



Institut für Informatik
Lehrstuhl für Organic Computing
Prof. Dr. rer. nat. Jörg Hähner

Masterarbeit

**Verbesserung der automatischen
Fahrzeugdiagnose mit Maschinellen
Lernverfahren**

Michael Krug

Erstprüfer: Prof. Dr. rer. nat. Jörg Hähner

Zweitprüfer: Prof. Dr. Bernhard Bauer

Betreuer: Anthony Stein, M.Sc.

Matrikelnummer: 1465630

Studiengang: Informatik und Informationswirtschaft (Master)

Eingereicht am: 18.06.2018

Abstract

Die vorliegende Masterarbeit beschäftigt sich mit dem Big Data Diagnosis (BDD) System von Daimler, das weltweit in den meisten Mercedes-Benz Werkstätten zur automatischen Fahrzeugdiagnose eingesetzt wird. Ziel ist es, die Leistung von BDD mit alternativen Maschinellen Lernverfahren zu steigern. Das aktuelle Produktivsystem von BDD basiert auf der One-versus-Rest Methode und XGBoost, einer beliebten Implementierung von Gradient Boosting Machines. Den Forschungsschwerpunkt dieser Arbeit bildet die Frage, ob die Reparaturprognosen von BDD durch den Einsatz von künstlichen neuronalen Netzen verbessert werden können. Hierfür wurde eine Anwendung in Python mit scikit-learn, Keras und TensorFlow entwickelt, die sich auf die Datenaufbereitung und das Training der Modelle konzentriert. Folglich kommt es bei der Leistung von BDD zu einem Vergleich zwischen Gradient Boosting Machines und künstlichen neuronalen Netzen, den momentan wohl erfolgreichsten Ansätzen im Maschinellen Lernen. Zur Optimierung der Vorhersagen wurden verschiedene Netzwerkarchitekturen sowie Hyperparameter-Kombinationen getestet und systematisch in Bezug auf die Daten der Mercedes-Benz A-Klasse evaluiert. Für die Modellselektion wurde eine 3-fache Kreuzvalidierung im Zusammenspiel mit einer teilautomatisierten Grid Search angewendet. Die Ergebnisse zeigen, dass durch den Einsatz von künstlichen neuronalen Netzen eine deutliche Leistungssteigerung von BDD möglich ist. Die besten Resultate lieferten die Netzwerkarchitekturen, bei denen die drei Oberklassen von Reparaturcodes getrennt trainiert wurden. Darüber hinaus ist es vielversprechend, die Daten von mehreren Fahrzeugbaureihen zusammen zu trainieren. Letztlich zeigt diese Arbeit, dass der Einsatz von künstlichen neuronalen Netzen eine echte Alternative zum Produktivsystem von BDD darstellen könnte.

Danksagung

An dieser Stelle möchte ich mich bei allen Mitarbeitern der Daimler TSS GmbH und des Lehrstuhls von Organic Computing an der Universität Augsburg bedanken, die mich bei meiner Masterarbeit unterstützt haben.

Mein besonderer Dank gilt Dr. Valentin Zacharias und Anthony Stein für die großartige Betreuung und Hilfsbereitschaft bei der Erstellung dieser Arbeit. Bei unseren Treffen haben Sie mich stets mit neuen Anregungen inspiriert und motiviert.

Nicht zuletzt bedanke ich mich auch bei meiner Familie, insbesondere bei meinen Eltern und meiner Schwester, die mir während des gesamten Studiums zur Seite standen und mich immer gut beraten haben.

Inhaltsverzeichnis

Abstract	i
Danksagung	iii
1 Einleitung	1
1.1 Motivation	1
1.2 Big Data Diagnosis bei Daimler	5
1.3 Zielsetzung	7
1.4 Verwandte Arbeiten	8
1.5 Aufbau der Arbeit	9
2 Einordnung der Problemstellung ins Maschinelle Lernen	11
2.1 Trainingserfahrung	11
2.2 Aufgabentyp	13
2.3 Leistungsmaß	14
2.4 Das Bias-Varianz Dilemma	16
2.5 Charakterisierung der BDD-Daten	18
3 Einführung in die Multi-Label Klassifikation	27
3.1 Metriken	27
3.1.1 Beispielorientiert	28
3.1.2 Labelorientiert	30
3.1.3 Ranking	32
3.2 Lösungsvarianten	33
3.2.1 Problemtransformation	33
3.2.2 Angepasste Algorithmen	37

3.2.3	Implizit unterstützende Algorithmen	38
4	Grundlagen künstlicher neuronaler Netze	41
4.1	Multilayer-Perzeptron	42
4.2	Training	44
4.2.1	Backpropagation	45
4.2.2	Gradientenverfahren	47
4.3	Regularisierung	49
4.3.1	Early Stopping und k-fache Kreuzvalidierung	50
4.3.2	Dropout	52
4.4	Weitere Optimierungsstrategien	53
4.4.1	Initialisierung der Gewichte	53
4.4.2	Varianten von ReLUs	54
4.4.3	Batch-Normalisierung	55
5	Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline	57
5.1	Verwendete Frameworks	57
5.2	Grundlegendes Design der ANNs	59
5.3	Auswahl vorhersagbarer Reparaturcodes	60
5.4	Netzwerkarchitekturen	62
5.4.1	Reduktion der Anzahl an Modellen	62
5.4.2	Transfer Lernen bei den SSL7-Codes	66
5.4.3	Beschreibungen der Fehlercodes als Features	69
5.5	Hyperparameter Optimierung	71
5.6	Auswahl der besten Modelle	74
5.7	Präsentation und Diskussion der finalen Ergebnisse	78
6	Zusammenfassung und Ausblick	83
	Literaturverzeichnis	I
	Abbildungsverzeichnis	VII
	Tabellenverzeichnis	IX

A Eidesstattliche Erklärung

XI

1 Einleitung

Zu Beginn dieses Kapitels wird die Motivation für diese Arbeit im Vordergrund stehen. Anschließend wird der zentrale Anwendungsfall bei Daimler vorgestellt. Daraufhin folgt die Zielsetzung sowie ein Abschnitt zu verwandten Arbeiten. Schließlich wird noch ein Überblick über die Inhalte der restlichen Kapitel dieser Arbeit gegeben.

1.1 Motivation

Das Sammeln und Klassifizieren von Informationen ist für Unternehmen im Zeitalter des digitalen Wandels von zentraler Bedeutung. Da die riesigen Datenmengen oftmals nicht mehr manuell ausgewertet werden können, erfährt die Entwicklung von automatisierten Verfahren in den verschiedensten Branchen einen großen Aufschwung. Nicht wegzudenken sind dabei die Fortschritte auf dem Gebiet des *Maschinellen Lernens*, einem Teilgebiet der *Künstlichen Intelligenz*.

Künstliche Intelligenz ist ein Thema, das die Menschen schon seit langer Zeit beschäftigt. Im Jahr 1950 untersuchte Turing in seiner wegweisenden Arbeit die Frage, ob Maschinen in der Lage sind zu denken und stellte den bekannten *Turing-Test* vor [Tur09]. Seitdem gab es verschiedene Ansätze, intellektuelle Aufgaben zu automatisieren, die normalerweise von Menschen ausgeführt werden [Cho18].

Der *wissensbasierte Ansatz* war lange das dominierende Paradigma in der Künstlichen Intelligenz. Dabei wird eine Menge von explizit programmierten

1 Einleitung

Regeln genutzt, um klar definierte Problemstellungen mit logischen Schlüssen zu lösen [GBCB16]. Diese regelbasierten Systeme erreichten ihren Höhepunkt in den 1980er Jahren, als immer mehr *Expertensysteme* für die Industrie entwickelt wurden [Cho18].

Für Problemstellungen, die logisch unscharf und formal nur schwer beschreibbar sind, war der wissensbasierte Ansatz allerdings nicht geeignet. Aufgaben wie die Bild- oder Spracherkennung, die von Menschen intuitiv und ohne großen Aufwand gelöst werden können, stellten folglich die größte Herausforderung in der Künstlichen Intelligenz dar. Hier sorgte das Maschinelle Lernen für einen echten Durchbruch.[GBCB16]

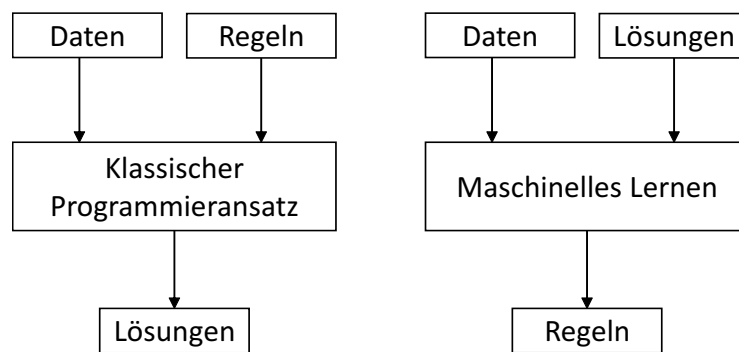


ABBILDUNG 1.1: Maschinelles Lernen im Vergleich zur klassischen Programmierung [Cho18]

Maschinelles Lernen basiert auf der Erkennung von Mustern und ermöglicht Computern das eigenständige Lernen anhand von Erfahrungen. Im Gegensatz zu den regelbasierten Systemen gibt es hier eine integrierte Lernkomponente, die den Entwicklern einen Großteil der Arbeit abnimmt, da die Fähigkeiten eines Systems nicht bis ins letzte Detail programmiert werden müssen [Cho18].

Wie in Abbildung 1.1 zu erkennen ist, sind Maschinelle Lernsysteme in der Lage, die für eine Problemstellung relevanten Regeln automatisch zu finden. Dabei wird grundsätzlich eine Funktion approximiert, die die unabhängigen Variablen (Input Daten) möglichst genau auf die abhängigen Variablen (Output Daten) abbildet [Bis06]. Die erlernten Regeln können dann z. B. für die Klassifikation von unbekannten Daten verwendet werden.

Eine prägnante Definition des Maschinellen Lernens wird von Mitchell gegeben: «A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with Experience E.»[Mit97].

Für ein Maschinelles Lernmodell braucht man demnach eine definierte Aufgabe T, eine Art von Trainingserfahrung E und ein Leistungsmaß P, das zeigt, ob der Algorithmus richtig funktioniert. Beim Computerprogramm *AlphaGo*, das im Jahr 2016 den Weltmeister im Brettspiel Go mit Maschinellen Lernverfahren besiegt hat, war T die Fähigkeit Go zu spielen, P die geschätzte Elo-Wertung (allgemeines Maß für die Spielstärke von Go Spielern) des Systems und E erst ausgewählte Züge von Experten und später Spiele gegen AlphaGo selbst [SSS⁺17]. Wie in der Definition von Mitchell beschrieben, verbesserte sich die Elo-Wertung von AlphaGo mit der steigenden Anzahl an Trainingserfahrungen und erreichte schließlich sogar ein übermenschliches Niveau [SSS⁺17].

Maschinelles Lernen hat den Weg für viele neue Anwendungen freigemacht, die von Menschen teilweise gar nicht explizit programmiert werden können. Mittlerweile sind die Verfahren weit verbreitet und fester Bestand unseres Alltags. Sie werden z. B. genutzt, um die Resultate von Suchmaschinen zu optimieren, Betrugsversuche bei Kreditkarten aufzudecken, Videos und Songs zu empfehlen und Autos das autonome Fahren zu ermöglichen [Gér17]. Ein Ansatz, der in diesem Zusammenhang häufig verwendet wird, ist *Deep Learning*. Deep Learning ist ein Teilgebiet des Maschinellen Lernens (siehe Abbildung 1.2), bei dem das Lernen in mehreren aufeinander aufbauenden Schichten abläuft und das als grundlegendes Modell praktisch immer *künstliche neuronale Netze* (Artificial Neural Networks, kurz ANNs) verwendet [GBCB16].

Beim Deep Learning werden oft ANNs mit hunderten von Schichten (Layers) eingesetzt, die komplexe Konzepte aus einer Reihe von einfacheren Repräsentationen lernen [Cho18]. Den Prozess des Lernens kann man sich dabei als eine automatische Suche nach immer besser passenden Repräsentationen vorstellen, die die Input Daten möglichst genau auf die entsprechenden Output Daten abbilden. Bei der Bilderkennung lernt ein System auf Basis der Pixel

1 Einleitung

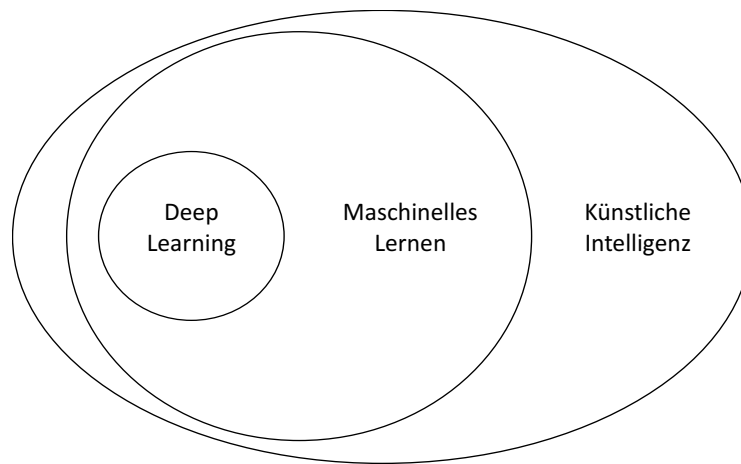


ABBILDUNG 1.2: Einordnung von Deep Learning, Maschinellern Lernen und Künstlicher Intelligenz [Cho18]

z.B. erst Repräsentationen für Kanten, Ecken und Konturen, bevor Teile eines Objekts wie Reifen oder Außenspiegel und später ganze Objekte wie ein Auto erkannt werden können [GBCB16].

Beim weltweit bekanntesten Wettbewerb der Objekterkennung, der *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), hat Deep Learning im Jahr 2012 für großes Aufsehen gesorgt, als das Sieger-Team den *Top-5 Fehler* von 26,1 % auf 15,3 % verbesserte [KSH12]. Seitdem wurde der jährliche Wettbewerb immer von einem Deep Learning Ansatz gewonnen. Im Jahr 2017 lag der Top-5-Fehler des besten Modells bei ILSVRC sogar nur noch bei rund 2.25 % [HSS17].

Die Erfolge von Maschinellen Lernverfahren, Deep Learning und speziell von ANNs sind eindrucksvoll und vielseitig, weshalb sie von Unternehmen auch verstärkt für industrielle Prozesse eingesetzt werden. Im Folgenden wird ein konkreter Anwendungsfall bei Daimler vorgestellt, der das zentrale Thema dieser Arbeit darstellt.

1.2 Big Data Diagnosis bei Daimler

Die Automobilindustrie steht aktuell vor großen Veränderungen wie der digitalen Vernetzung und dem autonomen Fahren. Diese Entwicklungen haben zur Folge, dass der Anteil an Software- und Elektrik-/Elektronik-Systemen in den Fahrzeugen noch größer wird. Mit dem steigenden Funktionsumfang nehmen auch die Wechselwirkungen zwischen den Komponenten und die Anzahl an potentiellen Problemen zu, was eine erhöhte Komplexität bei der Fahrzeugdiagnose mit sich bringt.

Für Automobilkonzerne ist es folglich eine große Herausforderung, die Zuverlässigkeit und Verfügbarkeit der Fahrzeuge zu verbessern und gleichzeitig die Kosten für Service und Instandhaltung möglichst gering zu halten. Vor diesem Hintergrund wird bei Daimler das *Big Data Diagnosis* (BDD) System zur automatischen Fahrzeugdiagnose eingesetzt. BDD ist weltweit in den meisten Mercedes-Benz Werkstätten im Einsatz und umfasst die Erfahrungen von Millionen von Werkstattbesuchen. Im Folgenden soll der Hauptanwendungsfall von BDD näher beleuchtet werden.

Wenn ein Mercedes mit einem Problem in die Werkstatt kommt, werden zuerst die Fehlercodes aus den Steuergeräten des Fahrzeugs ausgelesen und den Werkstatt Mitarbeitern im Diagnoseprogramm *XENTRY* präsentiert. Die Analyse der Fehlercodes ist der Startpunkt für die *Fahrzeugaufnahme* und die *Fahrzeugdiagnose*. Bei der Fahrzeugaufnahme werden die Kosten und die Dauer der Reparatur geschätzt und erste Hinweise hinsichtlich des Arbeitsauftrags an die Mechaniker weitergegeben. Anschließend folgt die Fahrzeugdiagnose, bei der mögliche Schadensursachen bewertet und verdächtige Komponenten getestet werden. Ziel von BDD ist es, diesen Prozess mit Hilfe entsprechender Reparaturprognosen zu beschleunigen und die Arbeit der Werkstatt Mitarbeiter zu erleichtern. Die Reparaturprognosen von BDD werden auf der Basis der Fehlercodes aus den Steuergeräten getroffen und lassen sich in drei Oberklassen einordnen:

- *Arbeitspositionen* (ASRA-Code): z. B. Luftfilter reinigen.

1 Einleitung

- *Schadensteile* (PARTS-Code): z. B. Luftfiltereinsatz (6510940100).
- *Schadensschlüssel* (SSL7-Code): z. B. Luftfiltereinsatz, verstopft.

Da ein Fahrzeug mehrere Schäden haben kann, wenn es in die Werkstatt kommt, werden zu jeder Oberklasse die drei Prognosen mit der höchsten Wahrscheinlichkeit dargestellt. Die Werkstatt Mitarbeiter können die Ergebnisse von BDD über eine Registerkarte in XENTRY aufrufen (siehe Abbildung 1.3) und bekommen im Optimalfall einen schnellen Überblick, welche Komponenten zu testen sind.

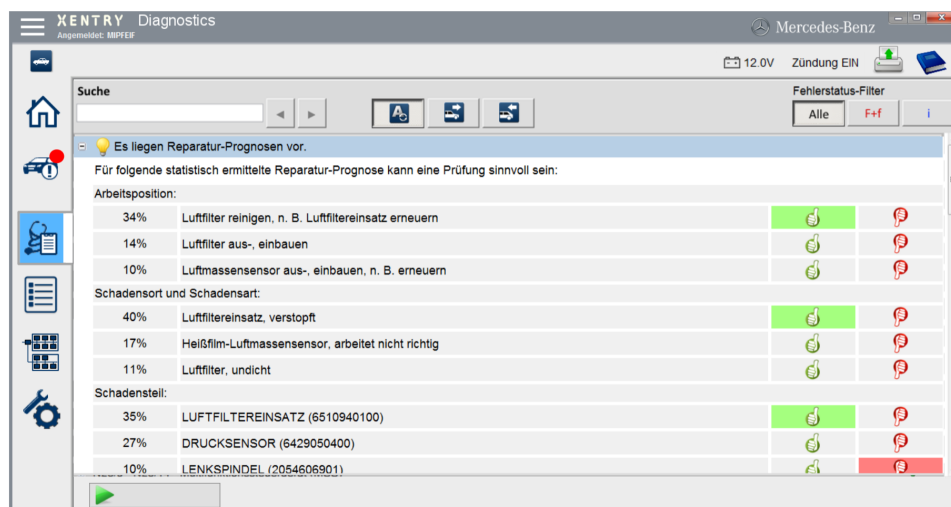


ABBILDUNG 1.3: Darstellung der BDD Prognosen im Diagnoseprogramm XENTRY

Die aktuelle Lösungsvariante von BDD basiert auf einer sogenannten *Pipeline*, bei der mehrere Programme in einem Arbeitsfluss (Workflow) miteinander verbunden werden. Die BDD-Pipeline sieht insgesamt vier Schritte vor: Den Datenabzug, die Datenaufbereitung, das Training und die Bewertung.

Als zentrale Datenquelle von BDD dient das interne Datenlager *Advanced Quality Analysis* (AQUA). AQUA beinhaltet die Diagnose-, Garantie- und Kundenzdaten von allen Daimler-Fahrzeugen und spielt eine wichtige Rolle bei der Qualitätssicherung im Konzern.

Nach dem Datenabzug aus AQUA folgt die Datenaufbereitung. Dabei werden u. a. die verschiedenen Fahrzeugbaureihen sowie die drei Oberklassen von

Reparaturcodes in verschiedene Datensätze getrennt. Des Weiteren werden einige Reparaturcodes (z. B. Lackschäden) herausgefiltert, die nicht anhand der Fehlercodes aus den Steuergeräten vorhergesagt werden können .

Beim Training von BDD werden bereits Maschinelle Lernverfahren eingesetzt. Genauer gesagt wird *XGBoost* in Kombination mit der *binären Relevanz* (One-versus-Rest) Methode (siehe Kapitel 3.2.1) verwendet. XGBoost ist eine beliebte Implementierung von *Gradient Boosting Machines* im Zusammenspiel mit Entscheidungsbäumen [CG16].

Nach dem Training folgt die Bewertung. Dabei werden die trainierten Modelle anhand ausgewählter Metriken (siehe Kapitel 2.3) evaluiert. Die vier Schritte der BDD-Pipeline werden in Abbildung 1.4 nochmal zusammengefasst dargestellt.

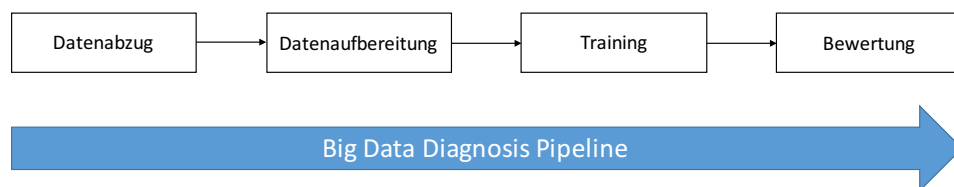


ABBILDUNG 1.4: Die vier Schritte der BDD-Pipeline

Zu betonen ist, dass BDD weltweit im Einsatz ist und einen wichtigen Beitrag zur Unterstützung der Werkstatt Mechaniker leistet. Da eine schnelle und genaue Fahrzeugdiagnose auch positive Auswirkungen auf die Kundenzufriedenheit und den Wert der Marke haben, gilt es die Reparaturprognosen von BDD zu optimieren.

1.3 Zielsetzung

Das Ziel der vorliegenden Arbeit besteht darin, die Reparaturprognosen von BDD durch den Einsatz von alternativen Maschinellen Lernverfahren zu verbessern und so die Werkstatt Mechaniker bestmöglich bei ihren Tätigkeiten zu

1 Einleitung

unterstützen. Nach einer Einordnung der Problemstellung von BDD ins Maschinelle Lernen, beabsichtigt diese Arbeit, einen Überblick über die möglichen Metriken und Lösungsvarianten bei der Multi-Label Klassifikation zu geben.

Den Forschungsschwerpunkt bildet dann die Frage, ob durch den Einsatz von ANNs eine Leistungssteigerung von BDD möglich ist. Im Rahmen dieser Arbeit wurde hierfür eine Anwendung in Python entwickelt, die sich hinsichtlich der BDD-Pipeline vor allem auf die Datenaufbereitung und das Training fokussiert. Vor diesem Hintergrund werden in dieser Arbeit die Grundlagen von ANNs vorgestellt. Anschließend wird der Einsatz von ANNs in der BDD-Pipeline genauer untersucht. Dabei werden verschiedene Netzwerkarchitekturen und Hyperparameter Kombinationen getestet und systematisch evaluiert.

Basierend auf den Resultaten der Evaluation soll schließlich diskutiert werden, ob der Einsatz von ANNs eine Alternative für das Produktivsystem von BDD darstellt und welche Netzwerkarchitekturen sowie Hyperparameter Kombinationen die besten Ergebnisse versprechen.

