameras beschränkt. Dieses Sichtfeld kann jedoch insbesondere im städtischen Verkehr sehr stark eingeschränkt sein, woraus sich ein Bedarf nach Systemen begründet, welche auch zur erweiterten Umfeldwahrnehmung eingesetzt werden können, um die bestehenden Systeme zu ergänzen und das autonome Fahren somit sicherer zu machen.

Im Zuge dieser Bachelorarbeit soll hierbei die Möglichkeit erforscht werden, Verkehrsteilnehmer auf Basis von sequenziell bereitgestellten *GNSS-Koordinaten* durch den Einsatz *maschineller Lernverfahren* in verschiedene Typen zu klassifizieren. Hierbei soll die Klassifikation möglichst realitätsnah, also insbesondere unter der Nutzung von realen Positionssequenzen aus dem Straßenverkehr, umgesetzt werden. Aus vorangegangenen Forschungsarbeiten ist dabei bekannt, dass reale Positionsdaten, die über das GNSS ermittelt werden, häufig ungenau und zu einem gewissen Grad verrauscht sind, weshalb im Verlauf dieser Arbeit zunächst einmal die folgende Frage untersucht werden soll:

*Wie können reale Positionsdaten, die Ungenauigkeiten und Rauschen aufweisen, so vorverarbeitet werden, dass sie sich gut für den Einsatz maschineller Lernverfahren eignen?*

Im Anschluss daran ergibt sich dann die primäre Forschungsfrage der Arbeit:

*Welche Verfahren des maschinellen Lernens sind für die Klassifizierung von Verkehrsteilnehmern auf Basis von sequenziellen Positionsdaten geeignet?*

Sollten die Ergebnisse dieser Arbeit als positiv zu bewerten sein, ließe sich auf Basis der Antworten zu diesen beiden Fragen, ein System entwickeln, welches in der Lage ist, Informationen zu anderen Verkehrsteilnehmern abzuleiten, während sich diese noch deutlich außerhalb der Reichweite der bisher üblichen Sensorsysteme von autonomen Fahrzeugen befinden.

## Aufbau der Arbeit

Um die im letzten Abschnitt formulierten Forschungsfragen dieser Arbeit zu beantworten, soll im nachfolgenden Kapitel 2 *Theoretischer Hintergrund* zunächst auf die wesentlichen Grundlagen eingegangen werden, die für ein Verständnis der weiteren Arbeit notwendig sind. Anschließend wird im Kapitel 3 *Verwandte Arbeiten* auf den bisherigen Stand der Forschung Bezug genommen. Es wird aufgezeigt welche verwandten Ansätze bei der Klassifikation von Verkehrsteilnehmern aus Positionsdaten bereits existieren und worin sich diese von dem Ansatz der vorliegenden Arbeit unterscheiden. In Kapitel 4 *Datengrundlage* wird der Datensatz vorgestellt, welcher die Basis für die in dieser Arbeit umgesetzten Klassifikation bildet. In Kapitel 5 *Anforderungen und Gesamtkonzept* folgt basierend darauf die Vorstellung der wichtigsten Anforderungen und des angestrebten Gesamtkonzeptes der Klassifikation. Die nachfolgenden Kapitel 6 *Umsetzung des Map-Matchings* und 7 *Umsetzung der Klassifikation* beschreiben die Realisierung der vielversprechendsten Verfahren zur Vorverarbeitung bzw. zur Klassifikation der Positionssequenzen. Die Ergebnisse werden im Anschluss daran im Kapitel 8 *Evaluierung und Diskussion* auf Basis unabhängiger Testdaten miteinander verglichen und hinsichtlich ihrer Qualität bewertet. Den Abschluss bildet das Kapitel 9 *Zusammenfassung und Ausblick*, in welchem ein Fazit gezogen und auf weiteres Forschungspotenzial hingewiesen wird.

# Theoretischer Hintergrund

## Global Navigation Satellite System

*Global Navigation Satellite System* (dt. globales Navigationssatellitensystem) ist der Sammelbegriff für die zahlreichen satellitengestützten Systeme, welche heute für den Zweck der Navigation und Standortbestimmung eingesetzt werden. Hierzu zählen neben den global verfügbaren Systemen, wie dem *Global Positioning System* (GPS)der Vereinigten Staaten von Amerika, dem *GLONASS* der russischen Föderation, dem *Galileo*-System der Europäischen Union und dem *BeiDou*-System der Volksrepublik China auch weitere regional beschränkte Unterstützungssysteme [vgl. 4, S. 2].

Die Standortbestimmung basiert bei allen Systemen auf dem sogenannten *Time-of-Arrival-Ranging*. Hierbei werden ausgehend von den Satelliten Signale (*Ranging-Codes)* gesendet, welche die aktuelle Position des sendenden Satelliten und den Sendezeitpunkt umfassen. Nachdem die Signale durch ein GNSS-fähiges Gerät empfangen wurden, werden die Signallaufzeiten errechnet. Diese werden im Anschluss mit der Signalgeschwindigkeit multipliziert, um die Entfernung zu den Satelliten zu bestimmen [vgl. 4, S. 19ff.]. Ist die Entfernung zu vier Satelliten (und deren Position) bekannt, kann daraus, wie in Abb. 1 veranschaulicht, der Standort des empfangenden Geräts, in Form von Koordinaten und Höhe, ermittelt werden. Theoretisch würden sogar drei Satelliten ausreichen, wenn davon ausgegangen werden kann, dass die Uhren der Satelliten und des Empfängers perfekt synchronisiert sind, was in der Praxis jedoch meist nicht der Fall ist [vgl. 4, S. 2].

Ein Bild, das Kreis, Kunst, Design enthält.

Automatisch generierte Beschreibung

Abb. : Ermittlung der GNSS-Position als Schnittpunkt mehrerer Kugeln [5]

Die Genauigkeit der Positionsbestimmung mittels GNSS ist durch viele Faktoren abhängig. Bspw. sollten die Satelliten für eine genaue Bestimmung in Relation zum Empfänger möglichst gleichmäßig verteilt sein. Sind die genutzten Satelliten eng beisammen oder ungleichmäßig verteilt kann die Positionsermittlung sehr ungenau werden. Hinzu kommen weitere Fehlerquellen, wie bspw. Signalbrechungen in der Atmosphäre oder die Reflektion von Signalen an der Erdoberfläche oder auch Gebäuden. Letztere können, ebenso wie bspw. Berge, auch das Signal einzelner Satelliten blockieren, und hiermit die Auswahl der Satelliten zur Standortbestimmung einschränken. Einige dieser Fehlerfaktoren, insbesondere die atmosphärisch-bedingten, können inzwischen durch Zusatzsysteme wie dem *European Geostationary Navigation Overlay Service* korrigiert werden. Diese senden über geostationäre Satelliten Korrektursignale, welche heutzutage von allen GNSS-fähigen Geräten standardmäßig bei der Berechnung des Standortes miteinbezogen werden. Hierdurch sind durchschnittliche Genauigkeiten von etwa 5-15m erreichbar [vgl. 6].

## Map-Matching

Wie im letzten Abschnitt bereits erwähnt, sind Positionen und somit auch Positionssequenzen, welche über das GNSS ermittelt werden im Regelfall fehlerbelastet. Zum einen durch Messungenauigkeiten, aber natürlich auch durch Abtastfehler, also den Verlust von Information zwischen zwei aufgenommenen GNSS-Punkten. Diese Fehler zu ignorieren kann potenziell zu falschen Analysen und entsprechend auch zu falschen Schlussfolgerungen in den Klassifikationsmodellen führen. *Map-Matching* ist ein möglicher Ansatz derartige Fehler zu minimeren. Gemeint ist hiermit das Abbilden von aufgenommenen Positionssequenzen auf eine digitale Repräsentation eines Straßen- und Wegenetzes, das üblicherweise in Form eines *Graphen* vorliegt. [vgl. 7, S. 1]. Abb. 2 zeigt ein Beispiel für ein solches Map-Matching.

Ein Bild, das Reihe, Screenshot, Diagramm, Karte enthält.

Automatisch generierte Beschreibung

Abb. : Map-Matching von aufgenommenen GNSS-Punkten auf das Straßennetz [7]

Beim Versuch eine Positionssequenz auf ein Straßen- und Wegenetz zu projizieren, handelt es sich im Grunde um ein Suchproblem. Gesucht wird hierbei diejenige Abfolge an Straßensegmenten, welche am wahrscheinlichsten zu der gegebenen Sequenz an GNSS-Punkten passt. Hierfür nutzen die meisten modernen Map-Matching-Services einen komplexen Algorithmus, der auf einem *Hidden-Markov-Model* (HMM*)* basiert und im Folgenden vereinfacht erläutert werden soll. Im ersten Schritt werden, wie in Abb. 3 veranschaulicht, für alle GNSS-Punkte passende Straßensegmente, sogenannte *Kandidaten*, ermittelt. Diese werden in Abhängigkeit verschiedener Parameter in einem gewissen Umkreis um die Punkte herum bestimmt [vgl. 7, S. 3].