1.4 Verwandte Arbeiten

Bei der Problemstellung von BDD handelt es sich um eine *Multi-Label Klassifikation* mit Daten von Millionen von Werkstattbesuchen und Tausenden verschiedenen Klassen. Diese Rahmenbedingungen sind im Zeitalter von *Big Data* keine Seltenheit mehr. *Chimera* ist z. B. ein System von Walmart, das Millionen von Produkten in über 5000 Produktgruppen klassifiziert. Das System nutzt eine Kombination aus Maschinellern Lernen, explizit definierten Regeln und *Crowdsourcing*, um eine genaue und kosteneffiziente Klassifikation zu ermöglichen. Eine Lösung, die ausschließlich auf Maschinellen Lernverfahren basiert, wird hier von den Verantwortlichen abgelehnt. Das liegt u. a. daran, dass nicht ausreichend Trainingsdaten für die zahlreichen Klassen generiert werden können.[SRYD14]

Zu betonen ist, dass bei der Recherche keine vergleichbare Arbeit im Bereich der Fahrzeugdiagnose gefunden werden konnte. In Bezug auf die Multi-Label Problemstellung finden sich jedoch zahlreiche verwandte Arbeiten in anderen Gebieten wie der Bioinformatik [ZZ06], der Codierung von medizinischen Diagnosen [LYNB08], der Extraktion von Informationen aus dem Internet [TRN09] sowie der Kategorisierung von Szenen [BLSB04], Textdokumenten [KTV08] und Produkten [SRYD14]. Grundsätzlich können Multi-Label Klassifikationen mit verschiedenen Lösungsansätzen (siehe Kapitel 3.2) und Algorithmen gelöst werden. ANNs wurden im Multi-Label Kontext z. B. für die funktionale Genomanalyse und die Textkategorisierung eingesetzt [ZZ06].

1.5 Aufbau der Arbeit

In Kapitel 2 wird die Problemstellung von BDD ins Maschinelle Lernen eingeordnet. Dabei werden die Art der Trainingserfahrung, der Aufgabentyp und das Leistungsmaß beschrieben. Darüber hinaus wird das *Bias-Varianz Dilemma* erläutert und der für diese Arbeit verwendete BDD-Datensatz untersucht. Kapitel 3 gibt einen Überblick über die verschiedenen Metriken und Lösungsvarianten bei der Multi-Label Klassifikation. Kapitel 4 beschäftigt sich mit den Grundlagen von ANNs. Dabei wird u. a. der Aufbau eines *Multilayer-Perzeptrons* vorgestellt und das Training von ANNs erläutert. In Kapitel 5 geht es um den Einsatz von ANNs in der BDD-Pipeline. Dabei wird näher auf die Datenaufbereitung und das Training der Modelle mit verschiedenen Netzwerkarchitekturen eingegangen. Außerdem werden die Ergebnisse der finalen Evaluation präsentiert und diskutiert. Kapitel 6 fasst diese Arbeit zusammen und gibt einen Ausblick.

2 Einordnung der Problemstellung ins Maschinelle Lernen

Mittlerweile gibt es viele verschiedene Arten von Maschinellen Lernsystemen. Deshalb wird die Problemstellung von BDD nun ins Maschinelle Lernen eingeordnet. Zu Beginn dieses Kapitels wird in Zusammenhang mit der bereits zitierten Definition von Mitchell [Mit97] (siehe Kapitel 1.1) die Art der Trainingserfahrung, der Aufgabentyp und das Leistungsmaß von BDD beschrieben. Anschließend wird das Bias-Varianz Dilemma erläutert, das eines der fundamentalen Probleme beim Maschinellen Lernen darstellt. Zu guter Letzt wird der in dieser Arbeit verwendete BDD-Datensatz genauer untersucht.

2.1 Trainingserfahrung

Beim Maschinellen Lernen wird zwischen vier verschiedenen Formen des Lernens unterschieden: *überwachtes*, *unüberwachtes*, *teilüberwachtes* und *bestärkendes Lernen* [Gér17]. Beim überwachten Lernen enthalten die Trainingsdaten auch die entsprechenden Lösungen, die in der Regel von Menschen annotiert werden. Zu den Input Daten gibt es demnach immer auch die entsprechenden Output Daten. Ziel ist es, eine Funktion zu approximieren, die den Input bei unbekannten Beispielen auf den entsprechenden Output abbildet [Bis06].

Unüberwachtes Lernen beschäftigt sich mit der Mustererkennung in den Eingabedaten ohne Beispiele für den erwarteten Output. Ziel ist es, die zugrunde-

2 Einordnung der Problemstellung ins Maschinelle Lernen

liegenden Strukturen in den Daten zu finden. Klassische Anwendungsgebiete sind u. a. das *Clustering* und die *Dimensionsreduktion* [Gér17].

Teilüberwachtes Lernen ist eine Kombination aus überwachtem und unüberwachtem Lernen, bei der die richtigen Lösungen nur zum Teil gegeben sind. Dies kann nützlich sein, da das Annotieren von Lösungen bei großen Datensätzen sehr viel Zeit und Geld kostet. Beim teilüberwachten Lernen ist der Anteil an Beispielen ohne die entsprechenden Lösungen in der Regel deutlich größer. Das liegt u. a. auch daran, dass die Ergebnisse vom unüberwachten Lernen mit einer kleinen Menge von annotierten Lösungen stark verbessert werden können.[Gér17]

Bestärkendes Lernen ist ein wenig spezieller, da es sich bei der Trainingserfahrung nicht auf einen aufbereiteten Datensatz konzentriert. Stattdessen beinhaltet es einen *Agenten*, der einen Umgebungsraum überwacht, Aktionen auswählt und durchführt. Je nach dem, ob diese Aktionen gut oder schlecht sind, wird der Agent belohnt oder bestraft. Ziel des Agenten ist es, mit Hilfe des Feedbacks aus dem Umgebungsraum eine Handlungsstrategie zu erlernen, durch die er so viele Belohnungen wie möglich bekommt.[GBCB16]

Ein weiteres Kriterium bei der Einordnung der Trainingserfahrung ist die Unterscheidung zwischen *Batch-Learning* und *Online-Learning*. Entscheidend ist hier, ob das System in der Lage ist, inkrementell von einem Datenstrom zu lernen oder immer wieder ein neues Modell mit allen verfügbaren Daten trainiert werden muss. Beim Batch-Learning wird ein System stets mit allen verfügbaren Daten trainiert. Da das Training dementsprechend länger dauert und einige Ressourcen in Anspruch nimmt, wird es offline durchgeführt und nicht sofort wiederholt, sobald neue Daten verfügbar sind. Nach dem Training wird das System produktiv eingesetzt und lernt vorerst nicht mehr dazu.[Gér17]

Wenn sich ein System besonders schnell auf neue Daten anpassen muss, ist Online-Learning zu bevorzugen. Online-Learning ermöglicht es, ein System inkrementell zu trainieren bzw. das bestehende Modell fortlaufend mit neuen Beispielen zu verbessern. Die neuen Daten werden dem System hier ohne große Verzögerung bereitgestellt, so dass es praktisch immer auf dem neuesten

Stand ist [Gér17]. Bei Computerprogrammen, die Aktienkurse beobachten und Transaktionen tätigen, ist eine schnelle Anpassung auf neue Kursdaten z. B. absolut notwendig. Außerdem können beim Online-Learning einige Ressourcen gespart werden. Zum einen muss das Modell nicht von Grund auf neu berechnet werden. Zum anderen braucht man die Daten nach dem Training nicht mehr speichern, sondern nur noch das resultierende Modell. Die Herausforderung beim Online-Learning liegt im Umgang mit fehlerhaften Daten und der Anpassung des Systems mit einer angemessenen Lernrate.[Gér17]

Bei der Trainingserfahrung von BDD handelt es sich um überwachtes Lernen, da es einen Datensatz gibt, der sowohl die Fehlercodes aus den Steuergeräten, als auch die entsprechenden Reparaturcodes enthält, die von den Werkstatt Mitarbeitern annotiert wurden. Des Weiteren ist Batch-Learning zu bevorzugen, da eine schnelle Anpassung des Modells nicht zwingend erforderlich ist. Es reicht aus, das Modell ca. einmal in der Woche mit den Daten der neuesten Werkstattbesuche zu trainieren.

2.2 Aufgabentyp

Die Anwendungsfälle des Maschinellen Lernens sind vielseitig. Beim überwachten Lernen, der Trainingserfahrung von BDD, gibt es prinzipiell zwei verschiedene Aufgabentypen: die *Regression* und die *Klassifikation*. Bei der Regression werden stetige Werte vorhergesagt wie z. B. der Preis eines Grundstücks auf der Basis von anderen Grundstücken in der Nachbarschaft [GBCB16].

Bei der Klassifikation muss ein Computerprogramm hingegen ermitteln, zu welcher diskreten Klasse ein bestimmter Input zuzuordnen ist. Dies kann auf unterschiedliche Weise erfolgen. In der Regel wird eine Funktion definiert, die auf Basis von einem Merkmalsvektor (Feature Vektor) einen numerischen Wert ausgibt. Dieser numerische Wert wird dann zur Identifikation der diskreten Klasse verwendet. Möglich ist auch, dass die Funktion eine Wahrscheinlichkeitsverteilung über die Klassen ausgibt und davon ausgehend diejenige mit der höchsten Wahrscheinlichkeit vorhergesagt wird.[GBCB16]

2 Einordnung der Problemstellung ins Maschinelle Lernen

Klassischerweise gibt es bei der Klassifikation immer nur eine richtige Klasse pro Trainingsinstanz. Bei der binären Klassifikation gibt es genau zwei mögliche Output-Klassen (z. B. Spam oder kein Spam) und bei der Multi-Klassen Klassifikation mindestens drei (z. B. Erkennung von handgeschriebenen Zahlen). In der Praxis treten jedoch auch häufig Problemstellungen auf, bei dem jedes Beispiel mit einer Menge von Labels assoziiert ist. In diesem Fall handelt es sich um eine Multi-Label Klassifikation.[ZZ14]

Ziel von BDD ist es, die passenden Reparaturcodes anhand der Fehlercodes aus den Steuergeräten vorherzusagen. Dafür muss das System die Zusammenhänge zwischen den Schadens- und Diagnosedaten lernen und Verallgemeinerungen für neue Beispiele ableiten. Da die Reparaturcodes diskrete Klassen darstellen und ein Fahrzeug mehrere Schäden gleichzeitig haben kann, ist der Aufgabentyp von BDD eine Multi-Label Klassifikation. Den Werkstatt Mitarbeitern sollen die vorhergesagten Labels mit einer Wahrscheinlichkeit dargestellt werden. Folglich gilt es eine Funktion zu approximieren, die als Ergebnis eine Wahrscheinlichkeitsverteilung über die Reparaturcodes ausgibt. Auf diese Weise können den Werkstatt Mitarbeitern dann zu jeder Oberklasse die drei Reparaturcodes mit der höchsten Wahrscheinlichkeit präsentiert werden.

2.3 Leistungsmaß

Um die Leistung von Maschinellen Lernverfahren bewerten zu können, muss ein Leistungsmaß definiert werden, das messbar ist und zur jeweiligen Aufgabe passt. Bei der Klassifikation wird z. B. oft die *Richtigkeit* (Accuracy) eines Modells gemessen, also der Anteil an Trainingsbeispielen, die vom Modell korrekt klassifiziert wurden. Um die Leistung eines Systems zu prüfen, wird grundsätzlich ein Testdatensatz verwendet. Dieser stellt im Optimalfall eine unabhängige und identisch verteilte Zufallsstichprobe des gesamten Datensatzes dar und ist getrennt vom Trainingsdatensatz. Auf diese Weise ist erkennbar, ob das Modell in der Lage ist, unbekannte Daten richtig zu klassifizieren.[GBCB16]

Bei BDD sind aktuell drei Metriken im Einsatz: *Attempt Rate*, *Top-3 Precision* und *Classes Largely Correct*. *Attempt Rate* spiegelt den Anteil an Beispielen wider, für die eine Vorhersage gemacht wird. Entscheidend ist hier, ob eine Klasse mit einer festgelegten Mindestwahrscheinlichkeit vorhergesagt werden kann. Gibt es bei einem Werkstattbesuch keine Klasse, die diese Mindestwahrscheinlichkeit erreicht, wird keine Vorhersage gemacht. Auf diese Weise versucht man zu verhindern, die Werkstatt Mitarbeiter mit unwahrscheinlichen Vorhersagen zu irritieren.

Top-3 Precision zeigt den Anteil an Beispielen, für die eine Vorhersage gemacht wird und bei denen einer der drei Klassen mit der höchsten Wahrscheinlichkeit korrekt vorhergesagt wurde. Diese Metrik ist sehr wichtig, da den Werkstatt Mitarbeitern zu jeder Oberklasse von Reparaturcodes die drei wahrscheinlichsten Vorhersagen dargestellt werden.

Classes Largely Correct gibt die Anzahl an Klassen an, die mit einer Wahrscheinlichkeit von mehr als 25 % vorhergesagt werden. Zur Verdeutlichung dieser Metrik werden in Tabelle 2.1 fünf Trainingsbeispiele sowie vier Klassen von Reparaturcodes (Motor, Bremse, Auspuff und Kupplung) dargestellt.

	Vorhergesagt	Korrekt
1	Motor, Bremse	Motor, Auspuff
2	Bremse	Kupplung
3	Bremse, Auspuff	Bremse
4	Motor, Bremse, Kupplung	Auspuff
5	Bremse	Motor, Auspuff

TABELLE 2.1: Beispieldatensatz zur Metrik *Classes Largely Correct*

Der Motor wird zwei mal vorhergesagt und ist in einem Fall korrekt. Die Bremse wird fünf mal prognostiziert und ist auch in einem Fall korrekt. Die Klassen Auspuff und Kupplung werden jeweils einmal vorhergesagt und sind nie korrekt. Folglich fällt nur der Motor unter die Kategorie *Classes Largely Correct*,

da er in 50 % der Fällen korrekt vorhergesagt wird. Die Bremse scheitert knapp mit 20 % an der vorgegebenen Mindestwahrscheinlichkeit von 25 %.

2.4 Das Bias-Varianz Dilemma

Beim Maschinellen Lernen geht es grundsätzlich darum ein Modell zu trainieren, das bestmöglich auf unbekannte Daten anwendbar ist. Diese Fähigkeit der Generalisierung kann jedoch nur indirekt über die Optimierung eines Modells auf die Trainingsdaten erreicht werden [Cho18]. Ein fundamentales Problem ist hier die Spannung zwischen Bias und Varianz. Bias stellt die vereinfachten Annahmen eines Modells dar, durch die eine bestimmte Zielfunktion leichter erlernt werden kann. Die Varianz drückt hingegen aus, wie stark die Vorhersagen eines Modells voneinander abweichen, insbesondere bei unbekannten Daten.[Bis06]

Ein entscheidender Faktor beim Bias-Varianz Dilemma ist die Komplexität des Modells, also z. B. der Grad an Polynomen bei der Zielfunktion [GBCB16]. In Abbildung 2.1 werden drei unterschiedliche Modelle dargestellt, die auf Basis eines Datensatzes trainiert wurden.

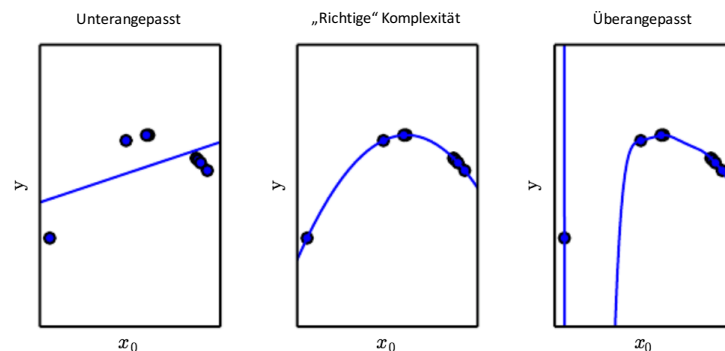


ABBILDUNG 2.1: Auswirkungen der Modell Komplexität auf das Bias-Varianz Dilemma [GBCB16]

Das linke Modell repräsentiert eine lineare Funktion, die einen hohen Bias hat und *unterangepasst* ist. Das rechte Modell stellt eine Polynomfunktion

neunten Grades dar, die eine hohe Varianz verursacht und *überangepasst* ist. Das mittlere Modell wurde mit einer quadratischen Funktion trainiert und scheint weder unter- noch überangepasst zu sein. Dementsprechend wurde die richtige Komplexität gewählt und es kann von einer guten Leistung bei der Generalisierung ausgegangen werden.[GBCB16]

Zu betonen ist, dass ein hoher Bias mit einer Unteranpassung und eine hohe Varianz mit einer Überanpassung des Modells einhergeht. Bei einer Unteranpassung werden wichtige Zusammenhänge in den Daten nicht berücksichtigt. Bei der Überanpassung lernt ein Modell hingegen sehr spezifische Muster, die die Leistung bei der Generalisierung beeinträchtigen können.[Cho18]

Im Idealfall hat ein Modell demnach einen kleinen Bias und eine kleine Varianz. Dies stellt jedoch ein Dilemma dar, da sich die beiden Fehlerquellen gegenseitig beeinflussen. Denn eine Verringerung der Varianz löst automatisch eine Erhöhung des Bias aus und umgekehrt nimmt auch die Varianz zu, wenn man den Bias einschränkt.[GBCB16]

Um die Leistung der Vorhersagen zu optimieren, gilt es also die richtige Balance zwischen Bias und Varianz zu finden. Hierbei ist es wichtig, die jeweilige Fehlerquelle zu erkennen. Wenn sowohl der Trainings-, als auch der Testfehler zu hoch sind, dann ist das Modell unterangepasst und ein hoher Bias ist das Problem. Folglich gilt es, die Komplexität des Modells zu erhöhen. Wenn der Trainingsfehler hingegen sehr klein ist und der Testfehler deutlich größer, dann ist das Modell überangepasst und leidet an einer hohen Varianz.[Bis06]

Die beste Möglichkeit, eine Überanpassung zu verhindern, ist es, mehr Trainingsdaten zu finden. Wenn das nicht möglich ist, muss das Modell beim Lernen eingeschränkt werden, so dass es sich auf die wichtigsten Muster konzentriert [Cho18].

2.5 Charakterisierung der BDD-Daten

Der initiale Datensatz, der für diese Arbeit verwendet wurde, resultiert aus einem Datenabzug von AQUA (siehe Abbildung 2.2).

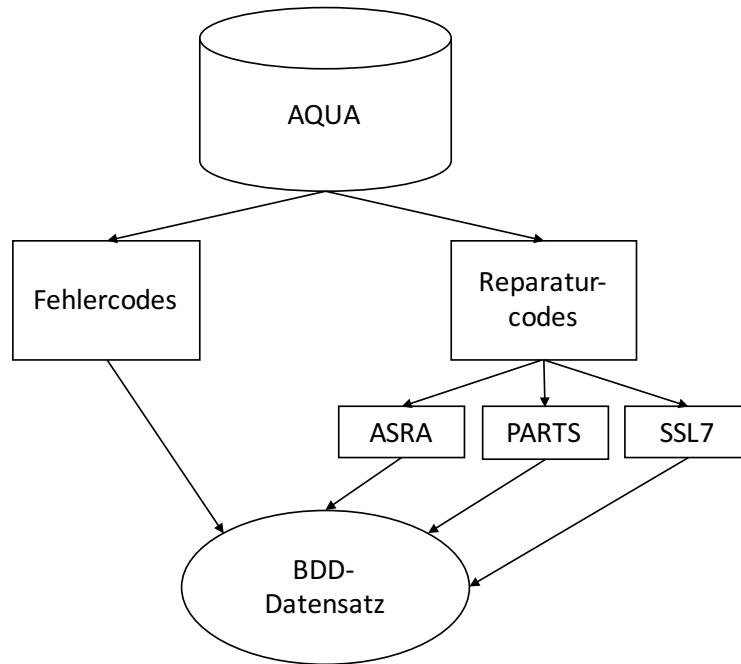


ABBILDUNG 2.2: Initialer BDD-Datensatz nach dem Abzug aus AQUA

Er umfasst die Schadens- und Diagnosedaten von über 2,7 Millionen Werkstattbesuchen. Die Daten sind in einer Matrix bzw. Tabelle gespeichert, die die Fehler- und Reparaturcodes sämtlicher Mercedes-Benz Baureihen (A-Klasse, B-Klasse etc.) enthält. Jede Zeile in dieser Matrix repräsentiert einen Werkstattbesuch und jede Spalte entweder einen Fehler- oder Reparaturcode. Insgesamt gibt es 47396 verschiedene Fehlercodes und 162729 Reparaturcodes, die ausschließlich binär-codiert sind. Jede Spalte sagt folglich aus, ob ein bestimmter Fehler- bzw. Reparaturcode aufgetreten ist oder nicht.

Aufgrund der hohen Anzahl an Spalten besteht die Matrix hauptsächlich aus Nullen. Es handelt sich daher um eine extrem dünnbesetzte Matrix (zu mehr als 99,9 %), was insbesondere Auswirkungen auf das Speicherformat und die Auswahl des Algorithmus bzw. der Implementierung hat. Bei der Nutzung des

2.5 Charakterisierung der BDD-Daten

CSR-Formats (Compressed Sparse Row) kann z. B. viel Speicherplatz gespart werden, da hier die Positionen der Nullen nicht explizit vom Computer gespeichert werden. Was die Auswahl der Implementierung betrifft, so kann XGBoost z. B. sehr gut mit dünnbesetzten Matrizen umgehen und ist deshalb auch sehr schnell [CG16].

Da ein Fahrzeug mehrere Probleme haben kann, gilt es die BDD-Daten in Hinblick auf die Multi-Label Problemstellung zu analysieren. Dafür werden zwei Kennzahlen definiert, mit denen man einen Multi-Label Datensatz beschreiben kann: Die Label-Kardinalität (LK) berechnet die durchschnittliche Anzahl an Labels pro Instanz und die Label-Dichte teilt darüber hinaus noch durch die Anzahl aller möglichen Labels [TK06]. Für die Formeln gilt folgende Notation: $D = \{(x_i, Y_i) | 1 \leq i \leq |D|\}$ steht für einen Multi-Label Datensatz mit $|D|$ Beispielen, Y für eine Label Menge und L für die Menge aller Labels.

- Label-Kardinalität: $LK(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} |Y_i|$
- Label-Dichte: $LD(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i|}{|L|}$

Neben LK und LD werden in Tabelle 2.2 außerdem noch folgende Charakteristiken präsentiert: die gesamte Anzahl an Labels, die minimale und maximale Anzahl an Labels pro Werkstattbesuch und der Median.

	Labels	Min	Max	Median	LK	LD
Reparaturcodes	162729	1	240	6	7.86	4.83e-05
ASRA-Codes	36029	0	89	3	3.98	0.00011
PARTS-Codes	84491	0	201	1	2.55	3.01e-05
SSL7-Codes	42209	0	39	1	1.34	3.12e-05

TABELLE 2.2: Charakterisierung der BDD-Daten

Die Zeilen zeigen die Ergebnisse für alle Reparaturcodes kombiniert und jeweils für die drei Oberklassen getrennt. Zu betonen ist erstmal, dass die insgesamt 162729 Labels ein hochdimensionales Problem darstellen, das durch

2 Einordnung der Problemstellung ins Maschinelle Lernen

die Trennung der Oberklassen reduziert werden kann. Da die minimale Anzahl an Labels bei den Werkstattbesuchen im kombinierten Fall eins und bei den getrennten Oberklassen jeweils null ist, gibt es folglich Werkstattbesuche, bei denen zu mindestens einer der Oberklassen kein Code annotiert wurde. Vorstellbar ist z. B. ein Werkstattbesuch mit zwei ASRA-Codes und einem SSL7-Code, aber ohne ein PARTS-Code.

Was die maximale Anzahl an Labels bei den Werkstattbesuchen betrifft, so ist zu erkennen, dass es verglichen mit dem Median und der Label-Kardinalität große Ausreißer nach oben gibt. Demnach haben die meisten Fahrzeuge beim Werkstattbesuch eher kleinere Schäden und nur wenige größere Probleme. Darüber hinaus zeigen die Werte von LK, dass nicht nur der Datensatz mit den kombinierten Reparaturcodes ein Multi-Label Problem darstellt, sondern auch die Datensätze mit den getrennten Oberklassen. Die Resultate der Label-Dichte verdeutlichen, dass es sich um eine extrem dünnbesetzte Matrix handelt.

Die Verteilung der Klassen in BDD ist generell sehr ungleichmäßig. So kommen bestimmte Fehler- und Reparaturcodes sehr häufig vor und andere nur sehr selten. Abbildung 2.3 und 2.4 zeigen die gruppierte Häufigkeiten der Fehler- bzw. Reparaturcodes in einer exponentiellen Darstellung. Bei den Fehlercodes gibt es etwa 20000, die 1-10 mal im Datensatz vorkommen und 5, die mehr als 100000 mal enthalten sind. Bei den Reparaturcodes sieht es ähnlich aus. Hier gibt es rund 110000, die 1-10 mal im Datensatz vorkommen und 10, die mehr als 100000 mal enthalten sind. Von allen Fehlercodes kommen rund 90 % 1-1000 mal vor. Bei den Reparaturcodes sind es in diesem Bereich sogar ca. 98 %.

Die Abbildungen 2.5 und 2.6 zeigen die Anzahl an Fehler- und Reparaturcodes bei den einzelnen Werkstattbesuchen. In den meisten Fällen wurden drei Fehlercodes und 5 Reparaturcodes annotiert. Insgesamt haben rund 80 % der Werkstattbesuche 1-10 Fehlercodes sowie 1-10 Reparaturcodes. Zu erkennen ist, dass die Anzahl an Werkstattbesuchen nach dem jeweiligen Maximum mit der steigenden Anzahl an Fehler- und Reparaturcodes exponentiell abnimmt.

2.5 Charakterisierung der BDD-Daten

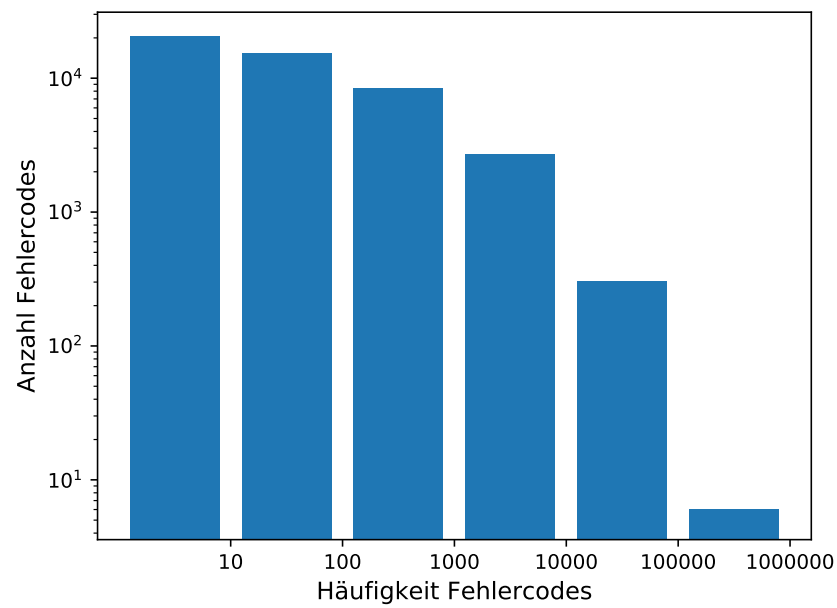


ABBILDUNG 2.3: Gruppierte Häufigkeiten der Fehlercodes

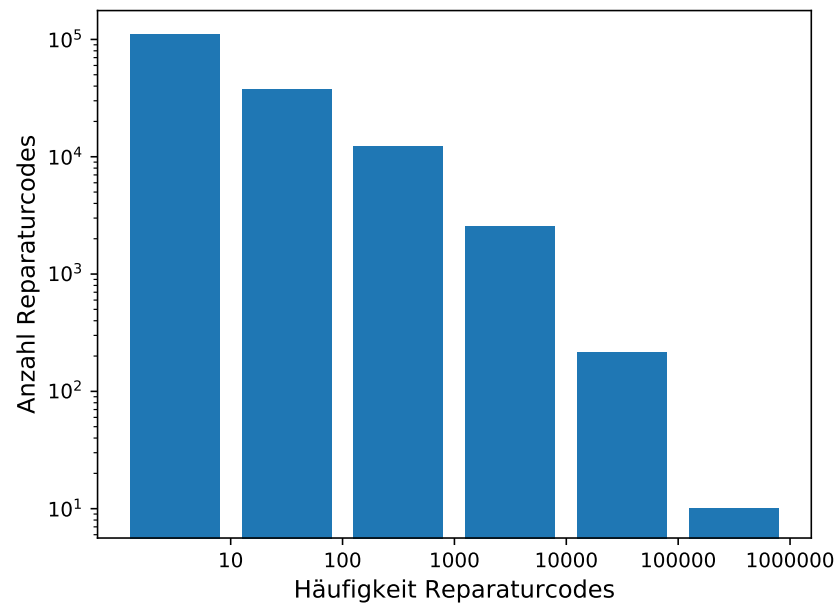


ABBILDUNG 2.4: Gruppierte Häufigkeiten der Reparaturcodes

2 Einordnung der Problemstellung ins Maschinelle Lernen

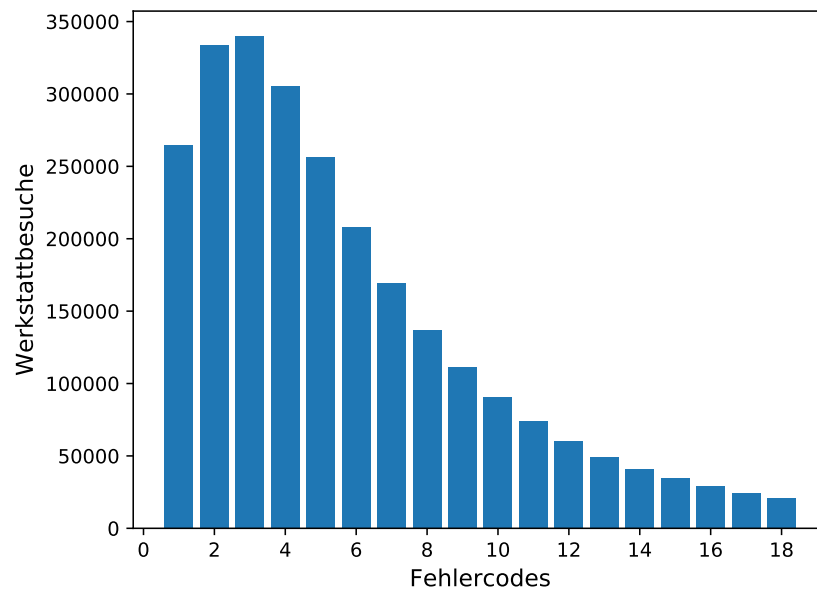


ABBILDUNG 2.5: Analyse der Werkstattbesuche mit Fokus auf die Fehlercodes

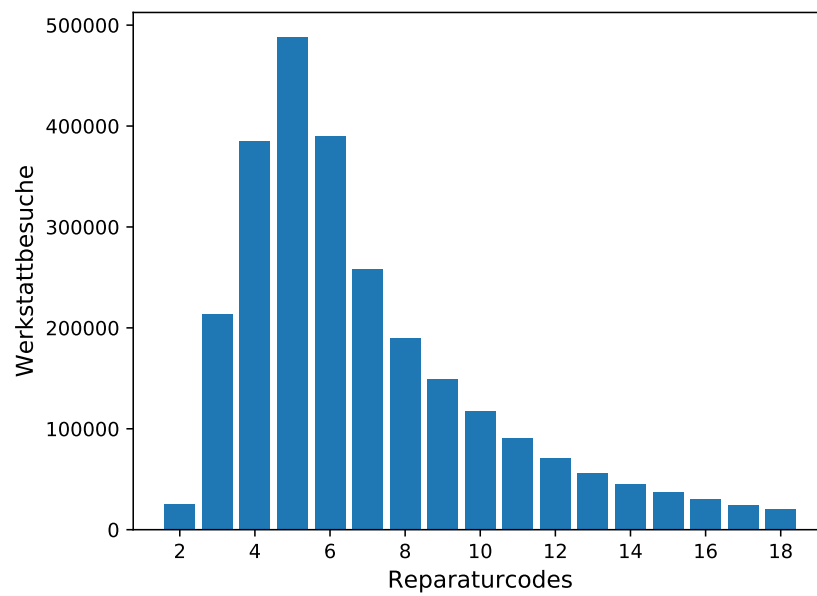


ABBILDUNG 2.6: Analyse der Werkstattbesuche mit Fokus auf die Reparaturcodes

2.5 Charakterisierung der BDD-Daten

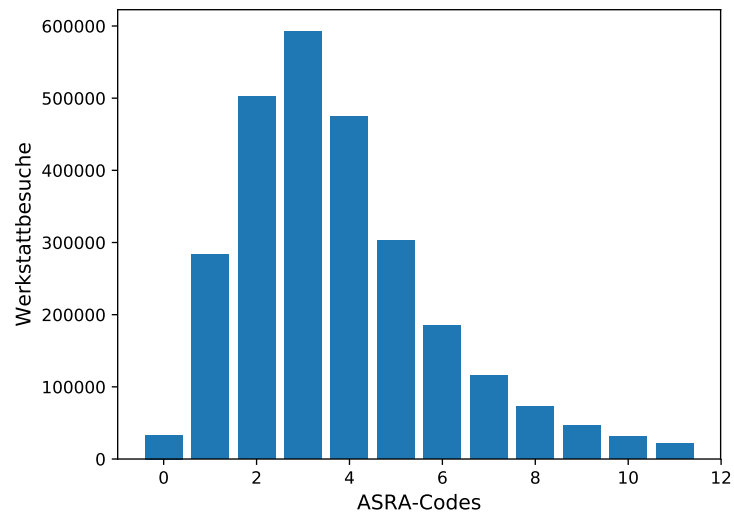


ABBILDUNG 2.7: Analyse der Werkstattbesuche mit Fokus auf die ASRA-Codes

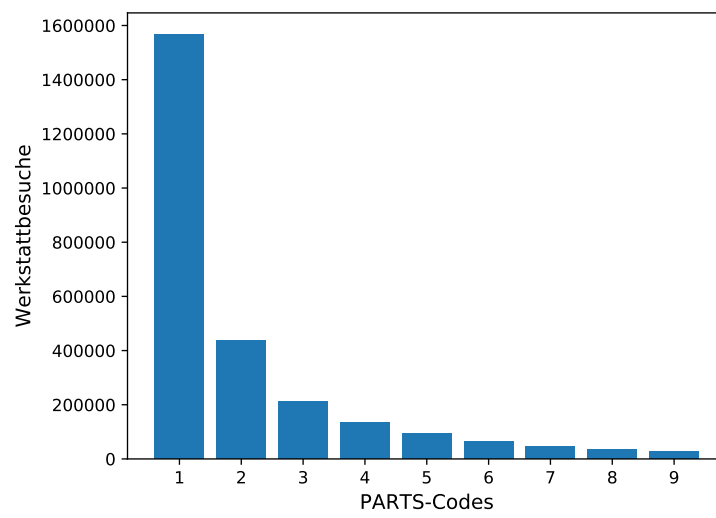


ABBILDUNG 2.8: Analyse der Werkstattbesuche mit Fokus auf die PARTS-Codes

2 Einordnung der Problemstellung ins Maschinelle Lernen

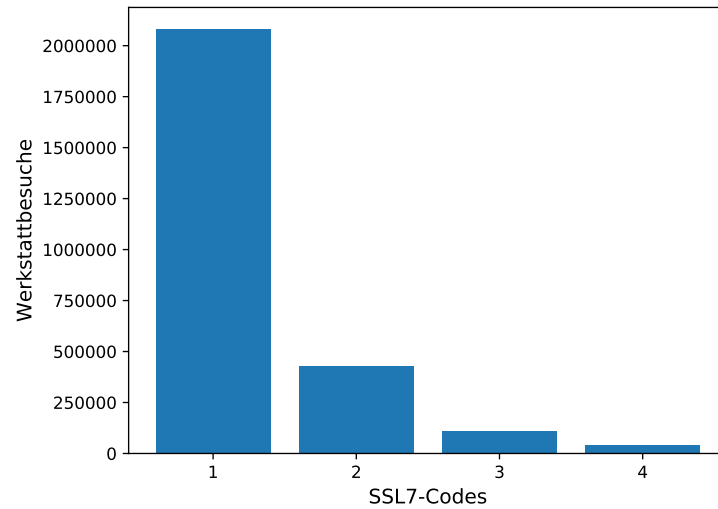


ABBILDUNG 2.9: Analyse der Werkstattbesuche mit Fokus auf die SSL7-Codes

Die Abbildungen 2.7, 2.8 und 2.9 zeigen die Anzahl an Reparaturcodes bei den einzelnen Werkstattbesuchen mit Fokus auf die jeweilige Oberklasse. Hier gibt es bei den meisten Beispielen drei ASRA-Codes, ein PARTS-Code und ein SSL7-Code. Insgesamt haben ca. 95 % der Werkstattbesuche 1-3 SSL7-Codes und rund 80 % 1-3 PARTS-Codes sowie 1-5 ASRA-Codes.

Zusammenfassend kann man die Problemstellung von BDD wie folgt beschreiben: Bei der Trainingserfahrung handelt es sich um überwachtes Lernen, wobei keine Notwendigkeit für ein Online-Learning System besteht und Batch-Learning deshalb zu bevorzugen ist. Der Aufgabentyp stellt eine Multi-Label Klassifikation dar, bei der die Vorhersagen für die Labels mit Wahrscheinlichkeiten anzugeben sind. Als Leistungsmaß werden die Attempt Rate, die Top-3 Precision und die Classes Largely Correct verwendet. Die Analyse der BDD-Daten hat gezeigt, dass es sich um ein hochdimensionales Problem mit einer extrem dünnbesetzten Matrix handelt. Darüber hinaus sind die Klassen allgemein sehr ungleichmäßig verteilt, was das Training und die Vorhersagen von seltenen Klassen erschwert. Nicht zuletzt gilt es bei der Auswahl des Maschinellen Lernverfahrens auch zu berücksichtigen, dass die Daten teilweise verrauscht sind. Die wichtigsten Punkte der BDD-Problemstellung werden nochmal in Abbildung 2.10 aufgeführt.

2.5 Charakterisierung der BDD-Daten

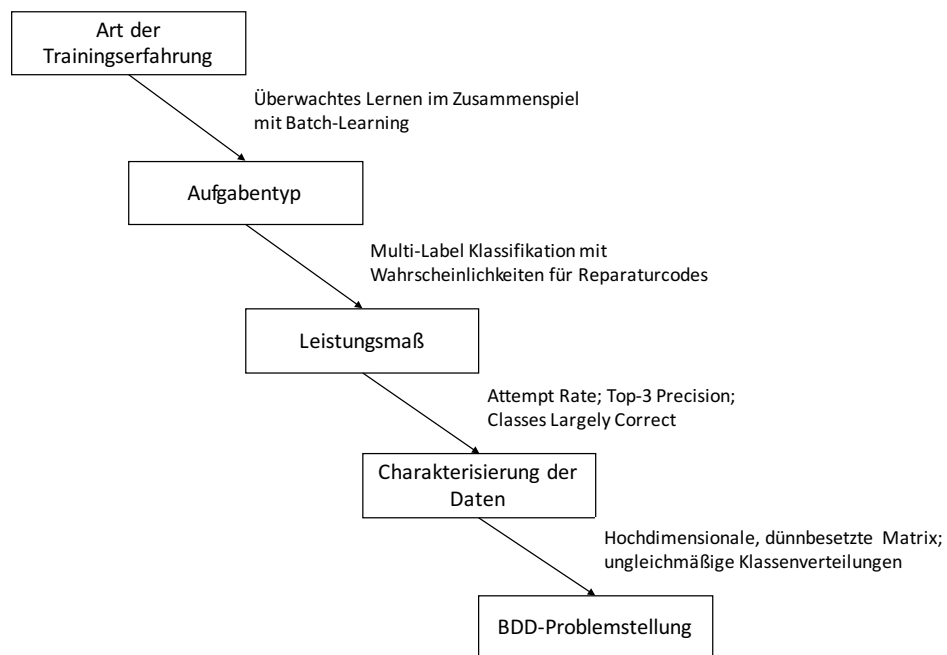


ABBILDUNG 2.10: Zusammenfassung der BDD-Problemstellung

3 Einführung in die Multi-Label Klassifikation

Typischerweise gibt es bei der Klassifikation immer nur eine richtige Klasse pro Trainingsinstanz. Dabei spricht man auch von einer *Single-Label Klassifikation*. In der Praxis kommen allerdings auch häufig Problemstellungen vor, bei dem jedes Beispiel mit einer Menge von Labels assoziiert ist. Multi-Label Problemstellungen kommen z. B. in der Bioinformatik sowie bei der Kategorisierung von Textdokumenten vor [ZZ14]. Da es sich auch beim Aufgabentyp von BDD um eine Multi-Label Klassifikation handelt, wird dieses Gebiet im Folgenden näher beleuchtet. Der Fokus liegt dabei auf den verschiedenen Metriken und Lösungsvarianten bei der Multi-Label Klassifikation.

3.1 Metriken

Multi-Label Metriken unterscheiden sich von denen, die bei der Single-Label Klassifikation genutzt werden. Grundsätzlich ist die Qualität der Vorhersagen bei Multi-Label Problemstellungen deutlich schwieriger zu bewerten, da jedes Beispiel mit einer Menge von Labels assoziiert ist. Demnach sollte bei einer Vorhersage berücksichtigt werden, ob die Labels komplett, teilweise oder gar nicht mit der tatsächlichen Lösung übereinstimmen. Diese Abwägung wird bei den Metriken der Single-Label Klassifikation nicht berücksichtigt.[ZZ14]

3 Einführung in die Multi-Label Klassifikation

Im Folgenden wird auf verschiedene Multi-Label Leistungsmaße eingegangen, die sich allgemein in drei Gruppen einteilen lassen: *Beispielorientierte*, *labelorientierte* und *Ranking* Metriken [MKGD12].

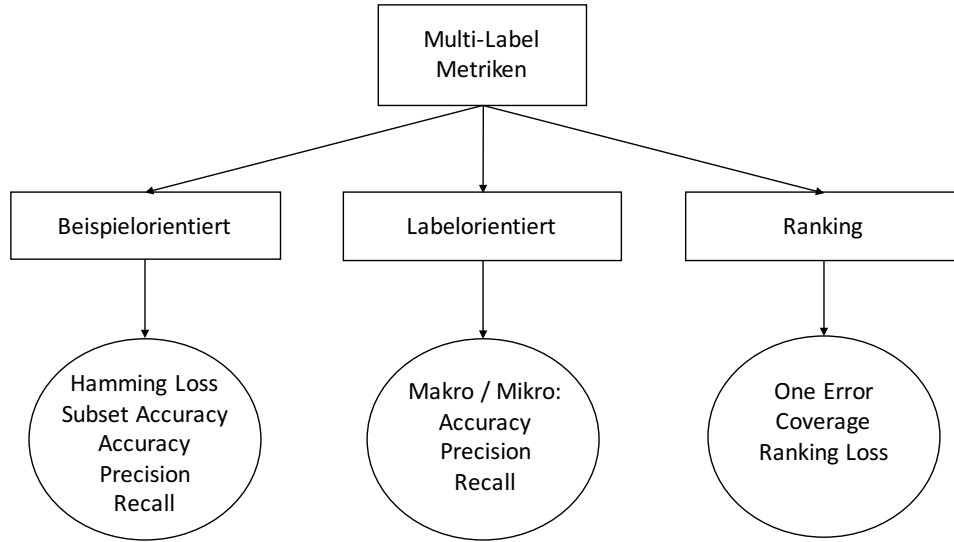


ABBILDUNG 3.1: Einordnung von Multi-Label Metriken [MKGD12]

Diese drei Gruppen sollen nun anhand der in Abbildung 3.1 aufgelisteten Metriken näher beleuchtet werden. Da die beispiel- und labelorientierten Leistungsmaße mit der entsprechenden Formel vorgestellt werden, gilt folgende Notation: $D = \{(x_i, Y_i) | 1 \leq i \leq |D|\}$ steht für einen Multi-Label Datensatz mit $|D|$ Beispielen, x für einen Feature Vektor, Y für eine Label Menge, L für die Menge aller Labels und $h()$ für einen Multi-Label Klassifikator, der als Output eine Menge an Labels liefert.

3.1.1 Beispielorientiert

Die beispielorientierten Leistungsmaße verdeutlichen die durchschnittlichen Unterschiede zwischen den tatsächlichen und den vorhergesagten Mengen an Labels in Bezug auf alle Beispiele eines Testdatensatzes [MKGD12].

- **Hamming Loss:** Zeigt den Anteil an falsch klassifizierten Labels pro Beispiel. Als falsch klassifiziert gilt, wenn ein relevantes Label vergessen oder

ein irrelevantes vorausgesagt wurde. Zu erwähnen ist, dass Δ in der Formel für die symmetrische Differenz zwischen zwei Mengen steht, was in der booleschen Logik einem *XOR* entspricht [TK06].

$$\text{hamming_loss} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{1}{|L|} |h(x_i) \Delta Y_i|$$

- **Subset Accuracy:** Berechnet den Anteil an korrekt klassifizierten Beispielen, d. h. die vorhergesagte Menge an Labels muss mit der tatsächlichen genau übereinstimmen [MKGD12]. Wenn für eine Instanz z. B. drei von vier Labels richtig vorhergesagt wurden und nur eines falsch, dann haben die drei richtigen Labels keinen Einfluss auf das Ergebnis der Metrik, da teilweise korrekte Label Mengen nicht berücksichtigt werden. Für den Multi-Label Kontext ist die Subset Accuracy folglich ein strenges Leistungsmaß, insbesondere wenn der Datensatz sehr viele verschiedene Labels beinhaltet [ZZ14]. Bei der Formel der Subset Accuracy wird die *Identitätsfunktion* $I()$ verwendet. Diese überprüft, ob die vorhergesagte Menge an Labels eine identische Abbildung der tatsächlichen Menge an Labels ist. Dabei liefert $I(true)$ den Wert 1 und $I(false)$ den Wert 0 [MKGD12].

$$\text{subset_accuracy} = \frac{1}{|D|} \sum_{i=1}^{|D|} I(h(x_i) = Y_i)$$

- **Accuracy:** Spiegelt den Anteil an korrekt klassifizierten Labels im Hinblick auf alle Instanzen wider. Die Accuracy wird anhand des *Jaccard-Koeffizienten* ermittelt, mit dem man allgemein die Ähnlichkeit zwischen zwei Mengen vergleichen kann, indem man die Größe der Schnittmenge durch die der Vereinigungsmenge teilt [TK06]. Der Jaccard-Koeffizient ist durch $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ definiert, wobei A und B zwei Mengen darstellen [ML07]. Accuracy ist nicht so streng wie die Subset Accuracy, weil teilweise korrekte Label Mengen im Ergebnis der Metrik berücksichtigt werden. Für die meisten Multi-Label Problemstellungen ist die Accuracy daher auch aussagekräftiger [ZZ14].

$$\text{accuracy} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|h(x_i) \cap Y_i|}{|h(x_i) \cup Y_i|}$$

3 Einführung in die Multi-Label Klassifikation

- Genauigkeit (Precision): Zeigt, wie viele von den vorhergesagten Labels wirklich relevant sind [TK06].

$$precision = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|h(x_i) \cap Y_i|}{|h(x_i)|}$$

- Wiedererkennung (Recall): Zeigt, wie viele von den relevanten Labels wirklich vorhergesagt wurden [TK06].

$$recall = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|h(x_i) \cap Y_i|}{|Y_i|}$$

3.1.2 Labelorientiert

Die labelorientierten Metriken bewerten die Qualität der Vorhersagen in Bezug auf die einzelnen Labels und liefern dann den Durchschnitt für alle Label Bewertungen. Die Durchschnittswerte können dabei nach dem *Makro*- oder dem *Mikro-Ansatz* ermittelt werden [MKGD12]. Bevor die entsprechenden Formeln präsentiert werden, gilt es noch die Begriffe *Tatsächlich Positive* (TP), *Tatsächlich Negative* (TN), *Falsch Positive* (FP) und *Falsch Negative* (FN) einzuführen. TP (*true* wurde prognostiziert und es ist *true*) und TN (*false* wurde prognostiziert und es ist *false*) repräsentieren richtige Vorhersagen und FP (*true* wurde prognostiziert, aber es ist *false*) sowie FN (*false* wurde prognostiziert, aber es ist *true*) stellen falsche Vorhersagen dar [ZZ14].