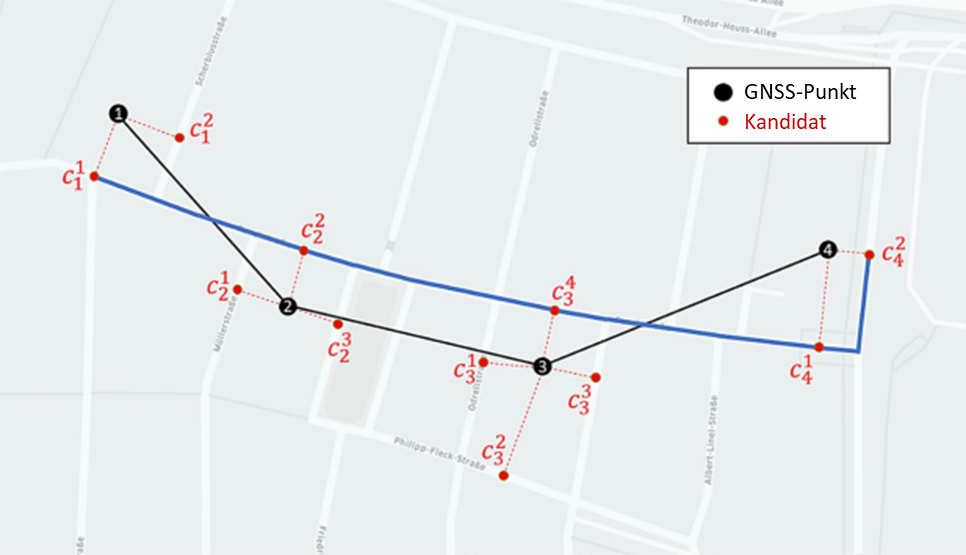


Abb. : GNSS-Punkte und zugehörige Kandidaten

Im nächsten Schritt wird das HMM eingesetzt. Dabei werden die gegebenen GNSS-Punkte als *beobachtbare Zustände* (measurements) betrachtet. Die zugehörigen Kandidaten bilden die Menge der *unbeobachtbaren Zustände* (hidden states). Gesucht wird nun die wahrscheinlichste Sequenz an Kandidaten, als Folge von Zuständen mit den höchsten Übergangswahrscheinlichkeiten, wobei die möglichen Zustandsübergänge durch die gegebenen Verbindungen des Straßennetzwerks definiert sind. Bei der Berechnung dieser Übergangswahrscheinlichkeiten kommen verschiedene Metriken zum Einsatz die unter anderem dazu führen, dass Kandidaten, welche sich näher an den aufgezeichneten Punkten befinden, als wahrscheinlicher angenommen werden. Außerdem werden Folgen von Kandidaten bevorzugt, die möglichst wenig komplexe Bewegungs-Manöver voraussetzen. Wurden die Übergangswahrscheinlichkeiten berechnet, so definieren diese einen Suchraum, in welchen nun der optimale Pfad mit Hilfe eines Suchalgorithmus, wie bspw. dem *A\*-* oder dem *Dijkstra-Algorithmus* bestimmt wird. Abb. 4 zeigt alle möglichen Übergänge zwischen den Kandidaten der in Abb. 3 gegebenen Positionssequenz. Der durch den Suchalgorithmus gefundene (optimale) Pfad und die zugehörigen Kandidaten sind hervorgehoben [vgl. 7, S. 3f.].

Ein Bild, das Kreis, Diagramm, Reihe, Design enthält.

Automatisch generierte Beschreibung

Abb. : Mögliche Übergänge und optimaler Pfad [7]

Ein großer Vorteil der HMM-basierten Services ist, dass sie sich sowohl für den Einsatz in *Online-* als auch in *Offline-Szenarien* eignen [vgl. 8, S. 4]. Unter Ersterem versteht man das Map-Matching unter Echtzeit-Bedingungen. Das heißt die Punkte werden bereits während der Aufnahme auf das Straßennetz abgebildet, wobei natürlich nur die bisher bekannten Punkte genutzt werden können, um die wahrscheinlichsten Kandidaten für den aktuellen Punkt zu ermitteln. Im Gegensatz dazu können in Offline-Szenarien auch alle nachfolgenden Punkte miteinbezogen werden, da das Map-Matching erst dann stattfindet, wenn die Aufnahme abgeschlossen wurde [vgl. 7, S. 1f.].

In Bezug auf die zugrundeliegenden Positionssequenzen zeigt eine Untersuchung von Pingfu Chao et al. [8], dass die Aufnahmerate einen wesentlichen Einfluss auf die Qualität des Map-Matchings besitzt. Demnach führen nicht nur zu niedrige, sondern auch zu hohe Aufnahmeraten (im Bereich von 1Hz) zu potenziell minderwertigen Matching-Ergebnissen.

## Maschinelles Lernen und Klassifikation

Eine weit verbreitete Definition des Begriffs *Maschinelles Lernen* (engl. *Machine Learning*)wurde 1959 durch Arthur Lee Samuel, einem amerikanischen Pionier auf dem Gebiet der Computerspiele und künstlichen Intelligenz, geprägt [vgl. 9]. Er definierte das Maschinelle Lernen als Fachgebiet, welches Computern die Fähigkeit verleiht, Probleme zu lösen, ohne explizit dafür programmiert zu werden. Um dies zu erreichen, werden beim Maschinellen Lernen verschiedene Algorithmen eingesetzt, welche dazu in der Lage sind, sich im Hinblick auf eine gegebene Aufgabe selbstständig zu verbessern, indem sie durch die Verarbeitung von Beispieldaten *Erfahrungen* sammeln. Die Menge der verwendeten Beispieldaten nennt man dabei *Trainingsdatensatz*. Ein einzelnes Beispiel wird als *Trainingsdatenpunkt* bezeichnet. Entsprechend nennt man den Lernprozess *Training*. [vgl. 10, S. 4] Die meisten Algorithmen sind hierbei modellbasiert, das heißt sie stützen sich auf (je nach Algorithmus und gewählten *Hyperparametern*) mehr oder weniger komplexe mathematische Modelle, deren *Modellparameter* im Zuge des Trainings so angepasst werden, dass sie im besten Fall Muster und Gesetzmäßigkeiten in den Daten widerspiegeln. Das trainierte *Modell* kann anschließend auf die konkrete Aufgabenstellung angewandt werden, wobei es dazu in der Lage ist auch mit neuen Datenpunkten umzugehen [vgl. 10, S. 21ff.].

Maschinelles Lernen kommt meist dann zum Einsatz, wenn die konventionellen Methoden der Informatik an ihre praktischen Grenzen stoßen. Dies ist insbesondere der Fall, wenn Aufgabenstellungen vorliegen, deren Lösungen nur durch einen sehr komplexen Regelsatz realisiert werden können. Dann ist es oftmals einfacher zu versuchen über große Datenmengen und maschinelle Lernalgorithmen Erkenntnisse zu gewinnen oder gar Problemlösungen zu finden [vgl. 10, S. 6]. Eine für das Maschinelle Lernen typische Art von Problem ist hierbei die *Klassifikation*. In diesem Kontext versteht man unter einem Klassifikationsproblem ein Vorhersageproblem, bei welchem die gesuchte Vorhersage verschiedene Klassen repräsentiert [vgl. 11, S. 1]. Vereinfacht gesagt werden bei einer Klassifikation also Objekte in verschiedene Kategorien eingeteilt. Im Zuge dieser Arbeit sollen bspw. Verkehrsteilnehmer in die Klassen *Fußgänger*, *Fahrradfahrer*, *Motorradfahrer* und *Autos* eingeteilt werden.

Algorithmen und Modelle, welche für die Vorhersage von Klassen eingesetzt werden können, nennt man *Klassifikationsverfahren* bzw. *Klassifikatoren*. Beim Training werden den Klassifikationsverfahren zusätzlich zu den Trainingsdatenpunkten auch die zugehörigen Lösungen, also die vorherzusagenden Klassen, auch *Labels* genannt, mitgegeben. Da die Algorithmen somit während des Lernens angeleitet werden, gehören Klassifikationsverfahren im Allgemeinen dem Teilbereich des *Überwachten Lernens* (engl. *Supervised Learning*) an. Neben diesem unterscheidet man noch die Bereiche *Halbüberwachtes* (engl. *Semi-Supervised*), *Unüberwachtes* (engl. *Unsupervised*) und *Bestärkendes Lernen* (engl. *Reinforcement Learning*) [vgl. 10, S. 9ff.]. Diese sind jedoch für die vorliegende Arbeit nicht von Relevanz.

## Eingesetzte Klassifikationsverfahren

Es existieren zahlreiche Maschinelle Lernverfahren, welche sich für den Einsatz als Klassifikationsverfahren eignen. Im Folgenden soll dabei lediglich auf die Klassifikationsverfahren eingegangen werden, welche im Zuge dieser Arbeit zum Einsatz kommen: *Support Vector Machine* und *Random Forest*. Da das letztere Verfahren dabei auf sogenannten *Decision-Trees* basiert, werden auch diese kurz erläutert.