- Makro Accuracy, Precision, Recall:

$$makro_accuracy = \frac{1}{|L|} \sum_{j=1}^{|L|} \frac{TP_j + TN_j}{TP_j + FP_j + TN_j + FN_j}$$

$$makro_precision = \frac{1}{|L|} \sum_{j=1}^{|L|} \frac{TP_j}{TP_j + FP_j}$$

$$makro_recall = \frac{1}{|L|} \sum_{j=1}^{|L|} \frac{TP_j}{TP_j + FN_j}$$

- Mikro Accuracy, Precision, Recall:

$$mikro_accuracy = \frac{\sum_{j=1}^{|L|} TP_j + \sum_{j=1}^{|L|} TN_j}{\sum_{j=1}^{|L|} TP_j + \sum_{j=1}^{|L|} FP_j + \sum_{j=1}^{|L|} TN_j + \sum_{j=1}^{|L|} FN_j}$$

$$mikro_precision = \frac{\sum_{j=1}^{|L|} TP_j}{\sum_{j=1}^{|L|} TP_j + \sum_{j=1}^{|L|} FP_j}$$

$$mikro_recall = \frac{\sum_{j=1}^{|L|} TP_j}{\sum_{j=1}^{|L|} TP_j + \sum_{j=1}^{|L|} FN_j}$$

Beim Makro-Ansatz wird das jeweilige Verhältnis an TPs, TNs, etc. unabhängig für alle Labels berechnet und dann der Durchschnitt genommen. Dahingegen werden beim Mikro-Ansatz erst die einzelnen Beiträge der Labels an TPs, TNs, etc. summiert und anschließend das entsprechende Verhältnis ermittelt [MKGD12].

Zur Verdeutlichung des Unterschieds zwischen dem Makro- und Mikro-Ansatz wird in Tabelle 3.1 ein Beispiel mit drei Labels (Motor, Bremse, Auspuff) dargestellt, die jeweils eine unterschiedliche Anzahl an TPs und FPs haben.

	TP	FP
Motor	12	3
Bremse	16	4
Auspuff	100	900

TABELLE 3.1: Beispieldatensatz zum Makro- und Mikro-Durchschnitt

Zu betonen ist, dass es zu dem Label Auspuff sehr viel mehr Beispiele gibt als zu den Labels Motor und Bremse. Anhand der Daten in Tabelle 3.1 werden im Folgenden beispielhaft die Berechnungen der *makro_precision* und der *mikro_precision* präsentiert:

- *makro_precision* : $\frac{1}{3} \left(\frac{12}{12+3} + \frac{16}{16+4} + \frac{100}{100+900} \right) = \frac{0.8+0.8+0.1}{3} = 0.567$
- *mikro_precision* : $\frac{12+16+100}{(12+16+100)+(3+4+900)} = \frac{128}{1035} = 0.124$

Die Resultate der beiden Berechnungen zeigen, dass es im Fall von ungleichmäßigen Klassenverteilungen einen großen Unterschied macht, ob man den

Makro- oder den Mikro-Durchschnitt zur Bewertung verwendet. Dementsprechend ist der Makro-Ansatz zu bevorzugen, wenn unterrepräsentierte Klassen eine wichtige Rolle spielen und nicht zu vernachlässigen sind. Andernfalls ist der Mikro-Durchschnitt empfehlenswert, da ungleichmäßige Klassenverteilungen hier stärker ins Gewicht fallen.[MKGD12]

3.1.3 Ranking

In der Praxis werden Multi-Label Problemstellungen auch oft mit Hilfe einer reellwertigen Funktion gelöst, die als Ergebnis eine Wahrscheinlichkeitsverteilung zurückgibt. In diesem Fall können Ranking Metriken verwendet werden. Dabei gilt es, die Labels nach ihrer Relevanz zu ordnen und mit der tatsächlichen Reihenfolge zu vergleichen [ZZ14].

- One Error: Berechnet den Anteil an Beispielen, bei denen das höchst bewertete Label nicht in der tatsächlichen Menge an Labels ist [ZZ14].
- Coverage: Zeigt, wie viele Einträge man in der geordneten Ranking Liste im Durchschnitt nach unten gehen muss, um alle relevanten Labels zu finden [MKGD12].
- Ranking Loss: Spiegelt den durchschnittlichen Anteil an Paaren von Labels wider, die für ein Beispiel umgekehrt geordnet sind, d. h. ein irrelevantes Label wurde höher eingestuft als ein relevantes [ZZ14].

Abschließend ist zu sagen, dass es in der Literatur einige Alternativen zu den bei BDD verwendeten Metriken gibt. Für die Bewertung der BDD-Modelle werden in dieser Arbeit jedoch ausschließlich die vom Fachbereich bei Daimler vorgegebenen Leistungsmaße (Attempt Rate, Top-3 Precision und Classes Largely Correct) verwendet.

3.2 Lösungsvarianten

Multi-Label Problemstellungen sind deutlich schwieriger zu lösen als Single-Label Klassifikationen. Eine Herausforderung ist z. B. die Größe des Output Raums, da die möglichen Label Kombinationen exponentiell mit der Anzahl an Labels zunehmen. Bei 30 verschiedenen Labels gibt es insgesamt 2^{30} bzw. über eine Milliarde mögliche Mengen an Labels. Vor diesem Hintergrund macht es Sinn, die Abhängigkeiten zwischen den Labels zu berücksichtigen. Denn Labels kommen oft zusammen vor und wenn ein spezifisches gesetzt ist, kann dies die Wahrscheinlichkeit für das Auftreten von bestimmten anderen deutlich beeinflussen.[ZZ14]

Wenn z. B. in der Mitte eines Bildes ein Kirchturm steht, ist die Wahrscheinlichkeit groß, dass auf diesem Bild auch Teile einer Stadt, das restliche Bauwerk der Kirche oder ein paar Leute zu sehen sind. Auf der anderen Seite ist es eher unwahrscheinlich, dass der Kirchturm inmitten eines Sees steht und sonst nur Bäume und Berge zu erkennen sind (der aus dem Reschensee ragende Kirchturm ist offensichtlich eine Ausnahme).

Grundsätzlich gibt es bei der Multi-Label Klassifikation drei verschiedene Lösungsvarianten: Die *Problemtransformation*, *angepasste Algorithmen* und *implizit unterstützende Algorithmen*. Abbildung 3.2 ist an [MKGD12] angelehnt und gibt einen Überblick über die verschiedenen Lösungsansätze, die im Folgenden noch näher beleuchtet werden.

3.2.1 Problemtransformation

Bei der Problemtransformation wird das Multi-Label Problem in etablierte Aufgabenstellungen wie z. B. die binäre oder die Multi-Klassen Klassifikation umgewandelt. Die Daten werden folglich an den Algorithmus angepasst. Die Methoden der Problemtransformation lassen sich in drei Gruppen einteilen: binäre Relevanz, *paarweise Zerlegung* und *Label Potenzmenge* [MKGD12].

3 Einführung in die Multi-Label Klassifikation

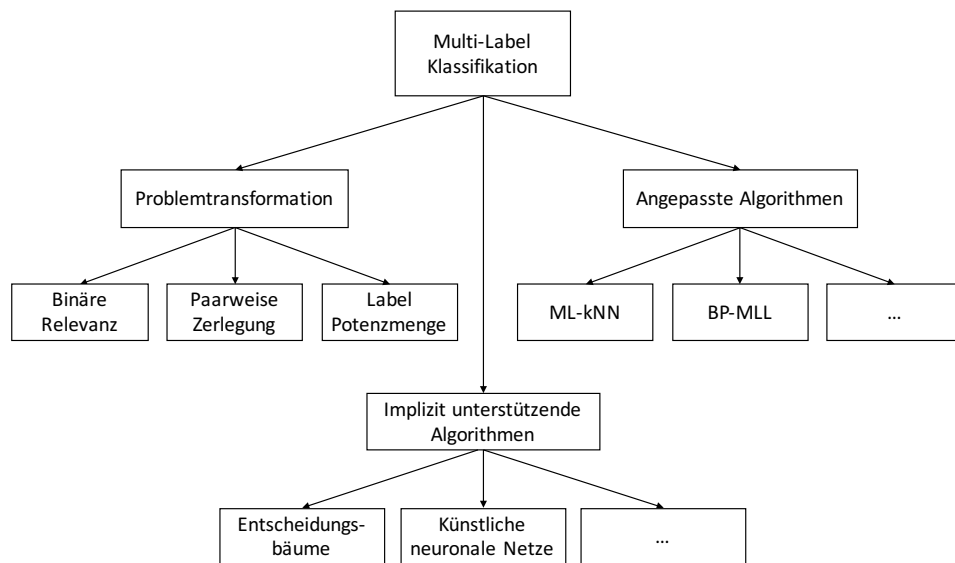


ABBILDUNG 3.2: Multi-Label Lösungsvarianten [MKGD12]

Um die Methoden anschaulich zu erklären werden die Daten in Tabelle 3.2 als Beispiel genommen. Gegeben sind vier Instanzen und drei verschiedene Schadensmöglichkeiten beim Auto (Motor, Bremse, Auspuff).

	Motor	Bremse	Auspuff
1	X		
2		X	X
3	X	X	
4			X

TABELLE 3.2: Beispieldatensatz zur den Methoden der Problemtransformation

Die binäre Relevanz Methode, die auch One-versus-Rest genannt wird, ist die aktuelle Lösungsvariante bei der BDD-Pipeline. Dabei wird für jedes der n Labels ein binärer Klassifikator trainiert. Die Multi-Label Problemstellung wird somit in n binäre Klassifikationsprobleme transformiert, die alle unabhängig voneinander sind. Im ersten Schritt wird für jeden der Klassifikatoren ein entsprechender Trainingsdatensatz erstellt. Tabelle 3.3 zeigt dies beispielhaft für die Schadensklassen Motor und Bremse.[TK06]

	Motor	\neg Motor
1	X	
2		X
3	X	
4		X

	Bremse	\neg Bremse
1		X
2	X	
3	X	
4		X

TABELLE 3.3: Beispieldatensätze zur binären Relevanz [TK06]

Nach der Vorbereitung der Datensätze wird ein binärer Algorithmus verwendet, um die verschiedenen Klassifikatoren zu trainieren. Bei der Klassifizierung eines neuen Beispiels entscheidet jeder Klassifikator, ob ein Label zu setzen ist oder nicht. Die Menge an Labels, die für eine neue Instanz vorhergesagt wird, resultiert dann aus der Vereinigung der Outputs aller Klassifikatoren. [ZZ14]

One-versus-Rest ist grundsätzlich sehr beliebt, weil es leicht zu verstehen und durchzuführen ist. Der größte Vorteil ist der geringe Rechenaufwand im Vergleich zu anderen Lösungsansätzen [MKGD12]. Im Hinblick auf die hochdimensionale Problemstellung von BDD ist dies ein entscheidendes Kriterium. Die Abhängigkeiten zwischen den Labels werden bei der binären Relevanz allerdings komplett ignoriert. Des Weiteren können ungleichmäßige Klassenverteilungen schlechte Ergebnisse verursachen, insbesondere wenn der Datensatz eine hohe Anzahl an Labels enthält [ZZ14]. Zusammenfassend kann man sagen, dass One-versus-Rest eine mögliche, jedoch keine ideale Lösungsvariante für BDD darstellt.

Bei der paarweisen Zerlegung (One-versus-One) wird ein binärer Klassifikator für jedes Paar von Labels trainiert. Bei n Labels werden demnach $\frac{n(n-1)}{2}$ binäre Klassifikatoren benötigt [Für02]. Vor dem Training wird für jeden Klassifikator ein entsprechender Trainingsdatensatz generiert, bei dem die Instanzen vom ersten Label als positive Beispiele genommen werden und die vom anderen Label als negative [ZZ14]. In Tabelle 3.4 werden die Datensätze für zwei Paare von Labels beispielhaft dargestellt.

3 Einführung in die Multi-Label Klassifikation

	Motor	Bremse
1	X	
2		X

	Bremse	Auspuff
1	X	
2		X

TABELLE 3.4: Beispieldatensätze zur paarweisen Zerlegung

Bei einem neuen Beispiel entscheidet sich dann jeder Klassifikator für eines der beiden Labels. Anschließend wird ein Mehrheitsvotum durchgeführt, d. h. die Labels werden nach der Summe ihrer Stimmen geordnet. Schließlich wird ein Ranking Algorithmus verwendet, um die Menge an Labels für ein neues Beispiel vorherzusagen [MKGD12]. Im Gegensatz zur binären Relevanz werden die Abhängigkeiten zwischen den Labels hier berücksichtigt. Ein weiterer Vorteil von One-versus-One ist, dass jeder Klassifikator nur mit den Daten trainiert wird, die für die Unterscheidung zwischen den zwei entsprechenden Labels relevant sind [Für02]. Für Algorithmen, die besser mit kleineren Datensätzen umgehen können, wie z. B. Support-Vektor-Maschinen (SVM), ist die paarweise Zerlegung oft eine gute Wahl [Gér17]. Im Fall von BDD ist One-versus-One jedoch nicht so gut geeignet, da aufgrund der hohen Anzahl an Labels enorm viele Klassifikatoren trainiert werden müssten.

Bei der Label Potenzmengen Methode wird eine Multi-Label Problemstellung in eine Multi-Klassen Klassifikation transformiert, bei der jede im Datensatz vorhandene Kombination von Labels eine Klasse repräsentiert (siehe Tabelle 3.5) [TK06].

	Motor	Auspuff	$\text{Motor} \wedge \text{Bremse}$	$\text{Bremse} \wedge \text{Auspuff}$
1	X			
2				X
3			X	
4		X		

TABELLE 3.5: Beispieldatensatz zur Label Potenzmenge [TK06]

Wie bei der paarweisen Zerlegung werden auch hier die Abhängigkeiten zwischen den Labels in Betracht gezogen. Bei hochdimensionalen Problemstellungen wie BDD kann der Raum von möglichen Label Kombinationen jedoch sehr komplex sein und einen hohen Rechenaufwand verursachen [MKGD12]. Außerdem kann eine hohe Anzahl an Labels dazu führen, dass die Methode sehr viele verschiedene Klassen generiert, zu denen es nur wenige Beispiele gibt. Dieses Problem wird durch ungleichmäßige Klassenverteilungen zunehmend verstärkt [TK06]. Vor diesem Hintergrund ist die Label Potenzmengen Methode eher nicht für BDD zu empfehlen.

Zu erwähnen ist, dass es für die Methoden der Problemtransformation bereits einige Erweiterungen gibt. Eine Variante der binären Relevanz sind z. B. die *Klassifizierungsketten* (Classifier Chains) [RPHF09]. Das Multi-Label Problem wird hier in eine Kette von binären Klassifikationsproblemen transformiert, bei der vorausgehende Vorhersagen als Features genutzt werden. Auf diese Weise können die Abhängigkeiten zwischen den Labels berücksichtigt werden [RPHF09].

Eine Erweiterung der Label Potenzmengen Methode ist z. B. *RAkEL* (Random k-labelsets). Dabei wird ein Ensemble von Klassifikatoren trainiert, die jeweils nur für eine Teilmenge aller Labels verantwortlich sind.[TV07]

3.2.2 Angepasste Algorithmen

Mittlerweile gibt es einige bekannte Algorithmen, die speziell für die Multi-Label Problemstellung angepasst wurden. *ML-kNN* ist eine Erweiterung des *k-nächste-Nachbarn*-Algorithmus, der in die Kategorie des *trägen Lernens* (Lazy Learning) einzuordnen ist [ZZ07]. Beim Lazy Learning wird die Generalisierung der Trainingsdaten bis zum Auftreten eines neuen Beispiels aufgeschoben. Demnach wird im Voraus kein Modell trainiert, wie es z. B. bei ANNs der Fall ist [Mit97]. Im ersten Schritt von ML-kNN werden für jede Instanz die k-nächsten-Nachbarn im Datensatz ermittelt. Anschließend wird die *Maximum-a-posteriori* (MAP) Methode im Zusammenhang mit statistischen Informa-

tionen von den Labels der Nachbarn angewendet. Für jedes Label wird z. B. die Anzahl an Nachbarn berechnet, die mit dem jeweiligen Label assoziiert sind [ZZ07]. ML-kNN kombiniert die Vorteile von Lazy Learning mit denen des Bayes-Ansatzes. Darüber hinaus wird das Problem von ungleichmäßigen Klassenverteilungen durch die geschätzten a-priori-Wahrscheinlichkeiten für jedes Label vermindert. Die Abhängigkeiten zwischen den Labels werden jedoch nicht berücksichtigt [ZZ14].

BP-MLL ist eine Anpassung des *Backpropagation* Algorithmus (siehe Kapitel 4.2.1), der standardmäßig zur Optimierung von ANNs verwendet wird [ZZ06]. Grundsätzlich ist auch die Verwendung des klassischen Backpropagation Algorithmus [RHW86] möglich, allerdings werden die Abhängigkeiten zwischen den Labels dabei nicht berücksichtigt. Daher wurde bei BP-MLL eine neue *Fehlerfunktion* definiert, die stark an die Ranking Loss Metrik (siehe 3.1.3) angelehnt ist. Im Prinzip geht es darum, dass Labels, die für eine Instanz relevant sind, auch tatsächlich höher eingestuft werden als irrelevante [ZZ06]. Ranking Loss wurde auch schon bei einer Multi-Label Erweiterung von SVMs als Fehlerfunktion verwendet [EW02]. Eine bekannte Erweiterung von Entscheidungsbäumen für den Multi-Label Kontext ist z. B. *ML-C4.5* [CK01].

Im Hinblick auf BDD stellen die angepassten Algorithmen mögliche Alternativen dar, die in dieser Arbeit aber nicht genauer evaluiert wurden. Grundsätzlich ist zu beachten, dass eine stärkere Berücksichtigung der Multi-Label Problemstellung bzw. der Label-Korrelationen praktisch immer mit einem erhöhten Rechenaufwand einhergeht [ZZ14].

3.2.3 Implizit unterstützende Algorithmen

Einige Algorithmen können Multi-Label Problemstellungen implizit unterstützen, wie z. B. Entscheidungsbäume oder ANNs. Im Fall von ANNs werden die verschiedenen Labels von den Output Neuronen repräsentiert und die Ergebnisse in Form von Wahrscheinlichkeiten angegeben. Mit Hilfe eines definierten

Schwellenwerts kann dann eine Menge an Labels für ein beliebiges Beispiel bestimmt werden. Abbildung 3.3 zeigt ein ANN mit einer *Eingabeschicht* (Input Layer), einer *verborgenen Schicht* (Hidden Layer) und einer *Ausgabeschicht* (Output Layer) mit drei Output *Neuronen*. Wenn der Schwellenwert bei der Prognose für ein Label z. B. bei 30 Prozent Wahrscheinlichkeit liegt, würden die vom linken und mittleren Output Neuron repräsentierten Labels vorhergesagt werden.

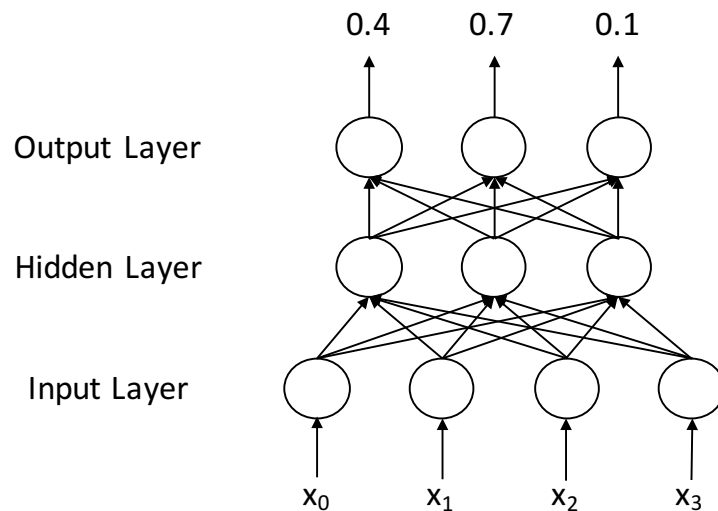


ABBILDUNG 3.3: ANN mit Wahrscheinlichkeiten als Outputs

Im Rahmen dieser Arbeit wird diese Lösungsvariante mit ANNs für BDD genauer untersucht. Die Input Neuronen repräsentieren dabei die Fehlercodes aus den Steuergeräten und die Output Neuronen die möglichen Reparaturcodes. ANNs können allgemein gut mit komplexen Datensätzen umgehen und sind in der Lage, die Labels mit den entsprechenden Wahrscheinlichkeiten vorherzusagen. Des Weiteren stellen sie einen robusten Ansatz gegenüber veräuschten Daten dar [Mit97]. Je nach Aufgabenstellung und Datensatz muss bei ANNs mit längeren Trainingszeiten gerechnet werden, was im Hinblick auf BDD kein Problem ist. Die Anwendung des Modells funktioniert normalerweise sehr schnell. Zu beachten, ist dass die Ergebnisse von ANNs praktisch nicht nachvollziehbar sind, was z. B. bei medizinischen Diagnosen problematisch ist. Im Fall von BDD spielt dieser Faktor keine entscheidende Rolle.[Mit97]

4 Grundlagen künstlicher neuronaler Netze

ANNs sind keine neue Erfindung, sondern wurden bereits im Jahr 1943 für ein vereinfachtes Modell von biologischen Neuronen vorgestellt [MP43]. Etwa 15 Jahre später wurde die Architektur des *Perzeptrons* eingeführt [Ros58], das den Aufbau von modernen ANNs maßgeblich beeinflusst hat.

Ein Perzeptron (siehe Abbildung 4.1) beinhaltet ein einzelnes künstliches Neuron, bei dem jeder Input mit einem Gewicht verbunden ist. Außerdem wird in der Regel noch ein Bias-Term mit einem konstanten Wert verwendet (z.B. $x_0 = 1$), der von keinem Input abhängt. Das künstliche Neuron berechnet im ersten Schritt eine gewichtete Summe aus allen Inputs und dem Bias-Term. Das Ergebnis wird dann an eine Schwellenwertfunktion weitergegeben. Wenn das Resultat über dem Schwellenwert liegt, wird der Wert 1 ausgegeben, ansonsten der Wert -1.[Mit97]

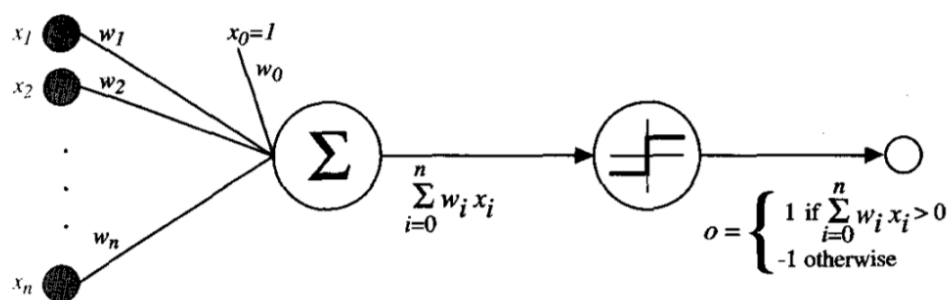


ABBILDUNG 4.1: Aufbau eines Perzeptrons [Mit97]

Ein Perzeptron mit einem künstlichen Neuron kann für die binäre Klassifikation von linearen Problemstellungen eingesetzt werden und ist in der Lage, alle primitiven booleschen Funktionen (AND, OR etc.) abzubilden [Bis06]. In den 1960er Jahren wurde jedoch festgestellt, dass Perzeptronen einige gravierende Schwächen haben und z.B. nicht in der Lage sind, einfache nicht-lineare Probleme wie ein XOR zu modellieren. Diese Hürde wurde mit der Einführung des Multilayer-Perzeptrons (MLP) überwunden.[Gér17]

4.1 Multilayer-Perzeptron

Bei einem MLP werden Perzeptronen in mehreren Schichten verteilt und komplett miteinander vernetzt, d. h. jedes Perzeptron ist mit allen Neuronen der nächsten Schicht verbunden (siehe Abbildung 3.3). Auf diese Weise können die Berechnungen vorwärts durch das Netz weitergeleitet werden [Mit97]. Ein MLP besteht aus einem Input Layer, mindestens einem Hidden Layer und einem Output Layer. Der Input Layer gibt die Eingangswerte an die Perzeptronen des ersten Hidden Layers weiter. Darüber hinaus hat der Input Layer keine Funktion mehr. In den Hidden Layers werden die Zusammenhänge in den Daten modelliert. Diese können bei einem MLP auch nicht-linear sein. Der Output Layer gibt dann das Ergebnis aller Berechnungen im Netz aus.[Gér17]

Bei der Modellierung der nicht-linearen Zusammenhänge und der Ausgabe des Output Layers spielt die Verwendung von *Aktivierungsfunktionen* eine entscheidende Rolle. *Rectified Linear Unit* (ReLU) ist z.B. eine beliebte Aktivierungsfunktion für die Hidden Layers, die im Vergleich zu anderen schnell berechenbar ist und in der Praxis gut funktioniert [Cho18]. Zu betonen ist, dass ReLU im negativen Bereich immer den Wert 0 zurückgibt, was im Hinblick auf das Training von MLPs problematisch sein kann (siehe Abschnitt 4.4.2). Die Formel von ReLU ist $relu(z) = \max(0, z)$ (siehe Abbildung 4.2), wobei z den Input für ein Neuron repräsentiert.[GBCB16]

Eine weitere bekannte Aktivierungsfunktion ist die *Sigmoid-Funktion*, die wie folgt definiert ist: $sig(z) = \frac{1}{1+e^{-z}}$. Sie ist S-förmig und liefert einen Wert

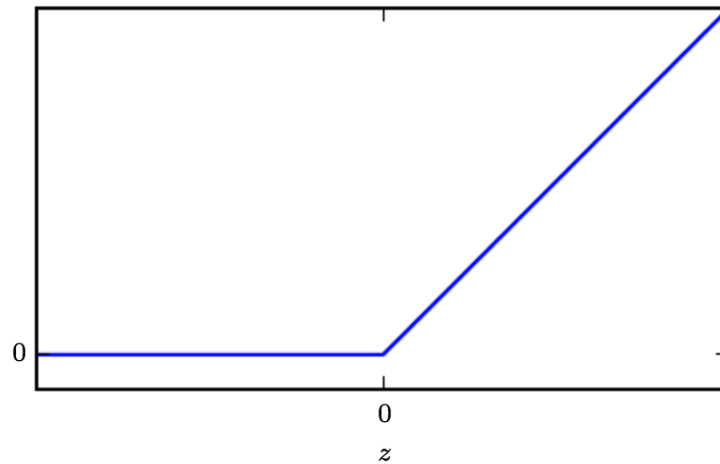


ABBILDUNG 4.2: ReLU Funktion [GBCB16]

zwischen 0 und 1 (siehe Abbildung 4.3), was insbesondere bei der Angabe von Wahrscheinlichkeiten im Output Layer praktisch ist. Im Hinblick auf eine Wahrscheinlichkeitsverteilung summieren sich die einzelnen Werte der Sigmoid-Funktion nicht auf den Wert 1 [Cho18]. Für Multi-Label Problemstellungen, bei denen ggf. mehrere Labels mit einer hohen Wahrscheinlichkeit angegeben werden müssen, ist die Sigmoid-Funktion demnach gut geeignet.

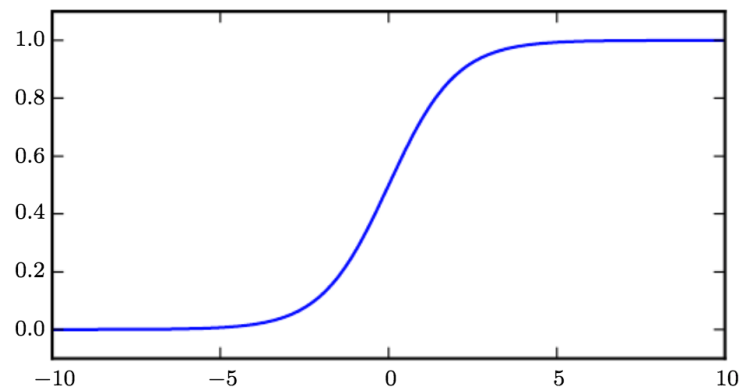


ABBILDUNG 4.3: Sigmoid-Funktion [GBCB16]

Wenn eine Wahrscheinlichkeitsverteilung für diskrete Klassen benötigt wird, wie z.B. bei der Multi-Klassen-Klassifikation, ist die *Softmax-Funktion* die richtige Wahl. Hier summieren sich die einzelnen Wahrscheinlichkeiten auf den

Wert 1. Die Formel der Softmax Funktion ist $softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1} e^{z_j}}$, wobei i und j beide einen Index über die Output Units darstellen [GBCB16].

Für den Aufbau eines MLPs sind neben den Aktivierungsfunktionen u. a. auch noch die Anzahl an Perzeptronen in den verschiedenen Schichten (Input-, Hidden- und Output-Units) sowie die Anzahl an Hidden Layers festzulegen. Die Input und Output Units können relativ leicht bestimmt werden, da sie vom Feature Vektor und der jeweiligen Problemstellung vorgegeben werden. Bei der Multi-Label Klassifikation repräsentieren die Output Units z.B. die möglichen Labels.

Bei den Hidden Layers und Hidden Units ist eine korrekte Auswahl allerdings nicht so offensichtlich. Sie bestimmen die Anzahl der Parameter und somit auch die Komplexität eines Modells. Da eine unangemessene Anzahl an Hidden Layers und Hidden Units zu einer Über- oder Unteranpassung des Modells führen kann, gilt es diese beiden Werte systematisch zu evaluieren [Cho18].

Zusammenfassend kann man sagen, dass die Forschung an ANNs mit der Einführung des MLPs einen großen Sprung nach vorne gemacht hat, da durch diesen Aufbau auch nicht-lineare Zusammenhänge in den Daten modelliert werden können. Theoretisch kann sogar jede noch so komplexe Funktion von einem Netz mit zwei Hidden Layers abgebildet werden [Cyb89].

Eine große Herausforderung stellte jedoch lange Zeit das Training von MLPs dar. Dieses soll im Folgenden näher beleuchtet werden.

4.2 Training

Das Training von einem ANN erfolgt durch die iterative Anpassung seiner Parameter, d. h. der Gewichte aller Neuronen. Auf diese Weise wird die Abbildung der Input Daten auf den erwarteten Output ermöglicht [Mit97]. Die Gewichte eines Netzes werden zu Beginn mit zufälligen Werten initialisiert, die keine sinnvolle Repräsentation der Problemstellung darstellen. Beim Durchlauf

der ersten Trainingsinstanzen wird das Ergebnis des Netzes also in der Regel stark von der gewünschten Ausgabe abweichen [Cho18].

Da ein MLP oft aus Millionen von Neuronen besteht, ist eine manuelle Anpassung der Gewichte praktisch unmöglich. Für einen Durchbruch beim Training sorgte der in den 1980er Jahren eingeführte Backpropagation Algorithmus [RHW86], der auf dem *Gradientenverfahren* basiert.

4.2.1 Backpropagation

Backpropagation beinhaltet im Prinzip drei Schritte, die wie in einer Art Schleife wiederholt werden: Die *Vorwärtspropagierung*, die Verwendung einer Fehlerfunktion und die *Rückwärtspropagierung* (siehe Abbildung 4.4).

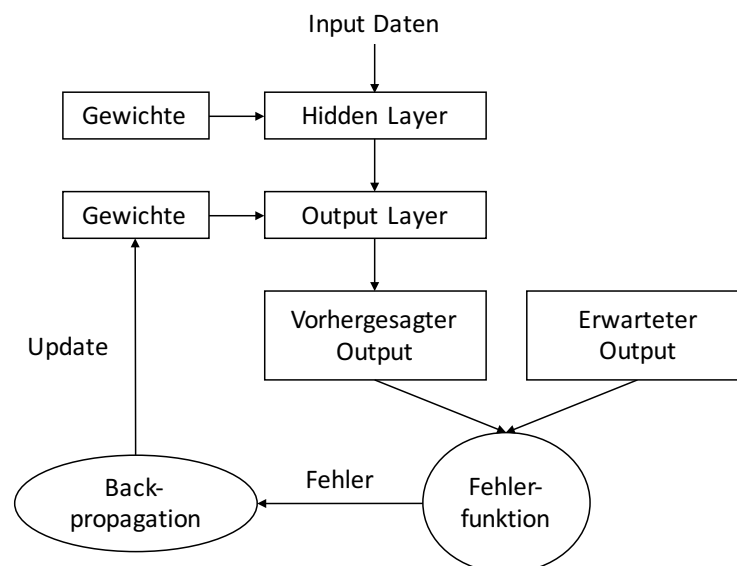


ABBILDUNG 4.4: Skizzierung des Backpropagation Algorithmus [Cho18]

Im ersten Schritt wird eine kleine Menge von Beispielen (Mini-Batch) vom Trainingsdatensatz genommen und vorwärts durch das Netz propagiert. Im zweiten Schritt wird die Differenz zwischen den Resultaten des Netzes und den erwarteten Outputs berechnet. Hierfür wird eine Fehlerfunktion definiert, die als Ergebnis die entsprechende Abweichung zurückgibt. Im dritten Schritt

4 Grundlagen künstlicher neuronaler Netze

wird dann der berechnete Fehler als Feedback Signal genutzt und rückwärts durch jeden Layer propagiert. Dabei werden die Gewichte aller Neuronen so angepasst, dass der Fehler bei jedem Durchlauf der Schleife ein bisschen verringert wird.[Cho18]

Die Anpassung der Gewichte stellt den kompliziertesten Teil des Backpropagation Algorithmus dar und wird mit Hilfe des Gradientenverfahrens gelöst. Konkret werden die Gradienten der Fehlerfunktion mit Bezug auf den Parametervektor berechnet und in Richtung des steilsten Abstiegs optimiert. Der Fehler des Output Layers wird dabei proportional auf die Gewichte der vorausgehenden Schichten verteilt, so dass das Feedback Signal auch beim ersten Hidden Layer im Netz ankommt. Auf diese Weise wird für jedes Neuron eine individuelle Abweichung berechnet, die dann zur Anpassung der Gewichte verwendet wird.[Mit97]

Durch die Minimierung der Fehlerfunktion lernt das Modell die Input Daten auf die erwarteten Outputs abzubilden. Die Auswahl der Fehlerfunktion ist folglich eine wichtige Design Entscheidung beim Aufbau eines MLPs. Beliebte ist z.B. die *quadratische Fehlerfunktion* $E = \frac{1}{2} \sum_{i=1}^n (t_i - o_i)^2$, wobei E für den Fehler, n für die Anzahl an Trainingsinstanzen, t für den erwarteten Output und o für die Vorhersage des ANNs steht [Mit97].

Im Zusammenhang mit der Optimierung von Wahrscheinlichkeitsverteilungen wird gerne die *Kreuzentropie* als Fehlerfunktion eingesetzt. Die Kreuzentropie ist eine bekannte Metrik aus der Informationstheorie, mit der man allgemein den Unterschied zwischen Wahrscheinlichkeitsverteilungen berechnen kann [Cho18].

Die Kreuzentropie ist wie folgt definiert: $-\sum_{i=1}^{|D|} t_i \log o_i + (1 - t_i) \log(1 - o_i)$, wobei t_i für den erwarteten Output und o_i für die Vorhersage des ANNs beim i -ten Beispiel vom Datensatz D steht [Mit97].

4.2.2 Gradientenverfahren

Das Gradientenverfahren ist ein in der Praxis häufig eingesetzter Optimierungsalgorithmus, der grundsätzlich gut geeignet ist, um in großen Hypothesenräumen nach optimalen Lösungen zu suchen. Grundvoraussetzung ist, dass die verwendete Fehlerfunktion differenzierbar ist [Mit97].

Ein entscheidender Parameter beim Training mit dem Gradientenverfahren ist die Lernrate. Sie bestimmt die Größe der Schritte in Richtung des steilsten Abstiegs bzw. in die entgegengesetzte Richtung des Gradienten. Wenn sie zu klein ist, braucht der Algorithmus sehr viele Iterationen, um das Minimum zu finden. Wenn sie zu hoch ist, kann es sein, dass die optimale Lösung bei der Suche übersprungen wird [GBCB16].

Bei der Anwendung des Gradientenverfahrens gibt es auch ein paar Schwierigkeiten. Zum einen braucht das Verfahren oft tausende von Iterationen bis es konvergiert und kann daher sehr lange dauern. Zum anderen ist es möglich, dass der Algorithmus in lokalen Minima stecken bleibt und es nach dem Training noch Potential zur Optimierung gibt [Mit97].

Abbildung 4.5 zeigt eine Fehlerfunktion mit zwei lokalen Minima, bei dem eines deutlich vom globalen Minimum entfernt ist und daher vermieden werden sollte. Das andere Minimum stellt hingegen eine akzeptable Lösung dar, da der Wert der Fehlerfunktion hier fast so klein ist wie beim globalen Minimum [GBCB16].

Das Gradientenverfahren hat verschiedene Varianten. So kann z.B zwischen dem *Batch*-, dem *Mini-Batch* und dem *stochastischen Gradientenverfahren* (Stochastic Gradient Descent, kurz SGD) unterschieden werden [Gér17]. Im Batch-Modus wird der ganze Trainingsdatensatz genutzt, um die Gradienten zu berechnen und die Anpassung der Gewichte vorzunehmen. Wenn jedes Beispiel eines Datensatzes einmal für das Training genutzt wurde, spricht man auch von einer Epoche. Beim Batch-Gradientenverfahren werden die Aktualisierungen der Gewichte folglich immer am Ende einer Epoche durchgeführt, was insbesondere bei großen Datensätzen sehr lange dauern kann [Gér17].

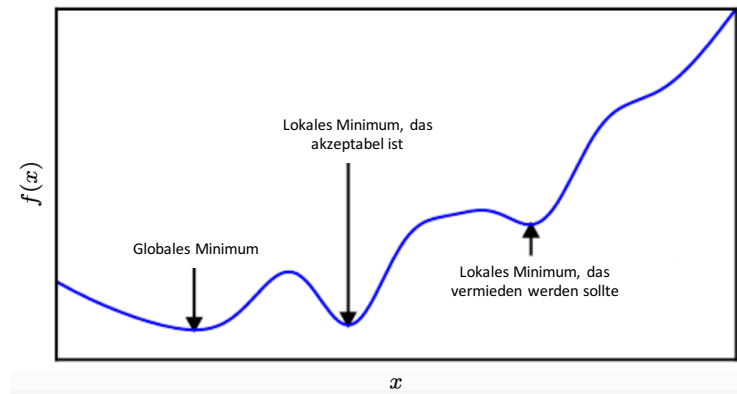


ABBILDUNG 4.5: Problem von lokalen Minima beim Gradientenverfahren [GBCB16]

Bei SGD werden die Gewichte hingegen nach jedem Beispiel angepasst. Die jeweilige Instanz wird dabei immer zufällig vom Trainingsdatensatz gewählt, wobei ggf. einige Beispiele mehrmals und andere gar nicht berücksichtigt werden [Cho18]. Ein Vorteil von SGD ist, dass es im Vergleich zum Batch-Modus weniger Rechenzeit benötigt. Allerdings ist ein Gradient, der anhand einer Instanz berechnet wird, generell nicht so stabil und kann daher auch ungenaue Anpassungen verursachen [GBCB16]. Die Suche nach der optimalen Lösung kann demnach erschwert werden, wenn immer wieder auf der Fehlerfunktion hin und her gesprungen wird. Auf der anderen Seite hat die Praxis gezeigt, dass SGD dadurch weniger anfällig ist, in lokalen Minima stecken zu bleiben [Mit97].

Ein Kompromiss aus dem Batch- und dem stochastischen Gradientenverfahren ist die Mini-Batch Variante. Dabei wird der Datensatz in eine Reihe von Mini-Batches geteilt, wobei die Anzahl an Beispielen pro Gruppe (Batch Size) oft zwischen 32 und 512 liegt [Gér17]. Wenn ein Trainingsdatensatz z.B. 1024 Instanzen beinhaltet und die Batch Size 256 beträgt, dann benötigt der Algorithmus vier Iterationen für eine Epoche.

Nach dem Durchlauf einer Mini-Batch wird der Fehler berechnet und die Gewichte werden angepasst. Da die Gradienten stabiler sind, wenn mehrere Beispiele berücksichtigt werden, konvergiert die Fehlerfunktion bei der Mini-Batch Variante in der Regel besser als bei SGD [Gér17]. Abbildung 4.6 verdeutlicht

die unterschiedliche Konvergenz der drei vorgestellten Varianten des Gradientenverfahrens in einem beispielhaften Hypothesenraum.

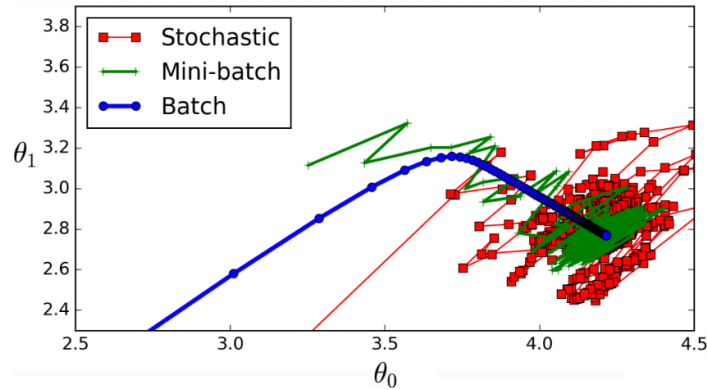


ABBILDUNG 4.6: Unterschiedliche Konvergenz des Batch-, Mini-Batch und stochastischen Gradientenverfahrens [Gér17]

Ein weiterer Vorteil des Mini-Batch Gradientenverfahrens im Vergleich zu SGD ist, dass die Rechenzeit durch eine Vektorisierung der Matrix Operationen verkürzt werden kann. Dieser Effekt wird durch den Einsatz von Grafikprozessoren (GPUs) zunehmend verstärkt [Gér17].

Zu erwähnen ist, dass es mittlerweile bereits einige optimierte Varianten des Gradientenverfahrens gibt, wie z.B. *Adam* (Adaptive Moment Estimation) [KB14], die eine schnellere Konvergenz beim Training ermöglichen.

4.3 Regularisierung

Ein fundamentales Problem im Maschinellen Lernen ist das Bias-Varianz Dilemma (siehe Kapitel 2.4). Im Zusammenhang mit ANNs gibt es mittlerweile einige Strategien, um eine möglichst gute Generalisierung auf unbekannte Daten zu ermöglichen. Die Methoden der *Regularisierung* passen den Lernalgorithmus so an, dass der Testfehler und nicht der Trainingsfehler bestmöglich reduziert wird [GBCB16]. Eine wichtige Frage ist z.B., wann das Training mit dem Backpropagation Algorithmus abgebrochen wird.

4.3.1 Early Stopping und k-fache Kreuzvalidierung

Um eine Unter- bzw. Überanpassung des Modells zu verhindern, gilt es das Training rechtzeitig abubrechen. Wenn der Algorithmus zu früh abgebrochen wird, kann es sein, dass wichtige Zusammenhänge in den Daten außer Acht gelassen werden. Wenn die Fehlerfunktion hingegen zu sehr auf die Trainingsdaten optimiert wird, lernt das Modell zu spezifische Muster [Cho18].

Beide Fälle haben negative Auswirkungen auf die Leistung bei der Generalisierung und sollten daher unbedingt vermieden werden. In diesem Zusammenhang wird gerne die Methode *Frühes Stoppen* (Early Stopping) verwendet. Beim Early Stopping kommt neben dem Trainingsdatensatz auch noch ein Validierungsdatensatz zum Einsatz. Dieser kann bereits während des Trainings zur Evaluation des Modells genutzt werden. Bei der Anwendung des Algorithmus können folglich zwei Lernkurven dargestellt werden: Der Trainings- und der Validierungsfehler (siehe Abbildung 4.7). [Mit97]

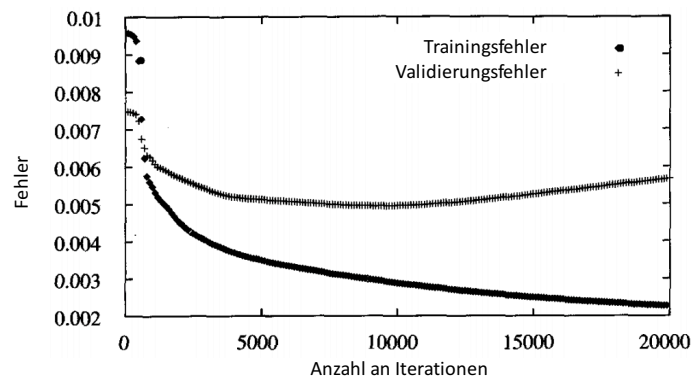


ABBILDUNG 4.7: Verlauf des Trainings- und Validierungsfehlers in Bezug auf die Anzahl an Iterationen [Mit97]

Eine Überanpassung tritt dann auf, wenn die Differenz zwischen dem Trainings- und Validierungsfehler zu groß wird. In Abbildung 4.7 sieht man, dass der Validierungsfehler nach knapp 10000 Iterationen leicht ansteigt, wohingegen sich der Trainingsfehler weiter seinem Minimum annähert. Grundsätzlich ist es empfehlenswert, das Training abubrechen, wenn der Validierungsfehler am

niedrigsten ist, da er den besten Indikator für die Leistung der Vorhersagen auf unbekannte Daten darstellt [Mit97].