### Support Vector Machine

Die Support-Vector-Machine (SVM) ist ein weit verbreitetes, mächtiges und flexibles maschinelles Lernverfahren, welches sich hervorragend für die binäre Klassifikation eignet. Basis der Klassifikation ist hierbei die lineare Separierung zweier Klassen im Merkmalsraum durch das Modellieren einer hierfür möglichst optimalen Hyperebene [vgl. 12, S. 283]. Zur besseren Vorstellung, was hiermit gemeint ist, zeigt Abb. 5 beispielhaft eine binäre Klassifikation zweier Schwertlilienarten anhand der Merkmale *Länge* und *Breite der Kronblätter*, die einen zweidimensionalen Merkmalsraum aufspannen. Links sind suboptimale Hyperebenen eingezeichnet, die zum Teil nicht einmal zu einer Separierung der Klassen führen. Rechts ist eine Hyperebene zu sehen, wie sie eine SVM finden würde.

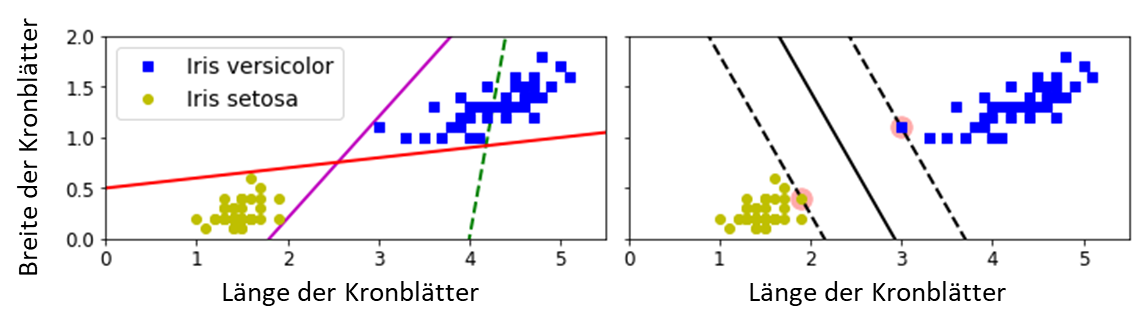


Abb. : Lineare Separierung durch suboptimale und optimale Hyperebene [10, S. 156]

Zu beachten ist, dass die optimale Hyperebene die Daten nicht nur separiert, sondern auch den größtmöglichen Abstand zwischen den Trainingsdatenpunkten der einzelnen Klassen hält. Dabei stützt sich die Hyperebene vollständig auf diejenigen Datenpunkte der beiden Klassen, die dieser am nächsten sind. Diese Datenpunkte werden als *Stützvektoren* (engl. *Support Vectors*) bezeichnet. Da alle anderen Datenpunkte für die Platzierung der Hyperebene irrelevant sind, können diese nach dem Training aus dem Modell verworfen werden, wodurch SVMs sehr speichereffizient sind [vgl. 12, S. 283f.].

In den meisten Anwendungsfällen ist es nicht möglich eine Hyperebene zu finden, die zwei Klassen fehlerfrei linear separieren kann. Außerdem sind die Hyperebenen von klassischen SVMs oftmals durch Ausreiser gestützt, welche dazu führen, dass die SVM die Daten schlecht verallgemeinert. Deswegen kommen heutzutage vor allem sogenannte *Soft-Margin-SVMs* zum Einsatz, die in Abhängigkeit von einem Hyperparameter in einem gewissen Maße Fehlklassifikationen zulassen. Dieser Hyperparameter wird als bezeichnet. Wie in Abb. 6 zu sehen, führt ein höheres im Allgemeinen zu weniger Fehlklassifikationen auf den Trainingsdaten. Allerdings werden die Daten dadurch auch potenziell weniger verallgemeinert, was zu einer schlechteren Qualität des Modells auf den Testdaten führen kann [vgl. 10, S. 156f.]. Man spricht in solch einem Fall von einem *Overfitting* in Bezug auf die Trainingsdaten.

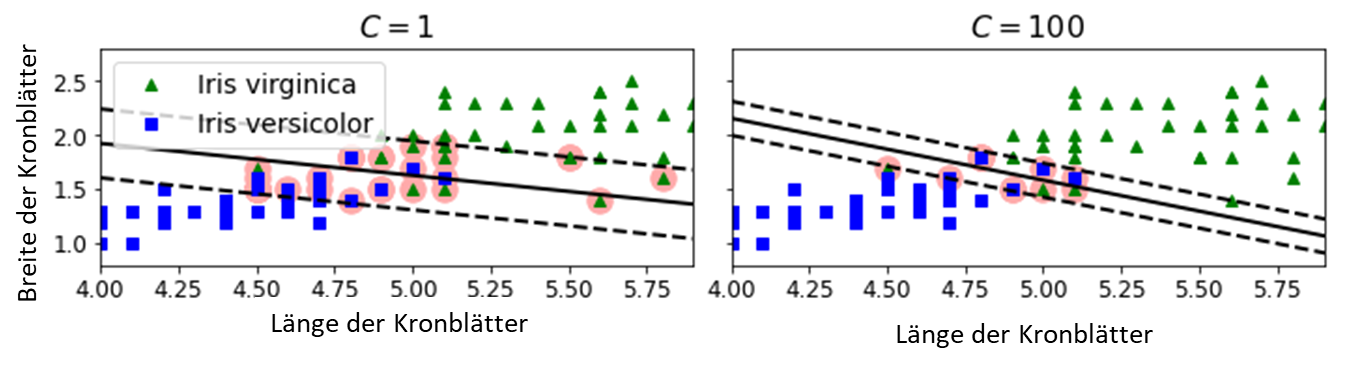


Abb. : Effekt des Regularisierungsparameters C bei Soft-Margin-SVMs [10, S. 157]

Oftmals benötigen SVMs zum Finden einer linear separierenden Hyperebene einen Raum der mehr Dimensionen als der gegebene Merkmalsraum aufweist. Um diesen zu erhalten können die Datenpunkte durch die Anwendung einer sogenannten *Kernel-Funktion* in einen solchen höherdimensionalen Raum abgebildet werden [vgl. 12, 285f.]. Abb. 7 dient erneut der Veranschaulichung. Der nicht separierbare Datensatz links, der lediglich das Merkmal enthält, wird durch das Hinzufügen des Merkmals linear separierbar.

Ein Bild, das Reihe, Diagramm enthält.

Automatisch generierte Beschreibung

Abb. : Hinzufügen von Merkmalen, um Datenpunkte separierbar zu machen [10, S. 159]

Durch die Abbildung über die Kernel-Funktion müssen die Merkmale anders als im hier gezeigten Beispiel jedoch nicht explizit zum Merkmalsraum hinzugefügt werden. Die Transformation in diesen erfolgt implizit. Die Wahl der Kernel-Funktion stellt hierbei einen weiteren Hyperparameter dar. Verbreitet sind neben dem *linearen Kernel* vor allem verschiedengradige *polynomielle* und der *Gaussche RBF-Kernel*, welcher mit einem weiteren Hyperparamter (Gamma) einhergeht [vgl. 10, S. 160ff.]. Auch ist ein Regularisierungsparameter. Ein hohes führt dazu, dass die Hyperebene unregelmäßiger und um einzelne Datenpunkte herum verläuft. Ein niedriges führt zu einer weicheren und damit oft besser verallgemeinernden Hyperebene [vgl. 10, S. 162].

### Decision-Tree und Random Forest

Decision-Trees sind mächtige und dabei dennoch gut interpretierbare Klassifikationsverfahren. Sie basieren auf einer gewöhnlichen Baumstruktur, bestehend aus der *Wurzel* sowie einer Menge an *Knoten*, *Zweigen* und *Blättern*. Der Baum bildet dabei ausgehend vom Wurzelknoten aufeinanderfolgende merkmalsbasierte Entscheidungsregeln ab, welche ggf. über Teilbäume zu den Blättern des Baumes führen. Diese repräsentieren schließlich die verschiedenen Kategorien des zugehörigen Klassifikationsproblems [vgl. 13, S. 273]. Abb. 8 zeigt einen solchen Entscheidungsbaum. Hierbei wird das Beispiel der Schwertlilienklassifikation aus dem letzten Abschnitt erneut aufgegriffen.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Abb. : Beispiel für einen Entscheidungsbaum [vgl. 10, S. 178]

Ein häufig verwendetes Verfahren zum Aufbau solcher Bäume ist der *CART* (*Classification and Regression Tree*) *Algorithmus*. Die Grundidee dieses Algorithmus ist dabei recht simpel: Er teilt die Menge der Trainingsdaten basierend auf einem Merkmal und einem Schwellenwert in zwei Untermengen auf. Dabei werden und so gewählt, dass die resultierenden Untermengen möglichst *rein* sind, also im besten Fall nur eine Klasse enthalten. Die Untermengen werden schließlich rekursiv nach demselben Prinzip aufgeteilt bis die maximale Tiefe oder eine andere Abbruchbedingung basierend auf weiteren Hyperparametern erreicht wurde. Zu diesen Hyperparametern gehören bspw. die minimale Anzahl an Datenpunkten, die ein Knoten enthalten muss, um weiter aufgeteilt werden zu dürfen, oder die minimale Menge an Datenpunkten, die ein Blatt mindestens enthalten muss. Alle drei erwähnten Hyperparameter sind Regularisierungsparameter, die dazu genutzt werden können ein Overfitting der Daten zu reduzieren. Auch das genutzte Reinheitsmaß ist ein Hyperparameter. Verbreitet ist die Nutzung der *Gini-Unreinheit* oder der *Entropie* [vgl. 10, S. 181ff.].