Der Verlauf der Lernkurve ist dabei generell nicht klar. So kann es sein, dass der Validierungsfehler erst ansteigt und nach einer gewissen Anzahl an Iterationen wieder beginnt zu fallen und ein neues Minimum erreicht. Daher wird in der Praxis oft ein zusätzliches Modell gespeichert, das die bisher besten Gewichte in Bezug auf den Validierungsfehler enthält. Für den Fall, dass weiter trainiert und der Validierungsfehler nicht mehr kleiner wird, kann dann auf das Modell mit den besten Parametern zurückgegriffen werden [GBCB16].

Die Dauer des Trainings bzw. die Anzahl an Epochen wird folglich von der Early Stopping Methode bestimmt. Ein wichtiger Parameter ist dabei die *Geduld* (Patience). Die Patience gibt die Anzahl an Epochen vor, für die mindestens weiter trainiert wird, sobald der Validierungsfehler nicht mehr fällt. Wenn der Validierungsfehler nach der vorgegebenen Anzahl an Epochen nicht mehr weiter gesunken ist, wird der Algorithmus abgebrochen [Mit97].

Zur Durchführung von Early Stopping muss der verwendete Datensatz in ein Trainings-, Validierungs- und Testdatensatz geteilt werden. Dabei empfiehlt es sich, erstmal in Trainings- und Testdaten zu teilen und den Testdatensatz ausschließlich für die finale Bewertung des fertig trainierten Modells zu verwenden [Gér17].

Für das Training kann der Trainingsdatensatz dann noch in Trainings- und Validierungsdaten getrennt werden. Damit die Ergebnisse statistisch repräsentativ sind und nicht zu sehr von einer bestimmten Partition abhängen, wird z.B. gerne die Methode der *k-fachen Kreuzvalidierung* eingesetzt [Cho18]. Dabei werden die Trainingsdaten in k Teilmengen gleicher Größe aufgeteilt. Anschließend wird mit $k - 1$ Partitionen trainiert und mit der verbleibenden evaluiert [Cho18]. Abbildung 4.8 verdeutlicht das Konzept am Beispiel einer 3-fachen Kreuzvalidierung.

Wenn jede Partition einmal zur Evaluation verwendet wurde, wird der Durchschnitt aus den k Ergebnissen berechnet. Dieser stellt dann das Endergebnis

4 Grundlagen künstlicher neuronaler Netze

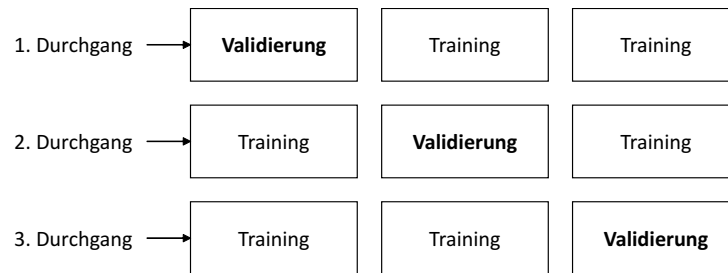


ABBILDUNG 4.8: Konzept der 3-fachen Kreuzvalidierung [Cho18]

der k -fachen Kreuzvalidierung dar, das u. a. auch für die Optimierung der Hyperparameter eines Modells (Lernrate, Batch Size etc.) nützlich ist [Cho18].

Ein großer Vorteil der k -fachen Kreuzvalidierung ist, dass keine Daten für die Validierung verloren gehen, da diese letztendlich auch für das finale Training des ausgewählten Modells genutzt werden können. Zu beachten ist, dass auf keinen Fall die Testdaten zur Validierung eines Modells verwendet werden dürfen. Dies hätte zur Folge, dass die Ergebnisse der finalen Bewertung grundsätzlich zu optimistisch ausfallen würden [Gér17].


4.3.2 Dropout

Eine weitere beliebte Methode, um die Vorhersagen auf unbekannte Daten zu optimieren ist *Dropout*. Dabei wird während des Trainings ein gewisser Anteil an Neuronen abgeschaltet, d. h. auf den Wert 0 gesetzt (siehe Abbildung 4.9) [SHK⁺14].

Der Anteil an Neuronen bewegt sich dabei normalerweise zwischen 0.2 und 0.5 und die Auswahl der Neuronen erfolgt zufällig. Auf diese Weise fokussiert sich das Netz beim Training eher auf abstraktere Konzepte in den Daten. Es wird also verhindert, dass sich gewisse Neuronen zu sehr auf einzelne Muster spezialisieren [Cho18].

Wenn das Training abgeschlossen ist, werden die abgeschalteten Neuronen wieder aktiviert. Bei der Verwendung des Modells stehen also alle Neuronen zur

Verfügung. Auf diese Weise kann die Leistung eines ANNs deutlich verbessert werden. Das liegt daran, dass es durch Dropout oft robuster wird und im Hinblick auf unbekannte Beispiele besser generalisiert [SHK⁺14].



0.5	0.1	0.7
1.7	0.3	0.6
1.2	0.1	0.0
1.3	0.2	0.5

0.5	0.0	0.7
1.7	0.0	0.6
0.0	0.1	0.0
1.3	0.0	0.0

ABBILDUNG 4.9: Anwendung einer Dropout-Rate von 0.5 bei einer Aktivierungsmatrix [Cho18]

4.4 Weitere Optimierungsstrategien

Einige Problemstellungen können aufgrund ihrer Komplexität am besten mit tiefen neuronalen Netzen gelöst werden. Bis in die späten 2000er Jahre war das Training von solchen Netzen allerdings praktisch nicht umsetzbar. Das zentrale Problem war hier, dass der Fehler beim Backpropagation Algorithmus nicht durch die vielen verborgenen Schichten geleitet werden konnte und das Feedback Signal nicht bei den vordersten Hidden Layers ankam [Cho18].

Dementsprechend konnten die Gewichte von diesen Schichten auch nicht angepasst werden, was das Training grundlegend beeinträchtigte. Neben dem Problem der *verschwindenden Gradienten* gibt es auch noch das der *explodierenden Gradienten*. Wie der Name schon sagt, werden die Gradienten hier immer größer, so dass die Gewichte in den verschiedenen Schichten in einem unverhältnismäßigen Ausmaß angepasst werden. Die Konsequenz ist, dass der Algorithmus die Orientierung verliert und nicht konvergiert [GBCB16].

4.4.1 Initialisierung der Gewichte

In [GB10] wurde dargelegt, dass das Problem von verschwindenden und explodierenden Gradienten u. a. mit der Initialisierung der Gewichte und der

Aktivierungsfunktion zusammenhängt. Zu der Zeit als [GB10] veröffentlicht wurde, war es üblich die Gewichte zufällig anhand einer Normalverteilung mit Mittelwert 0 und Standardabweichung 1 zu initiieren und als Aktivierungsfunktion die Sigmoid-Funktion zu verwenden. Diese Kombination sorgt dafür, dass die Varianz der Outputs in jedem Layer deutlich größer ist als die Varianz der Inputs. Bei der Vorwärtspropagierung durch das Netz wird die Varianz folglich immer größer, so dass die Sigmoid-Funktion in den hinteren Schichten des Netzes nahe ihrer Grenzwerte aktiviert wird [Gér17].

Dies stellt im Hinblick auf den Backpropagation Algorithmus ein echtes Problem dar, da die Gradienten an den Grenzwerten der Sigmoid-Funktion fast den Wert 0 haben. Da das Feedback Signal folglich schon zu Beginn der Rückwärtspropagierung sehr schwach ist, kommt es bei den vorderen Layers praktisch nicht mehr an [Gér17].

Um dies zu vermeiden, muss die Varianz der Outputs in jedem Layer der Varianz der Inputs entsprechen, und das sowohl bei der Vorwärts- als auch bei der Rückwärtspropagierung durch das Netz. Da dies allerdings nur möglich ist, wenn eine Schicht dieselbe Anzahl an Input und Output Verbindungen hat, wird in [GB10] die sogenannte *Xavier Initialisierung* der Gewichte vorgeschlagen.

Für die gleichmäßige Verteilung der Xavier Initialisierung wird die Formel $r = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$ verwendet, wobei n_{inputs} für die Input Verbindungen und $n_{outputs}$ für die Output Verbindungen der initialisierten Schicht stehen. Diese Lösung funktioniert sehr gut in der Praxis und kann das Training deutlich beschleunigen [Gér17].

4.4.2 Varianten von ReLUs

Wie gerade beschrieben, ist das Problem der verschwindenden und explodierenden Gradienten zum Teil auch auf die Sigmoid-Funktion zurückzuführen. Heutzutage wird in der Regel die ReLU Aktivierungsfunktion eingesetzt [GBCB16]. Zum einen können ReLUs schneller berechnet werden und zum anderen flacht

die Funktion im positiven Bereich nicht ab und verursacht dort folglich auch keine verschwindend kleinen Gradienten [Gér17].

Allerdings kann es passieren, dass einige ReLUs während dem Training nur noch den Wert 0 ausgeben. Das liegt daran, dass der Gradient immer 0 ist, sobald die Funktion einmal im negativen Bereich aktiviert wurde. Aus diesem Grund gibt es mittlerweile auch einige Varianten von ReLUs, wie z. B. *Leaky ReLU* oder die parametrisierte Leaky ReLU (PReLU) [XWCL15].

Leaky ReLU ist durch $\text{leaky_relu}_a(z) = \max(az, z)$ definiert, wobei z den Input für ein Neuron darstellt und a die Steigung angibt, mit der die Funktion im negativen Bereich abfällt [Gér17]. Auf diese Weise kann verhindert werden, dass im Netz sehr viele Neuronen nur noch den Wert 0 ausgeben. Bei PReLU ist a kein fest definierter Hyperparameter, sondern wird beim Training vom Backpropagation Algorithmus angepasst [XWCL15]. Abbildung 4.10 zeigt die Funktion von Leaky Relu bzw. PReLU im Vergleich zur normalen ReLU Funktion.

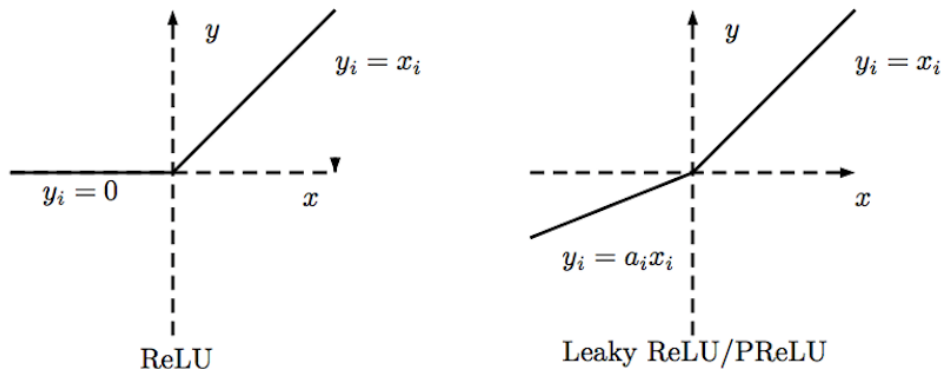


ABBILDUNG 4.10: Varianten der ReLU Aktivierungsfunktion [XWCL15]

4.4.3 Batch-Normalisierung

Das Problem von verschwindenden und explodierenden Gradienten kann durch die Xavier Initialisierung der Gewichte und den Einsatz von ReLUs als Aktivierungsfunktion eingeschränkt werden. Allerdings kann dieses Problem auch

während des Trainings wieder auftreten. Um dies zu verhindern, kann die Methode der *Batch-Normalisierung* eingesetzt werden [Gér17].

Bei der Batch-Normalisierung werden die Inputs jedes Layers vor der Aktivierungsfunktion normalisiert, skaliert und verschoben [IS15]. Die Praxis zeigt, dass diese Methode sehr gute Ergebnisse mit sich bringt und das Training deutlich beschleunigen kann. Darüber hinaus hat sie eine regularisierende Wirkung auf das Netz. Zu beachten ist jedoch, dass die Batch-Normalisierung für einen zusätzlichen Rechenaufwand sorgt.[Gér17]

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

Den Forschungsschwerpunkt dieser Arbeit bildet die Frage, ob die Reparaturprognosen von BDD durch den Einsatz von ANNs verbessert werden können. Hierfür wurde eine Anwendung in Python entwickelt, die sich im Hinblick auf die BDD-Pipeline (siehe Abbildung 1.2) auf die beiden mittleren Schritte konzentriert, nämlich die Datenaufbereitung und das Training. Zu Beginn dieses Kapitels werden die wichtigsten Bibliotheken und Programmiergerüste (Frameworks) präsentiert, die bei der Programmierung verwendet wurden. Anschließend wird die Auswahl der vorhersagbaren Reparaturcodes erläutert und es wird das Training anhand der verschiedenen Netzwerkarchitekturen vorgestellt. Des Weiteren wird näher auf die Hyperparameter Optimierung und die Modellselektion eingegangen. Schließlich werden die Ergebnisse der finalen Evaluation präsentiert und diskutiert.

5.1 Verwendete Frameworks

Um zu untersuchen, ob die Reparaturprognosen von BDD durch den Einsatz von ANNs verbessert werden können, wurde für diese Arbeit eine Python Anwendung entwickelt, die sich auf die Datenaufbereitung und das Training der

BDD-Modelle fokussiert. Dabei wurde u.a. *scikit-learn*, *Keras* und *TensorFlow* genutzt.

Scikit-learn ist eine öffentlich zugängliche (Open Source) Bibliothek, die einen leichten Einstieg ins Maschinelle Lernen ermöglicht. Sie ist in Python geschrieben und umfasst eine große Auswahl an Maschinellen Lernverfahren für überwachtes und unüberwachtes Lernen. Bei der Entwicklung der Bibliothek wurde vor allem auf die Benutzerfreundlichkeit, die Leistungsfähigkeit, die Dokumentation und auf eine konsistente Programmierschnittstelle (Application Programming Interface, kurz API) geachtet.[PVG⁺11]

Keras ist eine API höherer Abstraktion für die Entwicklung von ANNs [C⁺18]. Das Framework ist in Python geschrieben und modular aufgebaut. Bei der Entwicklung wurde darauf geachtet, dass die API benutzerfreundlich ist und schnelle Experimente ermöglicht. Keras unterstützt alle möglichen Netzwerkarchitekturen und ist daher auch bestens für den Deep Learning Kontext geeignet [Cho18]. Für die grundlegenden Berechnungen im Zusammenhang mit ANNs, wie z. B. Matrix Multiplikationen, nutzt Keras andere spezialisierte Frameworks. Zur Ausführung bzw. als sogenannte *Backend Engine* kann z. B. *Theano* [ARAA⁺16] oder *TensorFlow* [AAB⁺16] verwendet werden.

TensorFlow ist eine Open Source API, mit der man Maschinelle Lernverfahren entwickeln und ausführen kann [AAB⁺16]. Die Software wurde ursprünglich von Google entwickelt und zeichnet sich vor allem durch ihre Leistungsfähigkeit und Skalierbarkeit aus. Darüber hinaus ist das System sehr flexibel und kann reibungslos auf verschiedenen Plattformen wie z. B. CPUs, GPUs oder auch großen, verteilten Systemen ausgeführt werden [AAB⁺16]. Zu betonen ist, dass TensorFlow im Rahmen dieser Arbeit nur als Backend Engine für Keras eingesetzt wurde und folglich keine ANNs mit der TensorFlow API entworfen wurden.

5.2 Grundlegendes Design der ANNs

Beim Aufbau der ANNs wurden in dieser Arbeit einige grundlegende Design Entscheidungen getroffen. Beispielsweise wird immer ein MLP genutzt, bei dem die verschiedenen Layers vorwärts miteinander verbunden sind. In den Hidden Layers werden ReLUs als Aktivierungsfunktion eingesetzt und im Output Layer die Sigmoid-Funktion. Als Fehlerfunktion wird die in Keras vorhandene binäre Kreuzentropie verwendet [C⁺18]. Diese erinnert an die Methode der binären Relevanz, da die Kreuzentropie hier unabhängig für jede Klasse optimiert wird und die Abhängigkeiten zwischen den Labels folglich keine Rolle spielen.

Die Kombination aus der Sigmoid-Funktion im Output Layer und der binären Kreuzentropie als Fehlerfunktion wird allgemein zur Lösung von Multi-Label Problemstellungen empfohlen [Cho18]. Dabei wird für jedes Output Neuron eine Wahrscheinlichkeit berechnet, wobei sich die einzelnen Wahrscheinlichkeiten für die Labels nicht auf den Wert 1 addieren. Durch die Verwendung eines Schwellenwertes kann dann die Menge an Labels bestimmt werden, die für ein Beispiel hervorgesagt wird.

Für das Training der ANNs wird der Backpropagation Algorithmus genutzt. Konkret wird das Mini-Batch Gradientenverfahren in Kombination mit dem Optimierungsalgorithmus Adam verwendet. Zur Regularisierung der Modelle kommt Early Stopping im Zusammenspiel mit einer 3-fachen Kreuzvalidierung zum Einsatz. Des Weiteren werden Batch-Normalisierung und Dropout verwendet. Batch-Normalisierung wird immer vor der Aktivierung der Hidden Layers verwendet und Dropout stets danach. Außerdem werden die Gewichte anhand der gleichmäßigen Xavier Formel initialisiert.

Die Anzahl an Input Units orientiert sich am Feature Vektor. Die Anzahl an Output Units wird von den möglichen Reparaturcodes bestimmt. Zu betonen ist, dass es einige Hyperparameter gibt, wie z. B. die Lernrate oder die Anzahl an Hidden Layers, deren Werte durch eine systematische Hyperparameter Op-

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

timierung ermittelt werden. Abbildung 5.1 fasst das grundlegende ANN-Design für diese Arbeit zusammen.

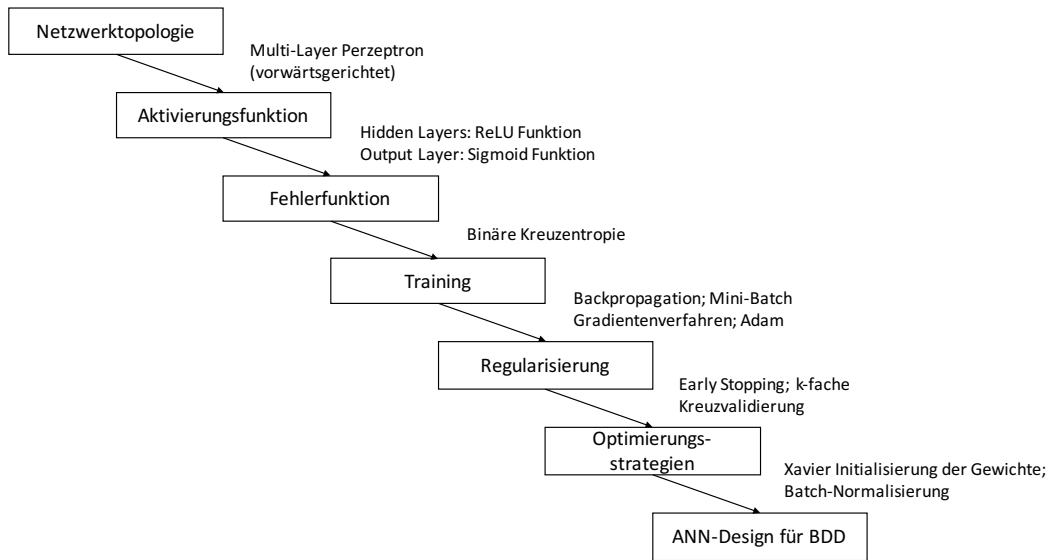


ABBILDUNG 5.1: Grundlegendes ANN-Design im Hinblick auf die BDD-Problemstellung

5.3 Auswahl vorhersagbarer Reparaturcodes

Der für diese Arbeit verwendete BDD-Datensatz enthält die Schadens- und Diagnosedaten von sämtlichen Mercedes-Benz Baureihen. In dieser Arbeit werden jedoch nur die Reparaturprognosen für die A-Klasse evaluiert. Tabelle 5.1 zeigt die ungefähre Anzahl an Werkstattbesuchen, Fehlercodes und Reparaturcodes für alle Baureihen und speziell für die A-Klasse.

	Werkstattbesuche	Fehlercodes	Reparaturcodes
Alle Baureihen	2700000	47000	163000
A-Klasse	128000	3000	16000

TABELLE 5.1: Vergleich zwischen dem Datensatz mit allen Baureihen und dem mit den Daten der A-Klasse

5.3 Auswahl vorhersagbarer Reparaturcodes

Nach der Filterung auf die A-Klasse wird der resultierende Datensatz in einen Trainings- und Testdatensatz geteilt. Der Anteil der Trainingsdaten beträgt hierbei 80 % und der der Testdaten 20 %, wobei die Zuordnung der einzelnen Instanzen zufällig erfolgt. Der Trainingsdatensatz wird für die Modellselektion darüber hinaus noch in Trainings- und Validierungsdaten geteilt. Aus diesem Grund ist der Anteil von 80 % auch gerechtfertigt. Im Übrigen wird der Testdatensatz nur für die finale Evaluation genutzt.

Da einige Reparaturcodes (z. B. Lackschäden) nicht anhand des elektrischen Zustands des Fahrzeugs vorhergesagt werden können, gilt es, diese vor dem Training zu entfernen. Bei der aktuellen BDD-Pipeline wird dies mit Hilfe von Assoziationsregeln zwischen den Fehler- und Reparaturcodes gelöst. Dabei werden alle Reparaturcodes, für die keine Assoziationsregel gefunden wurde, vom Datensatz gelöscht. In dieser Arbeit werden die vorhersagbaren Reparaturcodes auf eine andere Weise ermittelt: Für jede Oberklasse von Reparaturcodes wird ein ANN mit einer 3-fachen Kreuzvalidierung trainiert. Die Ergebnisse des Netzes werden dann jeweils auf den Validierungsdaten evaluiert. Alle Klassen, die mindestens einmal korrekt vorhergesagt wurden, werden für das weitere Training berücksichtigt. Alle anderen werden vom Datensatz entfernt.

Letztlich konnten dadurch 537 ASRA-Codes, 636 PARTS-Codes und 519 SSL7-Codes identifiziert werden, die mindestens einmal richtig vorhergesagt wurden. Über alle drei Oberklassen hinweg konnten die Reparaturcodes der A-Klasse von ca. 16000 auf knapp 1700 reduziert werden. Zu erwähnen ist, dass bei der aktuellen Lösungsvariante von BDD insgesamt 524 Classes Largely Correct für die A-Klasse gefunden werden. Demnach ist die Anzahl von ca. 1700 Klassen nicht zu gering.

Tabelle 5.2 zeigt die genauen Hyperparameter, die bei der Auswahl der vorhersagbaren Reparaturcodes verwendet wurden. Diese wurden heuristisch bzw. durch intensives Experimentieren bestimmt.