Decision-Trees bilden die Basis für ein weiteres Klassifikationsverfahren: den Random-Forests. Bei einem Random-Forest handelt es sich um ein *Ensemble* an Decision-Trees, welche jeweils auf Basis einer zufälligen Auswahl an Trainingsdatenpunkten erstellt werden. Außerdem werden auch die zur Aufteilung genutzten Merkmale zufällig ausgewählt. Eine Vorhersage trifft der Random-Forest, indem er alle Vorhersagen des Ensembles sammelt und anschließend diejenige Klasse vorhersagt, welche im Mehrheitsentscheid siegt. Das führt im Allgemeinen zu einer deutlich besseren und allgemeineren Vorhersage im Vergleich zu einem einzelnen Decision-Tree [vgl. 13, S. 274]. Wichtige Hyperparameter eines Random-Forests sind die Anzahl der enthaltenen Bäume, die Größe des Trainingsdatensatzes für einen einzelnen Baum und die Anzahl der zufällig auszuwählenden Merkmale, die für das Aufteilen einzelner Knoten in Betracht gezogen werden sollen [vgl. 10, S. 199f.]. Hinzu kommen die bereits erwähnten Hyperparameter für die Decision-Trees.

## Klassifikation mit Künstlichen Neuronalen Netzen

Künstliche Neuronale Netze (KNNs) sind komplexe maschinelle Lernmodelle, deren Funktionsweise in den Ursprüngen durch die Vernetzung von biologischen Neuronen in menschlichen Gehirnen inspiriert ist. Sie gelten als flexibel, mächtig sowie gut skalierbar und bilden die Grundlage für das heutzutage vielseitig eingesetzte *Deep Learning* zur Bewältigung äußerst komplexer Problemstellungen des Maschinellen Lernens. [vgl. 10, S. 281]. Im Folgenden soll zunächst ein Überblick über den Aufbau und die prinzipielle Funktionsweise von gewöhnlichen KNNs, sogenannten Feed-Forward-Netzen (FFNs), gegeben werden. Anschließend werden Rekurrente Neuronale Netze erläutert.

### Aufbau eines künstlichen Neuronalen Netzes

Im Allgemeinen bestehen Künstliche Neuronale Netze aus mehreren Verarbeitungseinheiten, welche in diesem Kontext als *Neuronen* bezeichnet werden. Diese Neuronen sind, wie in Abb. 9 schematisch dargestellt, in mehreren Schichten angeordnet, wobei die Ausgabe eines jeden Neurons einer Schicht an die Eingabe der Neuronen in der nächsten Schicht weitergereicht wird [vgl. 14, S. 6].

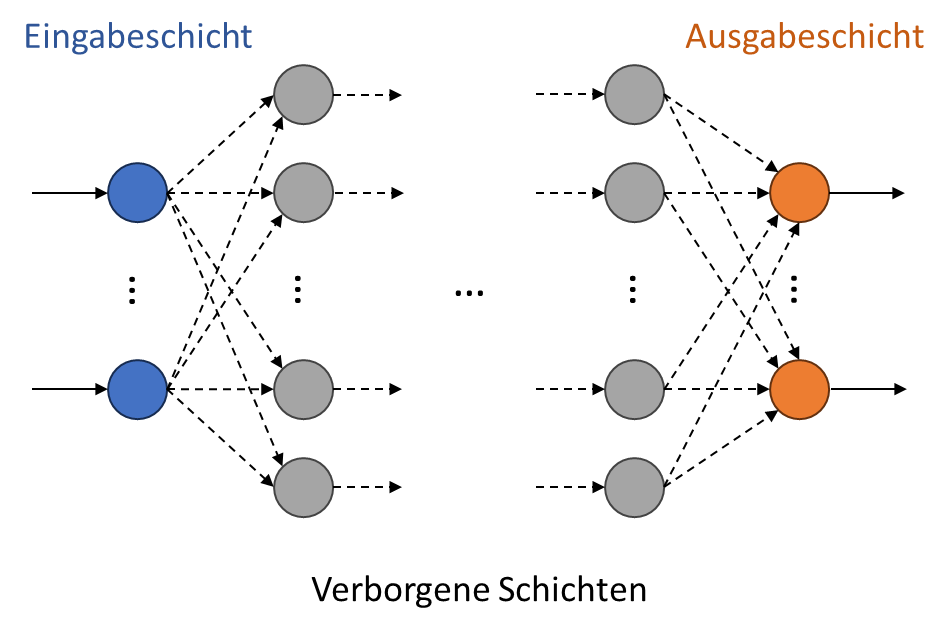


Abb. : Aufbau eines Künstlichen Neuronalen Netzes

Die erste Schicht eines KNNs bezeichnet man hierbei als *Eingabeschicht* (engl. *Input-Layer*) und die zugehörigen Neuronen als Eingabeneuronen (engl. *Input-Neurons*). Die Anzahl der Eingabeneuronen eines Netzes wird durch die Anzahl an gegebenen Merkmalen in den Trainingsdatenpunkten festgelegt. Die letzte Schicht eines KNNs wird analog als Ausgabeschicht (engl. *Output-Layer*) bezeichnet. Sie besteht aus den Ausgabeneuronen (engl. *Output-Neurons*) [vgl. 15, S. 11]. Die Anzahl der Ausgabeneuronen ist abhängig von der Problemstellung. Bei einem mehrklassigen Klassifikationsproblem verwendet man jedoch üblicherweise ein Neuron pro vorherzusagender Kategorie [vgl. 10, S. 295f.].

Etwas komplizierter ist der Aufbau des Netzes zwischen der Ein- und Ausgabeschicht. Die dort befindlichen Schichten bezeichnet man als *verborgene Schichten* (engl. *Hidden Layers*). Sowohl die Anzahl der verborgenen Schichten als auch die Anzahl der Neuronen in den einzelnen Schichten folgen keiner Faustregel [vgl. 15, S. 12]. Sie sind Hyperparameter, welche die Fähigkeit des Netzes beeinflussen, komplexe Muster und Konzepte in den Daten zu erlernen und abzubilden. Ein KNN mit mehr als zwei verborgenen Schichten gilt als tiefes neuronales Netzwerk (engl. *Deep Neural Network*) [vgl. 15, S. 37]. Komplexe Funktionen können sowohl durch tiefe als auch durch weite (viele Neuronen pro Schicht) KNNs modelliert werden. Im Allgemeinen besitzt das Erhöhen der Anzahl an Schichten jedoch eine deutlich höhere Parametereffizienz, wodurch eine vergleichbare Performanz durch deutlich weniger Neuronen erreicht werden kann [vgl. 10, S. 326f.].

### Künstliche Neuronen und Aktivierungsfunktionen

Mathematisch betrachtet implementiert ein einzelnes künstliches Neuron eine Funktion, welche einen Eingabevektor auf eine einzelne numerische Ausgabe abbildet. Diese Funktion ist wie folgt definiert:

Hierbei stellt einen Vektor dar, der jeder Eingabe des Neurons ein Gewicht zuordnet. ist der sogenannte *Bias-Term* und (Sigma) eine für gewöhnlich nicht lineare *Aktivierungsfunktion*. Die Ausgabe eines Neurons entspricht also der Ausgabe einer Aktivierungsfunktion in Abhängigkeit von der gewichteten Summe der Eingaben und dem Bias-Term. Hierbei ist die Eingabe, wie in Abb. 10 veranschaulicht, in der Regel durch die Ausgaben der Neuronen der vorherigen Schicht bestimmt [vgl. 14, S. 6].

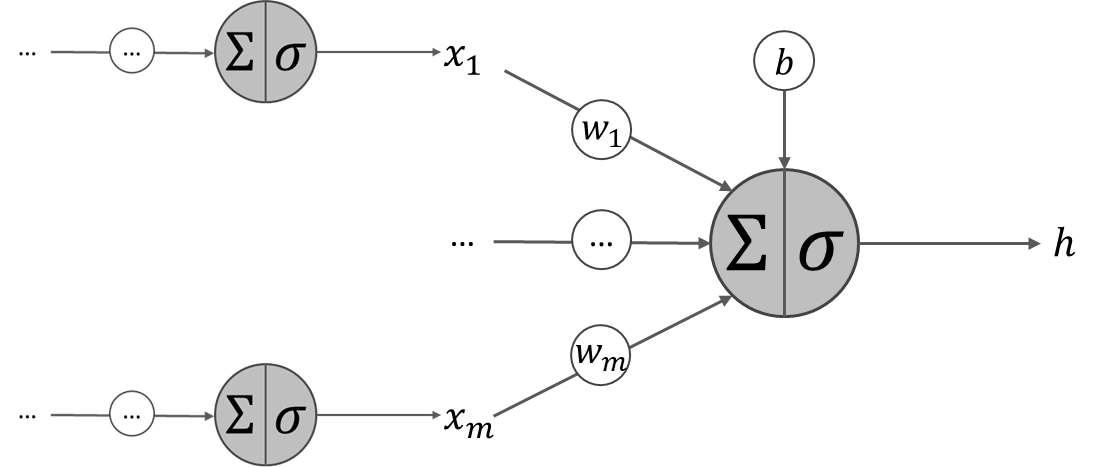


Abb. : Veranschaulichung eines künstlichen Neurons

Würde die Funktion aus der obigen Funktion entfallen, dann könnte ein künstliches Neuron lediglich lineare Transformationen abbilden. Da eine Verkettung von linearen Transformationen ebenfalls linear ist, könnten KNNs somit lediglich lineare Zusammenhänge abbilden und nicht zur Lösung komplexer Probleme eingesetzt werden. Deshalb werden bestimmte Aktivierungsfunktionen verwendet, um die Nichtlinearität der Neuronen zu garantieren. Dies ermöglicht KNNs theoretisch beliebig komplexe (stetige) Funktionen zu approximieren [vgl. 10, S. 293f.].

Die beliebtesten Aktivierungsfunktionen sind die *Sigmoid-Funktion* , die eine S-förmige Ausgabe im Intervall erzeugt, der *Tangens-Hyperbolicus* , der zu einer S-förmigen Ausgabe im Intervall führt, und die *Rectified-Linear-Unit-Funktion* , die sich sehr einfach berechnen lässt und eine Ausgabe im Intervall erzeugt [vgl. 10, S. 292f.]. Eine Darstellung dieser Funktionen ist in Abb. 11 zu sehen.

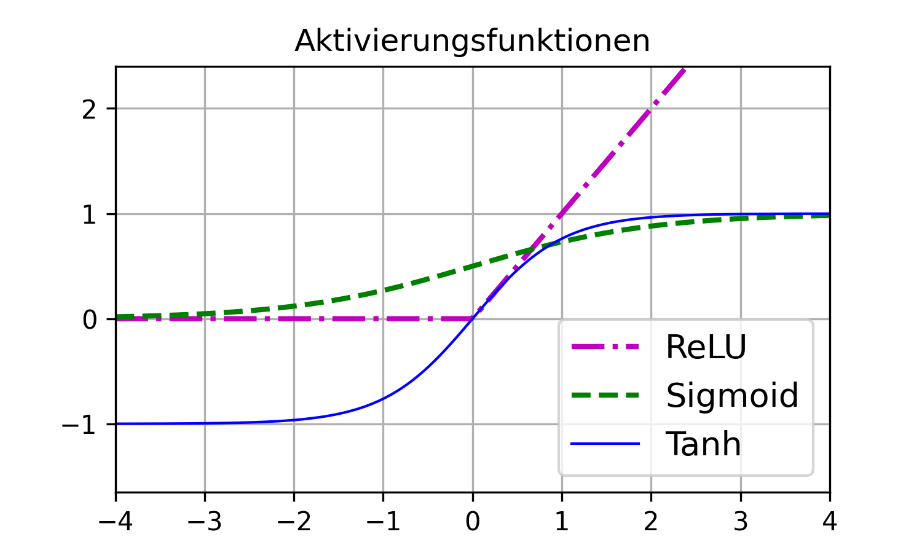


Abb. : Aktivierungsfunktionen: tanh(z),und

Für die Ausgabeschicht wird oftmals eine andere Aktivierungsfunktion als in den anderen Schichten des KNNs verwendet. Diese ist abhängig von der Aufgabenstellung und der angestrebten Ausgabe des Netzes. Bei mehrklassigen Klassifikationsproblemen nutzt man üblicherweise die *Soft-Max-Funktion* [vgl. 14, S. 6]. Diese ist für das Ausgabeneuron wie folgt definiert:

Hierbei entspricht der Anzahl der Ausgabeneuronen (bzw. der Klassen) und einem Vektor, der die gewichteten Summen aller Ausgabeneuronen enthält. Durch Anwendung der Soft-Max-Funktion summieren sich alle Ausgaben des KNNs zu 1 auf, wodurch man die Ausgabe des Neurons als Wahrscheinlichkeit interpretieren kann, dass der eingegebene Datenpunkt der zu zugeordneten Klasse angehört [vgl. 10, S. 149f.].

### Training neuronaler Netze

Beim Training der künstlichen Neuronen kommt es darauf an, die Gewichte und den Bias-Term so anzupassen, dass sie in Kombination mit der gewählten Aktivierungsfunktion zu der gewünschten Ausgabe führen.

### Rekurrente Neuronale Netze

## Bewertungsmaße für Klassifikatoren

Um die Qualität eines Klassifikationsverfahrens messbar zu machen und durch Training und Parameteroptimierung zu steigern, ist es wichtig Bewertungsmaße zu definieren, die die Vorhersageleistung eines Klassifikators widerspiegeln. In diesem Kapitel sollen solche Maße definiert und erläutert werden, wobei zum besseren Verständnis in allen Fällen schrittweise von einer binären Klassifikation auf ein n-klassiges Problem verallgemeinert wird.

### Konfusionsmatrizen

Ein verbreitetes Werkzeug, um die Vorhersageleistung eines Klassifikators zu visualisieren, ist die *Konfusionsmatrix*. Hierbei handelt es sich um eine Tabelle, bei welcher für eine Menge von Datenpunkten, die Schnittmengen aller vorhergesagten und tatsächlichen Klassen ausgezählt werden. Hierbei stehen typischerweise die Zeilen der Matrix für die tatsächlichen Klassen, wohingegen die Spalten die Vorhersagen repräsentieren [vgl. 10, S. 94]. Abb. X zeigt ein Beispiel für eine Konfusionsmatrix zur Auswertung eines binären Klassifikators auf Basis von 65 Datenpunkten.



Abb. : Beispiel für eine zweiklassige Konfusionsmatrix; eigene Abbildung

Konfusionsmatrizen können eine beliebige Anzahl von Klassen umfassen. Wichtig ist lediglich, dass deren Anordnung in den Zeilen und Spalten identisch ist, damit alle korrekt klassifizierten Datenpunkte auf der Hauptdiagonalen der Matrix widergespiegelt werden. Ein guter Klassifikator enthält entsprechend außerhalb dieser Hauptdiagonalen eine möglichst geringe Anzahl an Datenpunkten [vgl. 11, S. 2].

### Accuracy / Genauigkeit

Die *Accuracy* (dt. Genauigkeit) ist eine häufig genutzte Metrik bei Klassifikationsproblemen. Sie misst den Anteil der korrekt klassifizierten Datenpunkte an der Gesamtzahl aller betrachteten Datenpunkte. Für eine binäre Klassifikation ergibt sich somit die folgende Formel:

Bei N-klassigen Klassifikationsproblemen kann man die Accuracy berechnen, indem man zunächst die Accuracy-Werte für alle Klassen im Einzelnen und anschließend das arithmetische Mittel dieser Werte berechnet. Hiermit ergibt sich folgende Formel:

Die Accuracy ist ein sehr intuitives Maß. Unabhängig von der Anzahl der Klassen liegt sie (wie auch alle nachfolgenden Maße) immer im Intervall [0, 1], wobei ein Klassifikator umso besser ist, desto näher der Wert an 1 liegt. Allerdings sollte man vorsichtig sein, die Accuracy einzusetzen, falls der zugrundeliegende Datensatz unausgeglichen ist, da sie dazu tendiert große Klassifikationsfehler in unterrepräsentierten Klassen zu verbergen [vgl. 11, S. 3f.]. Für die Matrix in Abb. X berechnet sich eine Accuracy von 0,8.

### Precision / Relevanz

Die *Precision* (dt. Relevanz) misst die Genauigkeit der positiven Vorhersagen. Für eine binäre Klassifikation berechnet sie sich aus der folgenden Formel [vgl. 10, S. 95]:

Für N-klassige Klassifikationsprobleme haben sich verschiedene Herangehensweisen herausgebildet, die Precision zu berechnen. Die in dieser Arbeit verwendete, ist die sogenannte *Macro-Precision*, welche sich analog zur berechnet [vgl. 11, 6f.]:

Da die Precision durch die ausschließliche Beachtung der positiven Vorhersagen allein betrachtet sehr einseitig ist, geht Sie in der Regel mit dem *Recall* einher. Für die Matrix in Abb. X beträgt auch die Precision 0,8.

### Recall / Sensitivität

Der *Recall* (dt. Sensitivität), auch als Trefferquote bezeichnet, ist der Anteil der tatsächlich positiven Datenpunkte, die vom Klassifikator als solche vorhergesagt wurden. Er berechnet sich aus der folgenden Formel [vgl. 10, S. 95]:

Auch für den Recall haben sich im Hinblick auf N-klassige Klassifikationsprobleme verschiedene Herangehensweisen gebildet. In dieser Arbeit wird der *Macro-Recall* verwendet, der sich analog zur und berechnet [vgl. 11, 6f.]:

Auch der Recall ist für sich allein genommen sehr einseitig, deshalb wird er in der Regel gemeinsam mit der Precision ausgewertet. Für die Matrix in Abb. X beträgt der Recall lediglich rund 0,55.