Die Reduktion der Reparaturcodes hat u.a. zur Folge, dass einige Zeilen keinen gesetzten Reparaturcode mehr haben. Das liegt daran, dass die entsprechenden Klassen bzw. Spalten vom Datensatz entfernt wurden. Darüber hinaus

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

Lernrate	Hidden Layer	Hidden Units	Batch Size	Patience	Dropout
0.001	2	2500	256	20	0.5

TABELLE 5.2: Verwendete Hyperparameter für die Auswahl der vorhersagbaren Reparaturcodes

gibt es viele Zeilen, bei denen nur zu einer der drei Oberklassen ein Reparaturcode gesetzt ist. Dies fällt insbesondere auf, wenn man die Daten der drei Oberklassen voneinander trennt. So gibt es z. B. zahlreiche Werkstattbesuche ohne einen PARTS-Code. Erste Tests haben gezeigt, dass die irrelevanten Zeilen negative Auswirkungen auf die Leistung der BDD-Vorhersagen haben. Daher wurden die Zeilen ohne einen Reparaturcode vom Datensatz entfernt. Wie viele Werkstattbesuche dabei gelöscht wurden, wird bei der Vorstellung der verschiedenen Netzwerkarchitekturen noch deutlich.

5.4 Netzwerkarchitekturen

Im Zusammenhang mit dem Einsatz von ANNs in der BDD-Pipeline werden in dieser Arbeit verschiedene Netzwerkarchitekturen evaluiert. Diese basieren auf grundlegenden Ideen zur Verbesserung der BDD Modelle. Die vorgestellten Netzwerkarchitekturen werden immer mit der entsprechenden Abkürzung (NA1, NA2 etc.) vorgestellt.

5.4.1 Reduktion der Anzahl an Modellen

Bei der aktuellen BDD-Pipeline werden sowohl alle Baureihen, als auch alle Oberklassen von Reparaturcodes getrennt trainiert. Darüber hinaus wird die One-versus-Rest Methode genutzt, so dass für jede Klasse ein binäres XGBoost-Modell generiert wird (siehe Abbildung 5.2). Insgesamt entstehen durch diesen Aufbau über 60000 Modelle, was für eine große Komplexität sorgt.

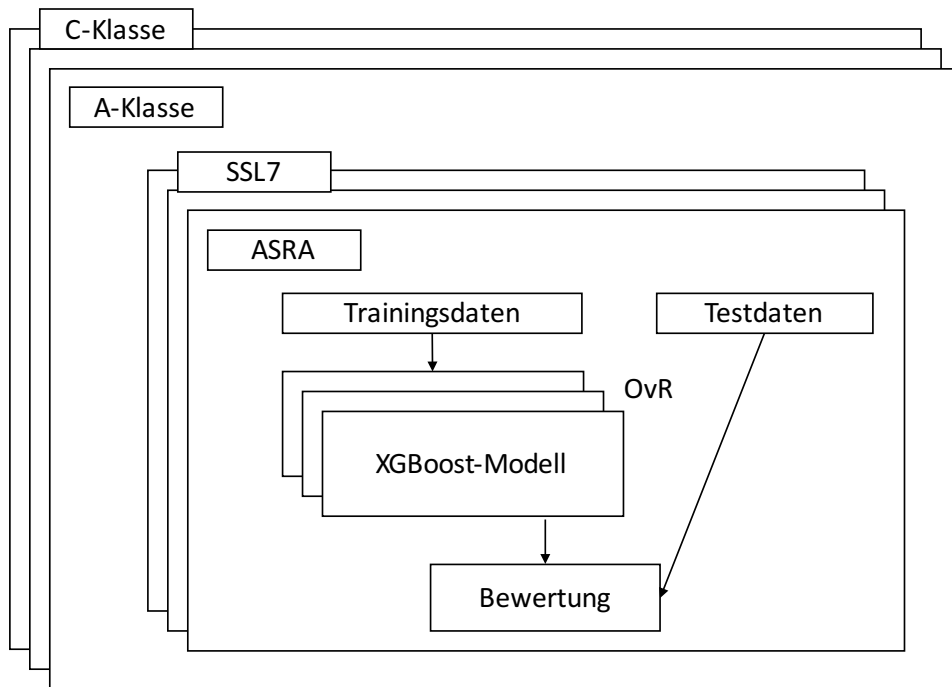


ABBILDUNG 5.2: Aktuelle Lösungsvariante von BDD mit XGBoost und OvR

Eine Idee zur Verbesserung der Reparaturprognosen von BDD ist es, diese große Anzahl an Modellen weitestgehend zu reduzieren. Da One-versus-Rest beim Einsatz von ANNs nicht benötigt wird, fällt schon mal ein Großteil der Modelle weg. Um die Anzahl der Modelle weiter zu reduzieren, werden in dieser Arbeit drei Netzwerkarchitekturen getestet. Zu betonen ist, dass dabei standardmäßig nur die Daten der A-Klasse berücksichtigt werden.

Bei der ersten Netzwerkarchitektur (NA1) wird ein ANN für jede der drei Oberklassen von Reparaturcodes trainiert. Dieser Ansatz ist ähnlich zur aktuellen Lösungsvariante von BDD, bloß ohne die One-versus-Rest Methode. Die Tabelle 5.3 zeigt die Anzahl an Werkstattbesuchen und Labels sowie die Label Kardinalität (siehe Kapitel 2.4) beim Trainings- und Testdatensatz für diese Netzwerkarchitektur. Diese Informationen werden im Folgenden auch zu allen anderen Netzwerkarchitekturen gegeben.

Auffällig ist, dass die Anzahl an Werkstattbesuchen stark variiert. Das liegt daran, dass nach der Trennung der Daten in Bezug auf die Oberklassen viele

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

	Oberklasse	Werkstattbesuche	Labels	LK
Training	ASRA	94778	537	2,67
	PARTS	65919	636	2,25
	SSL7	82733	519	1,17
Test	ASRA	23711	537	2,65
	PARTS	16521	636	2,26
	SSL7	20802	519	1,18

TABELLE 5.3: Trainings- und Testdatensatz zu NA1

Zeilen gar keinen Reparaturcode mehr hatten und daher gelöscht wurden. Wie bereits beschrieben, ist dies auf die Auswahl der vorhersagbaren Reparaturcodes zurückzuführen. Bei den PARTS-Codes wurden z. B. über ein Drittel der Zeilen vom Datensatz entfernt, weil sie für das Training irrelevant sind.

Die zweite Netzwerkarchitektur (NA2) sieht vor, ein ANN mit den Daten von allen Oberklassen zu trainieren. Ideal wäre es, wenn die Vorhersagen der ASRA-, PARTS- und SSL7-Codes auf diese Weise stärker miteinander zusammenhängen würden. Allerdings wird in dieser Arbeit die binäre Kreuzentropie als Fehlerfunktion genutzt, die die Abhängigkeiten zwischen den Labels generell nicht berücksichtigt. Im Gegensatz zu NA1, bei der die Oberklassen getrennt trainiert werden, ist die Anzahl an Labels und der Wert von LK hier deutlich größer (siehe Tabelle 5.4). Ein weiterer Unterschied ist, dass nicht so viele Werkstattbesuche vom Datensatz entfernt wurden, da pro Zeile meistens immer ein Reparaturcode gesetzt ist. Allerdings sind folglich auch viele Zeilen enthalten, bei denen es nur zu einer der drei Oberklassen einen Reparaturcode gibt. Sicher ist z. B., dass etwa ein Drittel der Werkstattbesuche keinen gesetzten PARTS-Code beinhalten.

Bei der dritten Netzwerkarchitektur (NA3) wird wie bei NA1 ein ANN für jede der drei Oberklassen trainiert. Allerdings werden bei NA3 neben den Daten der Mercedes-Benz A-Klasse auch die der CLA- und GLA-Klasse verwendet. Die

	Werkstattbesuche	Labels	LK
Training	99416	1692	5,01
Test	24835	1692	5,01

TABELLE 5.4: Trainings- und Testdatensatz zu NA2

Auswahl der vorhersagbaren Reparaturcodes wurde für die CLA- und GLA-Klasse nach dem gleichen Prozedere durchgeführt wie bei der A-Klasse (siehe Kapitel 5.3). Die Tabelle 5.5 zeigt die Anzahl der ausgewählten Reparaturcodes bei den verschiedenen Baureihen und Oberklassen. Außerdem wird für jede Oberklasse auch die Größe der Vereinigungsmenge der Reparaturcodes von den drei Baureihen präsentiert. Diese wird als Gesamtanzahl bezeichnet.

	ASRA	PARTS	SSL7
A-Klasse	545	637	519
CLA-Klasse	484	656	566
GLA-Klasse	437	539	471
Gesamtanzahl	685	865	825

TABELLE 5.5: Anzahl der vorhersagbaren Reparaturcodes bei der A-, CLA- und GLA-Klasse

An der Gesamtanzahl ist zu erkennen, dass die drei Baureihen eine relativ große Schnittmenge bei den ausgewählten Reparaturcodes haben, und das bei allen Oberklassen. Im Hinblick auf das Training der Modelle ist dies vielversprechend, da deutlich mehr Daten zur Verfügung stehen (nämlich von drei Baureihen) und die Reparaturcodes zu einem großen Anteil die gleichen sind (siehe Tabelle 5.6).

	Oberklasse	Werkstattbesuche	Labels	LK
Training	ASRA	260083	685	2,52
	PARTS	165153	865	2.24
	SSL7	237103	825	1,18
Test	ASRA	65026	685	2,52
	PARTS	41292	865	2.23
	SSL7	59329	825	1,18

TABELLE 5.6: Trainings- und Testdatensatz zu NA3

5.4.2 Transfer Lernen bei den SSL7-Codes

Eine weitere Idee zur Verbesserung von BDD zielt auf die Vorhersagen der SSL7-Codes ab. Dabei soll versucht werden, die Vorhersagen der SSL7-Codes durch eine Art *Transfer Lernen* zu verbessern. Transfer Lernen ist ein bekannter Ansatz beim Deep Learning, der gerne bei der Bild- oder Spracherkennung eingesetzt wird [GBCB16]. Torrey definiert Transfer Lernen z. B. wie folgt: «Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned» [TS10].

Ziel des Transfer Lernens ist es demnach die Leistungsfähigkeit bei der Hauptaufgabe durch die erlernten Regeln bei einer verwandten Aufgabe zu verbessern [TS10]. Abbildung 5.3 zeigt, dass durch Transfer Lernen prinzipiell drei positive Effekte entstehen können. Erstens kann eine höhere Leistungsfähigkeit beim Start des Trainings erreicht werden. Außerdem ist es möglich, dass durch Transfer Lernen weniger Zeit beim Training gebraucht wird, da die Leistungsfähigkeit bei der Hauptaufgabe durch die erlernten Regeln bei der verwandten Aufgabe schneller ansteigt. Letztlich ist es auch möglich, dass mit Transfer Lernen auch langfristig ein höheres Leistungsniveau erreicht wird.[TS10]

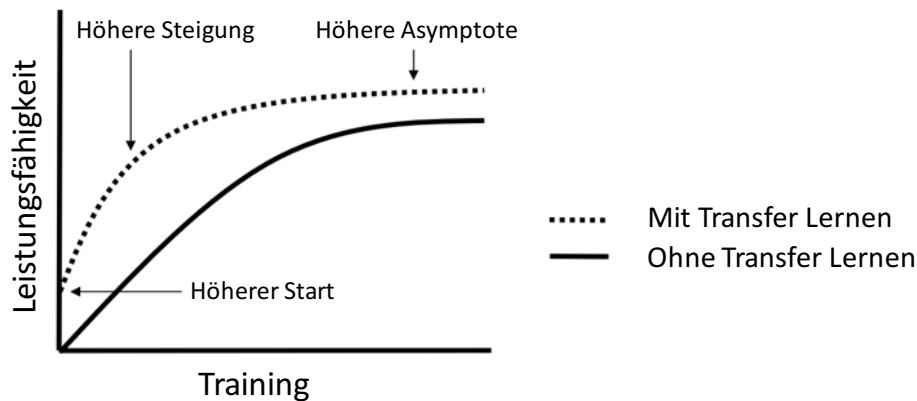


ABBILDUNG 5.3: Ziele des Transfer Lernens [TS10]

Stellt sich die Frage wie Transfer Lernen bei der Verbesserung der SSL7-Codes eingesetzt werden kann. Hierfür gilt es erstmal den Aufbau der SSL7-Codes näher zu beleuchten. Die ersten fünf Stellen des SSL7-Codes repräsentieren den Schadensort und die letzten zwei die Schadensart. Der Schadensort ist außerdem hierarchisch aufgebaut wie folgendes Beispiel verdeutlicht:

- SSL2 (ungenauer Schadensort): T_SSL7_14 steht für die Abgasanlage.
- SSL5 (Schadensort): T_SSL7_1400X steht für den Temperatursensor der Abgasanlage.
- SSL7 (Schadensort und -art): T_SSL7_1400X-73 repräsentiert einen elektrischen Fehler beim Temperatursensor der Abgasanlage.

Dieser Aufbau der SSL7-Codes soll nun für das Transfer Lernen genutzt werden. Die vierte Netzwerkarchitektur (NA4) sieht vor, jeweils ein künstliches neuronales Netz mit den SSL2-, SSL5- und SSL7-Codes zu trainieren. Entscheidend ist dabei, dass die finalen Gewichte eines Netzes jeweils zur Initialisierung des nächsten Netzes verwendet werden. Abbildung 5.2 verdeutlicht diese Vorgehensweise.

Da die Klassifizierung der SSL2-, SSL5- und SSL7-Codes verwandte Aufgaben darstellen, kann NA4 im Hinblick auf die zitierte Definition von Torrey [TS10]

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

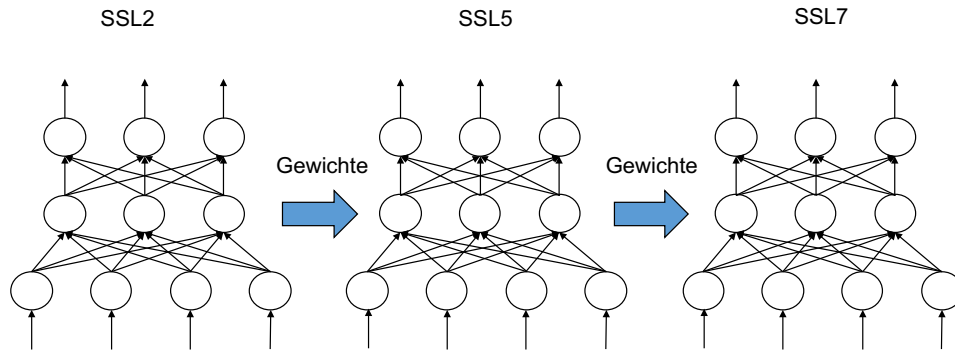


ABBILDUNG 5.4: Transfer Lernen beim Training des SSL7-Modells

als Transfer Lernen bezeichnet werden. Neben den allgemeinen Vorteilen des Transfer Lernens, wäre es optimal, wenn dieser Ansatz noch einen weiteren Effekt auf die Vorhersagen der SSL7-Codes hätte. Genauer gesagt soll durch die Initialisierung der Gewichte mit den Parametern des vorherigen Netzes erreicht werden, dass falsche SSL7-Vorhersagen mehr in die richtige Richtung gehen. Angenommen das Problem ist ein elektrischer Fehler beim Temperatursensor der Abgasanlage. Wenn der prognostizierte SSL7-Code nicht exakt richtig ist, aber z. B. auf die Abgasanlage oder besser noch auf den Temperatursensor der Abgasanlage deutet, dann ist das für die Werkstatt Mechaniker schon viel Wert.

	Oberklasse	Werkstattbesuche	Labels	LK
Training	SSL2	82733	45	1.09
	SSL5	82733	350	1,17
	SSL7	82733	519	1,17
Test	SSL2	20802	45	1.09
	SSL5	20802	350	1,17
	SSL7	20802	519	1,18

TABELLE 5.7: Trainings- und Testdatensatz zu NA4

Tabelle 5.7 zeigt, dass die Anzahl an SSL2- und SSL5-Codes deutlich kleiner ist als die Anzahl der SSL7-Codes. Demnach ist zu erwarten, dass insbesondere die SSL2- und SSL5-Codes besser vorhergesagt werden können als die SSL7-Codes. Durch das stufenweise ablaufende Training und der Weitergabe der entsprechenden Modell Parameter wird versucht, die erlernten Regeln des SSL2- und SSL5-Modells auf das SSL7-Modell zu übertragen und dadurch eine höhere Leistungsfähigkeit bei den Vorhersagen der SSL7-Codes zu realisieren.

5.4.3 Beschreibungen der Fehlercodes als Features

Eine weitere Idee zur Verbesserung von BDD ist es, den Feature-Raum zu erweitern. Konkret sollen die Beschreibungen der Fehlercodes aus den Steuergeräten als zusätzliche Features verwendet werden. Dafür wurde auf Basis der Beschreibungen ein sogenanntes *Bag-of-Words* (BoW) Modell erstellt, das im Bereich der *maschinellen Verarbeitung von natürlicher Sprache* (Natural Language Processing) öfters eingesetzt wird. Das BoW-Modell transformiert die Beschreibungen der Features in eine Menge der darin enthaltenen Wörter, wobei die Strukturen der Sätze komplett verloren gehen. Zu erwähnen ist, dass BoW-Modelle eher in ANNs mit einer geringen Anzahl von Hidden Layers eingesetzt werden und weniger im Deep Learning Kontext.[Cho18]

Im Hinblick auf das Vokabular des BoW-Modells werden nur Wörter berücksichtigt, die für die Reparatur Vorhersagen von BDD auch relevant sind. Die sogenannten *Stoppwörter*, wie z. B. Artikel oder Präpositionen, sind demnach nicht im BoW-Modell enthalten.

Das BoW-Modell wird für das Training der ANNs in eine Matrix transformiert, bei der jede Zeile für einen Werkstattbesuch steht und die Spalten die verschiedenen Wörter repräsentieren. Die einzelnen Features in der BoW-Matrix sind binär-codiert und machen deutlich, ob ein Wort in einer Beschreibung aufgetreten ist oder nicht. Dies wird in Tabelle 5.8 beispielhaft angedeutet. Demnach kommen in den Beschreibungen der Fehlercodes beim ersten Werkstattbesuch z. B. die Wörter Kommunikation und Sensor vor.

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

Kommunikation	Batterie	Sensor	Fehlfunktion
1	0	1	0
0	1	0	0
0	0	1	0

TABELLE 5.8: Beispielhafte BoW-Matrix

Die fünfte Netzwerkarchitektur (NA5) sieht so aus, dass für jede der drei Oberklassen ein ANN mit der BoW-Matrix als Input trainiert wird. Darüber hinaus werden die BoW-Features und die Fehlercodes in einer großen Feature-Matrix kombiniert. Diese wird dann in der sechsten Netzwerkarchitektur (NA6) genutzt, um für jede der drei Oberklassen ein ANN zu trainieren. Tabelle 5.9 zeigt den Trainings- und Testdatensatz von NA5 und NA6 mit Fokus auf die beiden Input Varianten.

	Input	Werkstattbesuche	Features	Median
Training	BoW	102034	859	19
	Fehlercodes + BoW	102034	4046	22
Test	BoW	25509	859	19
	Fehlercodes + BoW	25509	4046	22

TABELLE 5.9: Trainings- und Testdatensatz zu NA5 und NA6 mit Fokus auf die Input Varianten

Zu betonen ist, dass die BoW-Matrix ca. ein Drittel weniger Features hat als der normale Feature Vektor mit den Fehlercodes. Die relevanten Wörter bei den Beschreibungen sind also oft die gleichen. Folglich wird in einem Fall der Feature-Raum reduziert und im anderen Fall erweitert. Der Median bezieht sich hier auf die Anzahl der Features pro Werkstattbesuch. Bei den BoW-Features hat der Median den Wert 19 und ist damit im Vergleich zu den Fehlercodes deutlich größer. Denn bei den Fehlercodes hat der Median den Wert 3.

Abschließend ist zu sagen, dass sich NA1, NA3, NA5 und NA6 vom Aufbau her sehr ähnlich sind, da sie jeweils ein ANN für jede der drei Oberklassen trainieren. NA2 ist wiederum die einzige Netzwerkarchitektur, die die verschiedenen Oberklassen in einem ANN trainiert. Bei NA4 wird gar kein ASRA- bzw. PARTS-Modell generiert, da sich diese Netzwerkarchitektur ausschließlich auf die Vorhersagen der SSL7-Codes konzentriert.

5.5 Hyperparameter Optimierung

In Kapitel 5.2 wurden bereits einige wichtige Design Entscheidungen beim Aufbau der ANNs festgelegt. Die verschiedenen Netzwerkarchitekturen geben darüber hinaus die Anzahl der Input Units und Output Units vor. Um für jede Netzwerkarchitektur das beste Modell zu finden, werden in dieser Arbeit noch sechs Hyperparameter evaluiert, deren optimale Werte nicht offensichtlich sind: Lernrate, Anzahl der Hidden Layers, Anzahl der Hidden Units, Patience, Batch Size und Dropout-Rate.

Zur Optimierung dieser Hyperparameter wurde eine 3-fache Kreuzvalidierung in Kombination mit einer teilautomatisierten Variante der *Grid Search* verwendet. Die Vorgehensweise Grid Search orientiert sich dabei an den Empfehlungen von [Ben12] und wird im Folgenden näher erläutert. Die Grid Search ist eine automatisierte, flächendeckende Suche durch alle Kombinationen des definierten Suchraums bzw. durch alle Werte, die für die verschiedenen Hyperparameter definiert wurden [Ben12]. Bei der normalen Grid Search wächst die Anzahl an möglichen Hyperparameter Kombinationen allerdings sehr schnell an und kann folglich einen extremen Rechenaufwand verursachen. Wenn z. B. für jeden der sechs genannten Hyperparameter fünf Werte getestet werden sollten, müssten bei der normalen Grid Search insgesamt 15625 (5^6) verschiedene Hyperparameter Kombinationen evaluiert werden.[Ben12]

Da für diese Arbeit nur ein Computer für das Training und die Evaluation der BDD-Modelle zur Verfügung stand, waren die Ressourcen begrenzt. Für diesen Fall wird in [Ben12] eine Parametersuche vorgeschlagen, bei der die einzelnen

Parameter gemäß der Ceteris-paribus Annahme immer einzeln angepasst werden und die übrigen erstmal gleich bleiben. Die einzelnen Hyperparameter werden dabei immer ausgehend vom besten Modell angepasst. Die dabei verwendete Grid Search ist teilautomatisiert, weil die Auswahl des besten Modells manuell durchgeführt wird. Diese Vorgehensweise wird auch als *koordinierter Abstieg* (Coordinate Descent) bezeichnet [Ben12].

Bei der Anpassung der einzelnen Hyperparameter wurden verschiedene Werte ausprobiert. Die Anfangswerte (Defaultwerte) der Hyperparameter wurden mit Hilfe entsprechender Informationen aus der Literatur und durch intensives Experimentieren bestimmt. Bei der Lernrate von Adam wurde z. B. 0,001 als Defaultwert gewählt. Da Adam die Lernrate während des Trainings automatisch anpasst, muss die Lernrate in der Regel auch nicht so stark optimiert werden [Gér17]. Die Optimierung der Anzahl an Hidden Layers und Hidden Units wurde im Hinblick auf die Problemstellung von BDD mit verhältnismäßig kleinen Werten (ein Hidden Layer und jeweils 500 Hidden Units) gestartet. Anschließend wurde die Anzahl an Hidden Layers und Hidden Units bzw. die Komplexität des Modells immer weiter erhöht, bis die Ergebnisse nicht mehr besser wurden. Diese Vorgehensweise wird z.B in [Cho18] vorgeschlagen. Die Patience und die Batch Size wurden im Übrigen auf die gleich Weise optimiert, wobei der Anfangswert der Patience auf 5 und der der Batch Size auf 32 festgelegt wurde. Bei der Dropout-Rate wurde mit Bezug auf [SHK⁺14] 0,5 als Defaultwert gewählt.