### F1-Score

Da Precision und Recall oft gemeinsam ausgewertet werden kann es bequem sein diese zu einer einzigen Metrik zusammenzufassen, dem sogenannten *F1-Score*. Dieser berechnet sich aus dem harmonischen Mittelwert von Precision und Recall bzw. Macro-Precision und -Recall bei n-klassigen Klassifikationen:

Im Gegensatz zur Accuracy eignet sich der F1-Score auch zur Beurteilung von unausgeglichenen Datensätzen. Bei der Interpretation des Ergebnisses sollte lediglich beachtet werden, dass das harmonische Mittel niedrigeren Werten eine höhere Gewichtung gibt [vgl. 10, S. 96]. Für die Matrix in Abb. X beträgt der F1-Score ca. 0,65.

## Eingesetzte Technologien und Frameworks

### Valhalla Map-Matching API

Für der praktischen Umsetzung des Map-Matchings wird im Zuge dieser Arbeit die http-basierte Map-Matching-Service-API der open-source Routing-Engine *Valhalla* in der Version 3.3.0 eingesetzt. Diese Arbeit folgt damit der Empfehlung von Siavash Saki und Tobias Hagen, die in ihrem Paper [7] die Nutzung dieses Services als performante und gut skalierbare Alternative zu kommerziellen oder in der Nutzung limitierten Map-Matching-Services, wie bspw. dem von *Google Maps,* nahelegen.

Die API von Valhalla umfasst hierbei zwei verschiedene Map-Matching-Endpunkte, die ausgehend von der gleichen Eingabesequenz an GNSS-Punkten verschiedene Operationen ausführen. Der trace\_route Endpunkt gibt dabei im Wesentlichen die ans Straßennetz angepasste Sequenz mit einigen wenigen Zusatzinformationen (bspw. der Matching-Distanz) zurück, wohingegen der trace\_attributes Endpunkt dazu genutzt werden kann um detaillierte Informationen zu den zugeordneten Straßensegmenten (wie bspw. gültige Geschwindigkeitsbegrenzungen oder Art der Straße) zu erhalten. Für beide Endpunkte muss dabei explizit der *Costing*-Modusgesetzt werden. Die möglichen Modi umfassen u.a. ein Map-Matching für Fußgänger, Fahrradfahrer und motorisierte Straßenfahrzeuge. Darüber hinaus können zahlreiche zusätzliche Parameter für das Map-Matching gesetzt werden. Weitere Informationen hierzu sind der Dokumentation von Valhalla unter [16] zu entnehmen.

### Python

Der gesamte Code, der im Rahmen dieser Arbeit verfasst wurde, ist in der Programmiersprache *Python*, Version 3.10, geschrieben. Bei Python handelt es sich um eine universell einsetzbare, üblicherweise interpretierte, höhere Programmiersprache, die verschiedene Programmierparadigmen, wie bspw. die objektorientierte und funktionale Programmierung, unterstützt. Dabei ist Python dynamisch typisiert, wodurch sie sich auch als *Skriptsprache* eignet [vgl. 17]. Python besitzt eine ausführliche Dokumentation [18], welcher weitere Informationen zur Programmiersprache entnommen werden können.

Zu den Gründen, warum Python für die praktischen Umsetzungen dieser Arbeit gewählt wurde, zählen neben der einfachen Syntax und damit oft auch guten Lesbarkeit des resultierenden Codes insbesondere die zahlreichen nützlichen Standard- und Drittanbieter-Bibliotheken, die für diese Sprache existieren. Für das Verständnis des umgesetzten Codes werden dabei grundlegende Kenntnisse zu den folgenden zwei Bibliotheken vorausgesetzt: *pandas* und *NumPy*. Diese können bei Bedarf durch ein Studium der referenzierten Dokumentationen erlangt werden.

Bei pandas handelt es sich um eine Bibliothek zur Datenanalyse und -verarbeitung. Die Bibliothek bietet dabei zusätzliche Datenstrukturen, wie die sogenannten *Dataframes*, um auch sehr große Datensätze effizient zu manipulieren [vgl. 19]. Verwendete Funktionen und Objekte von pandas werden im Code über die *Namespace-Referenz* pd als solche gekennzeichnet.

NumPy ist eine Bibliothek für effiziente numerische Berechnungen. Auch NumPy kommt neben zahlreichen Funktionen mit eigenen Datenstrukturen, wie *Arrays*, die für performante Operationen auf höherdimensionalen Vektoren und Matrizen eingesetzt werden können. Diese Datenstrukturen bilden auch die Grundlage für andere Bibliotheken, wie bspw. Pandas [vgl. 20]. Im Code werden NumPy Funktionen und Objekte über die Namespace-Referenz np gekennzeichnet.

### scikit-learn

Scikit-learn gilt als umfangreichste open-source Bibliothek für das maschinell Lernen in Python. Die bereitgestellten Funktionen und Objekte umfassen dabei insbesondere die Bereiche Datentransformation und -vorverarbeitung, überwachte Lernverfahren, unüberwachte Lernverfahren sowie Modell-Evaluierung und -Auswahl. Dabei sind alle Implementierungen stark auf ihre Berechnungseffizienz optimiert. [vgl. 21, S. 353]. Besonders hervorzuheben ist außerdem das gute Schnittstellendesign, welches über all diese Bereiche hinweg einheitliche Konventionen festlegt. Dabei werden alle implementierten Objekte auf drei grundlegende Klassen generalisiert: *Estimatoren*, *Transformer* und *Prädiktoren*. Zu ersteren zählen alle Objekte, die interne Parameter auf Basis eines Datensatzes erlernen oder abschätzen können. Dies wird über die Methode fit() angestoßen, welche den Datensatz entgegennimmt. Alle Parameter können anschließend über öffentliche Attribute des Objekts abgerufen werden. Transformer sind Estimatoren, welche einen Datensatz zusätzlich transformieren können. Die Transformation wird mit der Methode transform()ausgeführt, die einen Datensatz entgegennimmt und den transformierten Datensatz zurückliefert. Die Transformation beruht dabei auf den gelernten Parameter der fit()-Methode. Alternativ kann auch die Methode fit\_transform() genutzt werden, die dem aufeinanderfolgenden Aufruf der beiden Methoden entspricht. Zu guter Letzt sind Prädiktoren Estimatoren, die in der Lage sind auf Basis gegebener Datenpunkte Vorhersagen zu treffen. Alle Prädiktoren besitzen hierfür die Methode predict(), welche einen Satz an Datenpunkten entgegennimmt und einen Satz entsprechender Vorhersagen zurückliefert. Außerdem besitzen alle Prädiktoren die Methode score(), über welche die Vorhersagequalität des Prädiktors über einen Testdatensatz und unter der Nutzung verschiedener Metriken ermittelt werden kann. Nötige Hyperparamter werden bei allen Klassen über den Objekt-Konstruktor gesetzt. [vgl. 10, S. 66].

Eine Abfolge von Transformern und ein abschließender beliebiger Estimator können außerdem als wiederverwendbare *Pipeline* definiert und abgespeichert werden. Wobei sich diese anschließend wie ein einziger Estimator verhält. Damit können bspw. alle benötigten Vorverarbeitungsschritte und eine anschließende Klassifikation zusammengefasst werden. Bei Bedarf können weitere Informationen zu Funktionen und Objekten von scikit-learn über die Dokumentation unter [22] bezogen werden.

### Keras und TensorFlow

*Keras* ist eine open-source Deep-Learning-Bibliothek, die es erlaubt alle möglichen Arten von neuronalen Netzen zu entwerfen, zu trainieren, auszuwerten und auszuführen. Seit Veröffentlichung der Referenzimplementierung im Jahr 2015 zählt Keras aufgrund ihrer Benutzerfreundlichkeit und Flexibilität zu den verbreitetsten Bibliotheken für das Deep-Learning mit Python. Einsteigern kommt hierbei zugute, dass die API von Keras an vielen Stellen durch die scikit-learn-API beeinflusst ist. Für die aufwendigen Berechnungen, welche insbesondere mit dem Training tiefer neuronaler Netze einhergehen, ist die Referenzimplementierung von Keras auf ein zusätzliches Rechen-Backend angewiesen. Ein solches wird beispielsweise durch *TensorFlow* bereitgestellt [vgl. 10, S. 297f.].

Bei TensorFlow handelt es sich um eine Bibliothek für numerische Berechnungen und datenstromgetriebene Programmierung, die sich durch ihre hohe Effizienz und Anpassbarkeit auszeichnet. TensorFlow wurde durch das Google-Brain-Team entwickelt und ist inzwischen Basis für eine Vielzahl an Google-Services mit umfangreichen Rechenanforderungen [vgl. 10, S. 379]. Seit Version 2 bringt TensorFlow seine eigene Keras-Implementierung (tf.keras) mit sich, welche die Referenzimplementierung um einige nützliche Zusatzfeatures, unter anderem zur Datenvorverarbeitung, erweitert. Außerdem bietet TensorFlow unter bestimmten Voraussetzungen auch die Möglichkeit Berechnungen auf einer oder mehreren GPUs durchzuführen, wodurch insbesondere das Training von Neuronalen Netzen beschleunigt werden kann [vgl. 10, S. 277f.].

Für einen tieferen Überblick über TensorFlow sei an dieser Stelle auf die zugehörige Webseite unter [23] verwiesen. In Bezug auf Keras können weitere Informationen zu den verwendeten Funktionen und Objekten bei Bedarf über die API-Dokumentation unter [24] bezogen werden.

# Verwandte Arbeiten

## Vorangegangene Abschlussarbeiten

Dieser Arbeit gehen drei Abschlussarbeiten voraus, welche sich ebenfalls mit dem Themenfeld *Klassifizierung von Verkehrsteilnehmern basierend auf Positionsdaten* im Zuge der erweiterten Umfeldwahrnehmung autonomer Fahrzeuge befasst haben. Nachfolgend werden die wichtigsten Erkenntnisse aber auch die Grenzen dieser Arbeiten aufgezeigt.

Die Bachelorarbeit von Recep Furgan Torlak [25] aus dem Jahr 2022 bildet im Wesentlichen den Grundstein für die späteren Arbeiten. Die Arbeit setzt sich mit verschiedenen Möglichkeiten der Datengewinnung auseinander, wobei der Fokus auf der Abfrage von Positionsdaten aus der Verkehrssimulation *CARLA* liegt, für die im Zuge der Arbeit auch eine Python-API umgesetzt wurde. Auch Möglichkeiten reale Daten zu gewinnen und simulierte Daten realistischer zu machen, werden in der Arbeit aufgeführt. Außerdem beschäftigte sich Torlak mit der Berechnung von Bewegungsinformationen aus den ermittelten Positionsdaten, welche später zur Klassifikation genutzt werden sollen. Die Klassifikation selbst ist jedoch kein Teil der Arbeit.

Maximilian Sohl knüpft in seiner Bachelorarbeit [26], ebenfalls aus dem Jahr 2022, direkt an die Arbeit von Torlak an. Er nutzt hierbei die aus der Verkehrssimulation CARLA gewonnenen Daten, um die Leistung von drei verschiedene Klassifikationsverfahren zu evaluieren. Bei diesen handelt es sich um die Verfahren *Support-Vector-Machine*, *Decision-Tree* und *K-Nearest-Neighbours.* Hierbei sollten die Klassifikatoren auf Basis verschieden stark verrauschter Positionsdaten bis zu fünf Fahrzeugklassen zu unterscheiden lernen, allerdings stellte sich im Verlauf der Arbeit heraus, dass auf Basis der Simulation höchstens drei Klassen (Fußgänger, Fahrradfahrer und motorisiertes Fahrzeug) unterschieden werden können. Vor der Klassifikation wurden, die über ein festes Zeitintervall von einer Minute gesammelten Positionsdaten in lange Eingabevektoren weiterverarbeitet, die die Bewegung der Verkehrsteilnehmer innerhalb dieses Intervalls repräsentieren. Eine Merkmalsreduktion fand nicht statt. Bei der anschließenden Klassifikation wurden die besten Ergebnisse durch die SVM erzielt, welche einen Accuracy-Wert von 93,9% auf den unverrauschten Daten erreichen konnte. Ein Verrauschen der Daten, um diese vermeintlich realistischer zu machen, führte in allen Fällen zu einem starken Abfall der Qualität der Klassifikation.

Auch Dietmar Fischer fokussiert sich in seiner Masterarbeit [27] von 2023 primär auf die Nutzung von Daten, die ohne Verfälschung der Verkehrssimulation CARLA entnommen wurden, um ein System zu entwickeln, welches durch den Einsatz von neuronalen Netzen dazu in der Lage ist Verkehrsteilnehmer in drei Typen (Fußgänger, Fahrradfahrer und motorisiertes Fahrzeug) und zusätzlich entsprechend ihrer Relevanz für den *Hero* (das eigene Fahrzeug) zu klassifizieren. Dabei untersucht Fischer im Wesentlichen die Performanz zweier Arten von neuronalen Netzen: Zum einen von gewöhnlichen Feed-Forward-Netzen (FFN) und zum anderen von *Deep-Q-Learning-Netzen* (DQLN). Letztere zählen anders als die übrigen erprobten Ansätze nicht mehr zu den überwachten, sondern zu den bestärkenden Lernverfahren. Im Zuge der Datenvorverarbeitung setzte Fischer anders als Sohl auf eine Reduktion der Eingabedimension auf 16 deskriptive Statistiken, welche die Bewegungsmerkmale des Verkehrsteilnehmers über ein festes Intervall von einer Minute zusammenfassen. Die Ergebnisse der Arbeit zeigen, dass der DQLN-Ansatz bei der Typklassifikation einen Accuracy-Wert von 90% auf den Simulationsdaten erreichen konnte. Das FFN erreichte 93%. Darüber hinaus erprobte Fischer die Modelle mit einem kleinen Datensatz an realen Positionssequenzen, wobei zumindest das beste FFN einen guten Accuracy-Wert von etwa 90% erzielen konnte. Der unternommene Versuch auch die Relevanz von anderen Fahrzeugen aus Sicht des Heros in Risikokategorien zu klassifizieren, hat zu keinen nennenswerten Ergebnissen geführt.

Anders als bei Fischer beschränkt sich die vorliegende Arbeit erneut lediglich auf die Klassifikation von Verkehrsteilnehmern in verschiedene Typen. Im Gegensatz zu allen vorangegangenen Arbeiten legt diese jedoch den Fokus nicht mehr auf Simulationsdaten, die innerhalb von CARLA gewonnen wurden, sondern komplett auf reale Daten, welche durch Teilnehmer des Straßenverkehrs erzeugt wurden und potenziell weiterer Vorverarbeitungen bedürfen. Des Weiteren soll erstmalig eine Unterscheidung verschiedener motorisierter Fahrzeuge untersucht werden, wobei auch auf bisher nicht erprobte maschinelle Lernverfahren zurückgegriffen werden soll. + keine festen Zeitintervalle mehr, weil Nonsense

## Vehicle Classification from Low-Frequency GPS Data with Recurrent Neural Networks

Die Arbeit *Vehicle Classification from Low-Frequency GPS Data with Recurrent Neural Networks* von Matteo Simoncini et. al. [14], die im Jahr 2018 veröffentlicht wurde, untersucht die Klassifikation von motorisierten Fahrzeugen in drei Kategorien: leichte, mittelschwere, und schwere Fahrzeuge. Basis hierfür bilden niederfrequente GNSS-Sequenzen mit einem variierendem Abtastintervall von durchschnittlich 90 Sekunden.

Der zugrundeliegende Datensatz ist äußerst umfangreich. Er umfasst in etwa eine Million Sequenzen, die durch insgesamt 55 Millionen GNSS-Punkte eine Strecke von etwa 56 Millionen gefahrenen Kilometern abbilden. Die Sequenzen wurden so vorverarbeitet, dass für alle paarweisen GNSS-Punkte die *gefahrene Strecke*, die *direkte Krähenflugdistanz*, die *verstrichene Zeit*, die *Geschwindigkeit* und *Intervallgeschwindigkeit*, die *Beschleunigung* und *Intervallbeschleunigung* sowie der *Straßentyp* vorlag. Globale Merkmale oder deskriptive Statistiken wurden nicht berechnet. Die für die Klassifikation verwendeten Sequenzen umfassten mindestens 20 Zeitschritte, was einem Zeitfenster von etwa 30 Minuten entspricht.

Für die Klassifikation wurden verschiedene Rekurrente Neuronale Netze mit LSTM-Schichten entworfen. Dabei kamen auch reguläre Feed-Forward-Schichten zum Einsatz, die vor und nach den LSTM-Schichten platziert wurden, was den Autoren zufolge die Vorhersageleistung der Netze deutlich verstärkte. Das beste Modell erreichte auf den Testdatensatz eine Trefferquote (Recall) von 85% für leichte und sogar 93% für schwere Fahrzeuge. Allerdings hat das Modell Probleme in der Unterscheidung von mittelschweren und leichten Fahrzeugen, wodurch es für mittelschwere Fahrzeuge lediglich eine Trefferquote von 48% erzielen konnte.

Nichtsdestotrotz zeigt die Arbeit, dass der Einsatz von RNNs durch ihre Fähigkeit sequenzielle Daten zu verarbeiten, ein vielversprechendes Verfahren für die Klassifikation von Fahrzeugen auf Basis von GNSS-Sequenzen ist. Die Autoren heben dabei in ihrem Fazit hervor, dass die Nutzung von höherfrequenten GNSS-Sequenzen bei einem ähnlichen Ansatz das Potenzial besitzt, die aufgezeigten Schwächen zu überwinden und Fahrzeuge zuverlässiger und auch schneller zu klassifizieren.

## Vehicle Classification using GPS Data

Im Zuge ihrer Arbeit *Vehicle Classification using GPS Data* setzten sich Zhanbo Sun und Xuegang Ban bereits im Jahr 2013 mit der Klassifikation von Fahrzeugen auf Basis von GNSS-Sequenzen auseinander. Untersucht wird hierbei lediglich ein binäres Klassifikationsproblem: Die Unterscheidung zwischen PKWs und LKWs, basierend auf einem Datensatz der 52 PKW- und 84 LKW-Sequenzen mit Längen von 15-20min umfasst. Alle genutzten Sequenzen bilden lediglich Fahrten auf Zubringerstraßen ab, die mit einer Abtastrate von etwa drei Sekunden aufgenommen wurden.

Aus den Sequenzen wurden im Zuge der Arbeit verschiedene globale Merkmale mit Geschwindigkeits- sowie Beschleunigungs- und Verzögerungsbezug extrahiert. Anschließend wurden verschiedene Modelle zur Merkmalsauswahl (engl. *feature selction*) und Klassifikation entwickelt und deren Ergebnisse veranschaulicht und ausgewertet. Die Klassifikationsmodelle basierten hierbei auf Support-Vector-Machines mit quadratischen Kernel-Funktionen.

Die Autoren heben hervor, dass Merkmale, die das Beschleunigungs- und Verzögerungsverhalten der Fahrzeuge abbilden, sich besser zur Klassifikation verschiedener motorisierter Fahrzeuge eignen als geschwindigkeitsbasierte Merkmale. Am effektivsten waren hierbei der Anteil von Beschleunigungs- und Verzögerungswerten über und die Standardverteilungen der Beschleunigung und Verzögerung. Unter der Nutzung dieser vier Merkmale konnte durch das beste SVM-Modell eine Fehlklassifizierungsrate von nur 4,2% (bzw. eine Accuracy von 95,8%) auf den Testdaten erreicht werden.

# Datengrundlage

## Gewinnung der Ausgangsdaten

Wie wurden die Daten gewonnen? -> MotionTrace

Unter welchen Umständen und unter welchen Einschränkungen wurden die Daten erzeugt?

## Beschreibung der Ausgangsdaten

Wie umfangreich ist die Datenbasis?

Wie sehen die Rohdaten aus?

Sind die Daten ausgeglichen?

Probleme in den Daten, welche beachtet werden müssen?

## Weiterverarbeitung zu Bewegungsdaten

# Anforderungen und Gesamtkonzept der Klassifikation

Teil 1: Wie genau sieht das zugrundeliegende Problem aus und welche konkreten Aufgaben/Anforderungen ergeben sich daraus?

Teil 2: Wie könnte man dieser Aufgaben Herr werden? Welche Ansätze und Ideen werden gewählt?

# Umsetzung des Map-Matchings

## Einrichtung der Valhalla-Engine

## Umsetzung und Evaluierung des Road-Snappings

## Vorklassifikation zur Bestimmung des korrekten Matching-Modus

### Erzeugung des Trainingsdatensatzes

### Auswahl des Klassifikators

### Evaluierung

# Umsetzung der Klassifikation

### Erzeugung der Trainingsdatensätze (inkl. Vorverarbeitungen)

### Training und Optimierung der Modelle

### Vorstellung der besten Klassifikatoren

Weitere Unterkapitel ggf. abhängig von schlussendlich verwendeten Verfahren.

# Evaluierung und Diskussion

Weitere Unterkapitel abhängig von schlussendlich verwendeten Verfahren. Dieses Kapitel wird aber auf jeden Fall einen Vergleich zwischen gematchten und ungematchten Daten bei der finalen Klassifikation enthalten.

# Zusammenfassung und Ausblick

…

# Quellenverzeichnis

[1] Gartner. „Gartner Says 5G Networks Have a Paramount Role in Autonomous Vehicle Connectivity: CSPs Need to Ensure Participation in Safety Design of Autonomous Vehicles.” https://​www.gartner.com​/​en/​newsroom/​press-​releases/​2018-​06-​21-​gartner-​says-​5g-​networks-​have-​a-​paramount-​role-​in-​autonomous-​vehicle-​connectivity (Zugriff am: 20. Juni 2023).

[2] IZK. „5GKC: 5G basiertes Testfeld für das automatisierte Fahren.” https://​5gkc.net​/​ (Zugriff am: 20. Juni 2023).

[3] H. Winner, S. Hakuli, F. Lotz und C. Singer, Hg. *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort,* 3. Aufl. (ATZ/MTZ-Fachbuch). Wiesbaden: Springer Vieweg, 2015. [Online]. Verfügbar unter: https://​ebookcentral.proquest.com​/​lib/​kxp/​detail.action​?​docID=​1997888

[4] E. D. Kaplan und C. J. Hegarty, *Understanding GPS/GNSS: Principles and applications* (GNSS technology and applications series). Boston, London: Artech House, 2017.

[5] Wikipedia. „Globales Navigationssatellitensystem.” https://​de.wikipedia.org​/​w/​index.php​?​title=​Globales\_Navigationssatellitensystem&​oldid=​233312836 (Zugriff am: 22. Juni 2023).

[6] magicmaps. „Wie funktioniert Satellitennavigation?” https://​www.magicmaps.de​/​gnss-​wissen/​wie-​funktioniert-​gps/​?​L=​0 (Zugriff am: 22. Juni 2023).

[7] S. Saki und T. Hagen, „A Practical Guide to an Open-Source Map-Matching Approach for Big GPS Data,“ *SN COMPUT. SCI.*, Jg. 3, Nr. 5, 2022, doi: 10.1007/s42979-022-01340-5.

[8] P. Chao, Y. Xu, W. Hua und X. Zhou, „A Survey on Map-Matching Algorithms,“ Okt. 2019. [Online]. Verfügbar unter: https://​arxiv.org​/​pdf/​1910.13065

[9] Wikipedia. „Arthur Samuel (computer scientist).” https://​en.wikipedia.org​/​w/​index.php​?​title=​Arthur\_Samuel\_(computer\_scientist)&​oldid=​1149301740 (Zugriff am: 26. Juni 2023).

[10] A. Géron, *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow: Konzepte, Tools und Techniken für intelligente Systeme,* 2. Aufl. Heidelberg: O'Reilly, 2020.

[11] M. Grandini, E. Bagli und G. Visani, „Metrics for Multi-Class Classification: an Overview,“ Aug. 2020. [Online]. Verfügbar unter: https://​arxiv.org​/​pdf/​2008.05756

[12] A. Mammone, M. Turchi und N. Cristianini, „Support vector machines,“ *WIREs Comp Stat*, Jg. 1, Nr. 3, S. 283–289, 2009. doi: 10.1002/wics.49. [Online]. Verfügbar unter: https://​wires.onlinelibrary.wiley.com​/​doi/​pdfdirect/​10.1002/​wics.49​?​download=​true

[13] Ali, Jehad, R. Khan, N. Ahmad und I. Maqsood, „Random Forests and Decision Trees,“ *IJCSI International Journal of Computer Science Issues*, Jg. 9, Nr. 3, 2012, Art. Nr. 5. [Online]. Verfügbar unter: https://​www.uetpeshawar.edu.pk​/​TRP-​G/​Dr.Nasir-​Ahmad-​TRP/​Journals/​2012/​Random%20Forests%20and%20Decision%20Trees.pdf

[14] Matteo Simoncini, Leonardo Taccari, Francesco Sambo, Luca Bravi, Samuele Salti und Alessandro Lori, „Vehicle Classification from Low-Frequency GPS Data with Recurrent Neural Networks,“ *Transportation Research Part C: Emerging Technologies*, Jg. 91, 2018, doi: 10.1016/j.trc.2018.03.024.

[15] Michael Nielsen, „Neural Networks and Deep Learning,“ 2019. [Online]. Verfügbar unter: https://​static.latexstudio.net​/​article/​2018/​0912/​neuralnetworksanddeeplearning.pdf

[16] „Valhalla Docs: Map Matching API.” https://​valhalla.github.io​/​valhalla/​api/​map-​matching/​api-​reference/​#trace-route-action (Zugriff am: 24. Juni 2023).

[17] Python Foundation. „Python documentation: General Python FAQ.” https://​docs.python.org​/​3/​faq/​general.html​#what-is-python-good-for (Zugriff am: 24. Juni 2023).

[18] Python Foundation. „Python 3.10.11 documentation.” https://​docs.python.org​/​3.10/​ (Zugriff am: 24. Juni 2023).

[19] pandas. „pandas documentation.” https://​pandas.pydata.org​/​docs/​ (Zugriff am: 24. Juni 2023).

[20] NumPy. „NumPy documentation.” https://​numpy.org​/​ (Zugriff am: 24. Juni 2023).

[21] J. Hao und T. K. Ho, „Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language,“ *Journal of Educational and Behavioral Statistics*, Jg. 44, Nr. 3, S. 348–361, 2019, doi: 10.3102/1076998619832248.

[22] scikit-learn. „machine learning in Python — scikit-learn 1.2.2 documentation.” https://​scikit-learn.org​/​stable/​ (Zugriff am: 25. Juni 2023).

[23] „TensorFlow,” [Online]. Verfügbar unter: https://​www.tensorflow.org​/​

[24] Keras, „Keras API reference,“ 2023. [Online]. Verfügbar unter: https://​keras.io​/​api/​

[25] R. F. Torlak, „Detektion der Bewegung von Verkehrsteilnehmern aus Positionsdaten,“ Bachelorarbeit, Hochschule für angewandte Wissenschaften Coburg, 2022.

[26] M. Sohl, „Klassifizierung der Bewegungsmuster von Mobilfunkteilnehmern zur erweiterten Umfeldwahrnehmung autonomer Fahrzeuge,“ Bachelorarbeit, Hochschule für angewandte Wissenschaften Coburg, 2022.

[27] D. Fischer, „Verwendung von Positionsdaten zur automatisierten Klassifizierung von Verkehrsteilnehmern mittels maschinellen Lernverfahren,“ Masterarbeit, Hochschule für angewandte Wissenschaften Coburg, 2023.

1. Test