Tabelle 5.10 zeigt die verschiedenen Werte der Hyperparameter, die im Rahmen dieser Arbeit evaluiert wurden.

Ausgehend vom Anfangswert eines Hyperparameters wurde folglich mindestens ein höherer und ein niedrigerer Wert evaluiert. Wenn z. B. der höhere Wert ein besseres Ergebnis mit sich brachte, dann wurde die Suche in diese Richtung erweitert. Wie in Tabelle 5.10 zu erkennen ist, wurde die jeweilige Schrittgröße bei der Anpassung der verschiedenen Hyperparameter konstant gehalten. Da die Optimierung immer vom besten Modell ausging, wurden die

5.5 Hyperparameter Optimierung

Hyperparameter	Getestete Werte
Lernrate	0.01, 0.001, 0.0001
Hidden Layers	1, 2, 3, 4
Hidden Units	500, 1000, 1500, 2000
Patience	5, 10, 15, 20, 25
Batch Size	32, 64, 128, 256, 512
Dropout-Rate	0.4, 0.5, 0.6

TABELLE 5.10: Getestete Werte bei der Hyperparameter Optimierung

Werte der verschiedenen Hyperparameter im Laufe dieses Prozesses mehrfach durchprobiert.

Da der Unterschied zwischen dem Trainings- und Validierungsfehler ein wichtiger Indikator für die Leistung bei der Generalisierung ist, wurden die beiden Lernkurven bei der Hyperparameter Optimierung stets berücksichtigt. Abbildung 5.5 zeigt den Trainings- und Validierungsfehler für das finale ASRA-Modell bei der Netzwerkarchitektur mit den getrennten Oberklassen.

Zu erkennen ist, dass die beiden Lernkurven relativ weit auseinander gehen und das trainierte Modell demnach überangepasst ist. Allerdings konnte festgestellt werden, dass diese Überanpassung grundsätzlich für bessere Ergebnisse sorgte. Das liegt wahrscheinlich daran, dass die Fehlerfunktion nicht ganz genau zu den verwendeten Metriken passt.

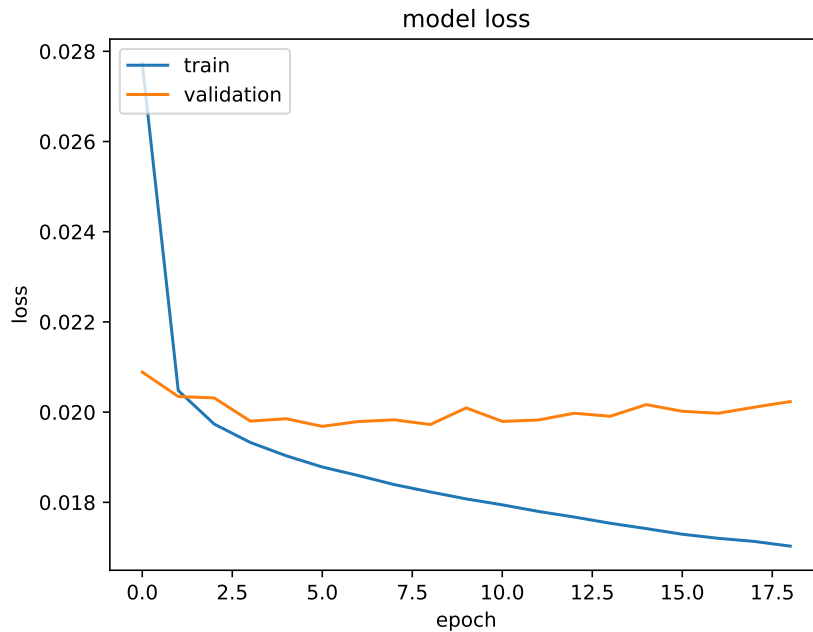


ABBILDUNG 5.5: Lernkurve des besten ASRA-Modells

5.6 Auswahl der besten Modelle

Ziel der Hyperparameter Optimierung ist es, das jeweils beste Modell für die verschiedenen Netzwerkarchitekturen zu finden. Da die Ergebnisse der Hyperparameter Optimierung auf einer 3-fachen Kreuzvalidierung basieren, mussten die durchschnittlichen Resultate von den drei Trainingsdurchgängen der verschiedenen Modelle verglichen werden. Auf diese Weise konnte jeweils das beste Modell für eine Netzwerkarchitektur gefunden werden.

Die Auswahl der besten Modelle wurde dabei mit Hilfe von dreidimensionalen Streudiagrammen (Scatter Plots) getroffen. Jede Dimension repräsentiert dabei eine der drei vom Fachbereich vorgegebenen Metriken: Attempt Rate, Top-3 Precision und Classes Largely Correct. Abbildung 5.6 zeigt das dreidimensionale Streudiagramm zur Auswahl des besten ASRA-Modells bei der Netzwerkarchitektur mit den getrennten Oberklassen.

Die Punkte zeigen die durchschnittlichen Ergebnisse der verschiedenen ANNs bei der 3-fachen Kreuzvalidierung, wobei jedes ANN mit anderen Hyperpa-

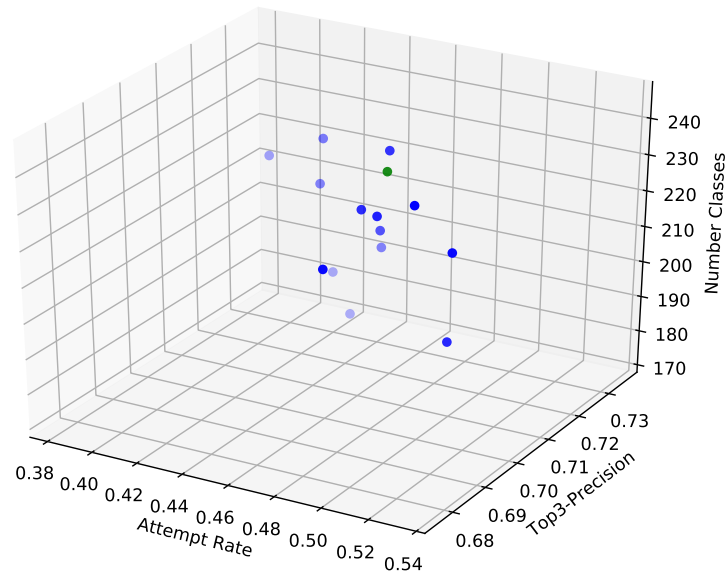


ABBILDUNG 5.6: Dreidimensionaler Scatter Plot zur Auswahl des besten ASRA-Modells bei NA1

rametern trainiert wurde. Die Punkte wurden anhand der Resultate von Attempt Rate, Top-3 Precision und Classes Largely Correct im dreidimensionalen Raum platziert. Im Hinblick auf die Skalen der drei Achsen, wäre es optimal, wenn sich die Punkte oben rechts im Diagramm befinden würden. Der grün dargestellte Punkt repräsentiert hier das beste ASRA-Modell bei der Netzwerkarchitektur mit den getrennten Oberklassen. Dementsprechend wurde für jede Netzwerkarchitektur und ggf. jede Oberklasse ein entsprechendes Streudiagramm erstellt.

Die Selektion des besten Modells erfolgte stets manuell. Genauer gesagt wurde geschaut, welcher Punkt im Vergleich zu den anderen am besten im Streudiagramm platziert ist. Dies war nicht immer ganz eindeutig, da es praktisch nie ein Modell gab, das in allen drei Metriken besser war als die anderen. Das liegt u.a. auch daran, dass sich Attempt Rate, Top-3 Precision und Classes Largely Correct gegenseitig beeinflussen. In Abbildung 5.7 kann man z. B. einen linearen Zusammenhang zwischen Attempt Rate und Top-3 Precision erkennen.

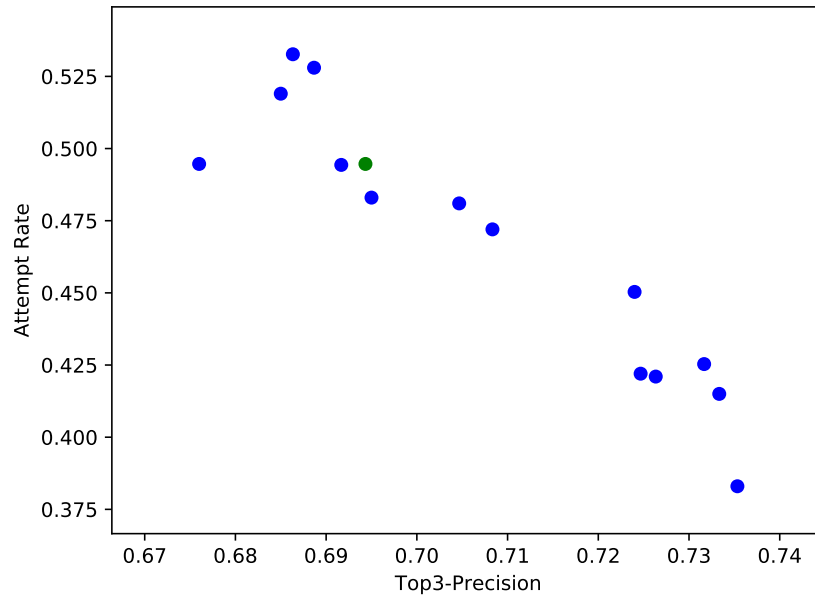


ABBILDUNG 5.7: Zweidimensionaler Scatter Plot mit Fokus auf Attempt Rate und Top-3 Precision

Dementsprechend haben die Modelle mit einer höheren Attempt Rate in der Regel eine niedrigere Top-3 Precision und umgekehrt. Die Attempt Rate ist wiederum abhängig vom verwendeten Schwellenwert. Dieser bestimmt, welche Wahrscheinlichkeit ein Output Neuron mindestens haben muss, damit das entsprechende Label vorhergesagt wird.

Abbildung 5.8 zeigt, dass auch Attempt Rate und Classes Largely Correct korrelieren. Genauer gesagt steigt Classes Largely Correct in den meisten Fällen proportional mit der Attempt Rate an.

Bei der Auswahl der besten Modelle hat sich gezeigt, dass die in Tabelle 5.11 dargestellten Werte für die sechs Hyperparameter in fast allen Fällen zu den besten Ergebnissen geführt haben. LR steht im Folgenden für die Lernrate, HL für die Anzahl an Hidden Layers, HU für die Anzahl an Hidden Units, PA für die Patience, BS für die Batch Size und DR für die Dropout-Rate.

Die Werte von LR, HL, HU und DR wurden dabei bei allen für die finale Evaluation ausgewählten Modelle verwendet. Nur bei PR und BS wurde

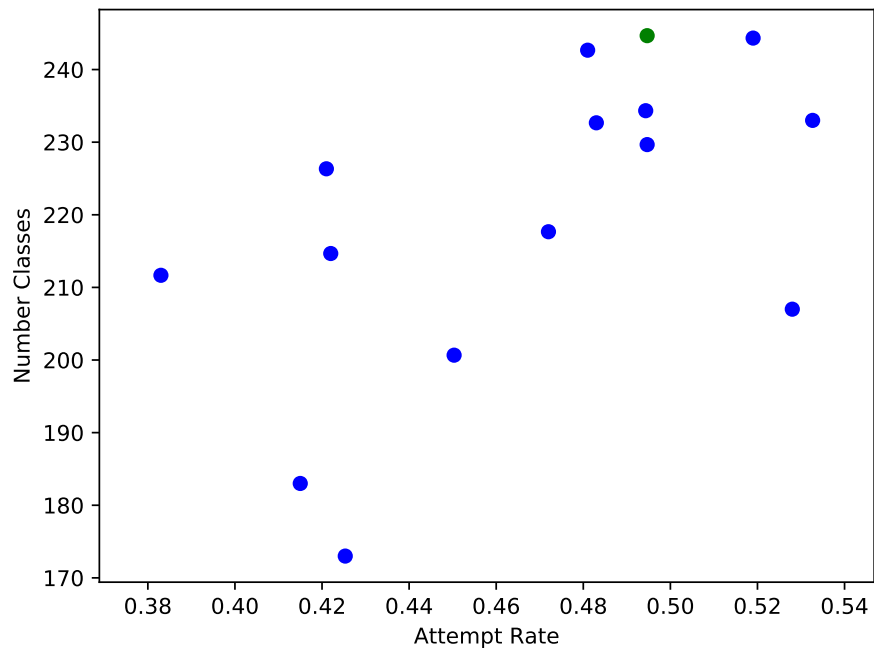


ABBILDUNG 5.8: Zweidimensionaler Scatter Plot mit Fokus auf Attempt Rate und Classes Largely Correct

LR	HL	HU	PA	BS	DR
0.001	2	1000	15	128	0.5

TABELLE 5.11: Optimierte Hyperparameter für die finale Evaluation

bei einzelnen Modellen ein leicht niedrigerer bzw. höherer gewählt. Bei NA3, der Netzwerkarchitektur mit den Daten von mehreren Baureihen, wurde beim SSL7-Modell z. B. eine PA von 20 gewählt. Im Hinblick auf die größere Anzahl an Daten macht dies auch Sinn. Beim ASRA- und PARTS-Modell von NA3 liegt der optimale Wert von PA allerdings bei 15. Bei NA4 musste für das Training des SSL2- und SSL5-Modells eine kleinere PA gewählt werden, da die Modelle sonst zu stark überangepasst wären. Der optimale Wert von PA beträgt beim SSL2-Modell 3 und beim SSL5-Modell 10.

Generell konnte festgestellt werden, dass die Leistung der BDD Vorhersagen nicht so stark von den feingranularen Hyperparameter Optimierungen abhängt. Was die Anzahl an Hidden Layers betrifft, so können grundsätzlich auch viele nicht-lineare Problemstellungen mit zwei Hidden Layers gelöst werden. Dabei kommt es u. a. auch auf die Aussagekraft der Input Daten an. Bei BDD sind die einzelnen Features ziemlich aussagekräftig. Daher ist eine hohe Anzahl an Hidden Layers nicht zwingend erforderlich [GBCB16].

Die jeweils besten Modelle der verschiedenen Netzwerkarchitekturen wurden für die finale Evaluation nochmal mit den ganzen Trainings- und Validierungsdaten trainiert, also mit 80 % des gesamten Datensatzes. Der Testdatensatz mit den restlichen 20 % der Daten wurde dabei zur Bewertung der ausgewählten Modelle genutzt. Im folgenden Abschnitt werden die Ergebnisse der finalen Evaluation präsentiert und diskutiert.

5.7 Präsentation und Diskussion der finalen Ergebnisse

Tabelle 5.12 fasst die Ergebnisse der finalen Evaluation für die verschiedenen Netzwerkarchitekturen (NA1, NA2 etc.) zusammen. XGB steht hier für die aktuelle Lösungsvariante von Daimler. Zu betonen ist, dass sich die Ergebnisse der Evaluation nur auf die Daten der Mercedes-Benz A-Klasse beziehen. Im

5.7 Präsentation und Diskussion der finalen Ergebnisse

Fall von NA3 wurde zwar mit Daten von verschiedenen Baureihen trainiert, aber nur auf die Reparatur Prognosen für die A-Klasse evaluiert.

Bei NA1 ist im Hinblick auf Classes Largely Correct eine deutliche Leistungssteigerung in Vergleich zu XGB festzustellen. Bei den SSL7-Codes werden hier sogar mehr als doppelt so viele Classes Largely Correct gefunden als bei XGB. Außerdem ist NA1 auch bei der Attempt Rate immer leicht besser als XGB und beim PARTS-Modell sogar deutlich. Bei der Top-3 Precision sind die Ergebnisse von NA1 und XGB relativ ausgeglichen.

Die Ergebnisse von NA2 sind mit Ausnahme der Attempt Rate beim ASRA-Modell immer schlechter als die von NA1. Das Training eines ANNs mit den Daten von allen Oberklassen ist folglich kein vielversprechender Ansatz. Das Hauptproblem ist hier wahrscheinlich, dass es bei zu vielen Trainingsinstanzen nur zu einer der drei Oberklassen einen Reparaturcode gibt. Das lässt sich u. a. daran erkennen, dass die Leistung von NA2 im Vergleich zu NA1 insbesondere beim PARTS-Modell stark abfällt. Dabei gilt es bemerken, dass ca. ein Drittel der Werkstattbesuche bei NA2 keinen PARTS-Code beinhalten. Bei den ASRA- und SSL7-Codes ist dieser Anteil nicht so groß (siehe Tabelle 5.3).

Die Resultate von NA3 zeigen bei Attempt Rate und Classes Largely Correct eine deutliche Leistungssteigerung im Vergleich zu XGB. XGB ist hingegen nur bei der Top-3 Precision des SSL7- und ASRA-Modells besser als NA3. Im Vergleich zu NA1 fällt auf, dass die Attempt Rate bei NA3 immer deutlich besser ist. Das liegt u. a. auch daran, dass bei NA3 ein andere Mindestwahrscheinlichkeit in Bezug auf die Vorhersagen für die Labels verwendet wurde. Bei NA3 liegt die optimale Mindestwahrscheinlichkeit bei 0,3 bzw. 30 % Wahrscheinlichkeit und bei allen anderen Modellen bei 0,5. Dieser Unterschied ist wahrscheinlich dadurch zu erklären, dass deutlich mehr Daten für das Training von NA3 zur Verfügung standen als bei den anderen Netzwerkarchitekturen. Zu betonen ist, dass NA1 dafür bei der Top-3 Precision bessere Ergebnisse als NA3 liefert. Insgesamt sind die Resultate von NA3 leicht besser als die von NA1 und generell sehr vielversprechend.

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

	Oberklasse	Attempt Rate	Top-3 Precision	Number Classes
XGB	ASRA	0.475	0.715	195
	PARTS	0.372	0.688	235
	SSL7	0.512	0.699	94
NA1	ASRA	0.49	0.706	260
	PARTS	0.474	0.76	313
	SSL7	0.522	0.615	207
NA2	ASRA	0.557	0.651	253
	PARTS	0.402	0.63	288
	SSL7	0.492	0.597	167
NA3	ASRA	0.647	0.642	251
	PARTS	0.512	0.739	339
	SSL7	0.645	0.615	212
NA4	SSL7	0.557	0.612	208
	SSL5	0.501	0.713	182
	SSL2	0.992	0.748	30
NA5	ASRA	0.437	0.704	181
	PARTS	0.517	0.728	258
	SSL7	0.467	0.649	173
NA6	ASRA	0.452	0.73	241
	PARTS	0.549	0.731	294
	SSL7	0.51	0.626	200

TABELLE 5.12: Ergebnisse der finalen Evaluation

5.7 Präsentation und Diskussion der finalen Ergebnisse

Die Ergebnisse von NA4 zeigen, dass die Vorhersagen der SSL7-Codes im Vergleich zu NA1 sehr ähnlich sind und nur bei der Attempt Rate eine leicht bessere Leistung haben. Zu erkennen ist, dass insbesondere die SSL2-Codes, aber auch die SSL5-Codes sehr gut prognostiziert werden können. Die erhofften Effekte des Transfer Lernens auf die Vorhersagen der SSL7-Codes sind in den entsprechenden Resultaten allerdings nicht zu sehen. Da bei NA4 immer drei Modelle trainiert werden müssen, dauert das Training länger als bei NA1. Ob die fehlerhaften SSL7-Prognosen durch das Transfer Lernen mehr in die richtige Richtung gehen, kann in dieser Evaluation nicht direkt gesagt werden. Denn dies wird von den verwendeten Metriken nicht widerspiegelt. Diesen Effekt könnten z. B. fachliche Experten für die Fahrzeugdiagnose überprüfen, indem sie die Vorhersagen des SSL7-Modells von NA1 und NA4 mit Bezug auf die tatsächlichen SSL-Codes vergleichen.

Die Ergebnisse von NA5 verdeutlichen, dass die Reparaturprognosen anhand der Beschreibungen der Fehlercodes bzw. der BoW-Matrix grundsätzlich funktionieren und im Vergleich zu XGB z. B. deutlich bessere Ergebnisse bei den Classes Largely Correct liefern. Allerdings sind die Resultate von NA5 nicht so gut wie die von NA1, wo die Fehlercodes als Features verwendet werden und nicht deren Beschreibungen. Die Kombination der BoW-Features und der Fehlercodes in einer großen Feature-Matrix bringt ebenfalls keine Vorteile gegenüber NA1. Dies verdeutlichen die Resultate von NA6. Die Ergebnisse von NA1 und NA6 sind grundsätzlich sehr ähnlich und variieren nur leicht. Die Erweiterung des Feature-Raums mit den Beschreibungen der Fehlercodes ist folglich kein vielversprechender Ansatz.

Alles in allem zeigen die Ergebnisse der finalen Evaluation, dass durch den Einsatz von ANNs eine deutliche Leistungssteigerung von BDD im Vergleich zur aktuellen Lösungsvariante möglich ist. Dies gilt insbesondere im Hinblick auf die Metriken Attempt Rate und Classes Largely Correct. Die besten Resultate lieferten NA1 und NA3. Bei diesen Netzwerkarchitekturen wurden die drei Oberklassen von Reparaturcodes getrennt trainiert. NA3 ist besonders vielversprechend, weil die Ergebnisse hier nochmal leicht besser sind als bei

5 Einsatz von künstlichen neuronalen Netzen in der BDD-Pipeline

NA1. Darüber hinaus kann NA3 noch mit den Daten von anderen Fahrzeugbaureihen erweitert werden.

6 Zusammenfassung und Ausblick

Diese Arbeit beschäftigte sich mit BDD, einem System von Daimler, das weltweit in den meisten Mercedes-Benz Werkstätten zur automatischen Fahrzeugdiagnose eingesetzt wird. Im Hinblick auf den steigenden Funktionsumfang in den Fahrzeugen leistet BDD einen wichtigen Beitrag zur Unterstützung der Werkstatt Mitarbeiter. Folglich hat Daimler ein großes Interesse daran, die Reparaturprognosen von BDD zu optimieren. Das aktuelle Produktivsystem von BDD basiert auf der One-versus-Rest Methode und XGBoost, einer beliebten Implementierung von Gradient Boosting Machines. In dieser Arbeit wurde genauer untersucht, ob es möglich ist, die Leistung von BDD durch den Einsatz von alternativen Maschinellen Lernverfahren zu verbessern.

Die Reparaturprognosen von BDD lassen sich in drei Oberklassen einteilen und werden auf der Basis von den Fehlercodes aus den Steuergeräten der Fahrzeuge vorhergesagt. Im Hinblick auf Maschinelles Lernen gilt es folglich eine Funktion zu approximieren, die die Fehlercodes bestmöglich auf die entsprechenden Reparaturcodes abbildet. Die Analyse der Problemstellung hat gezeigt, dass es sich bei BDD um überwachtes Lernen handelt und es keinen Bedarf für ein Online-Learning System gibt. Für das Training der Modelle stehen grundsätzlich genug Daten zur Verfügung. Allerdings müssen auch sehr viele verschiedene Reparaturcodes vorhergesagt werden, die allgemein sehr ungleichmäßig verteilt sind.

Der Aufgabentyp von BDD stellt eine Multi-Label Klassifikation dar, wobei die Reparaturprognosen mit Wahrscheinlichkeiten angegeben werden sollen. Die bei BDD verwendeten Metriken (Attempt Rate, die Top-3 Precision und Classes Largely Correct) sind speziell für die Problemstellung angepasst und

6 Zusammenfassung und Ausblick

werden vom zuständigen Fachbereich bei Daimler vorgegeben. Diese Arbeit hat gezeigt, dass man in der Literatur auch einige alternative Multi-Label Metriken zu den aktuell eingesetzten findet.

Des Weiteren wurde in dieser Arbeit deutlich gemacht, dass es bei der Multi-Label Klassifikation verschiedene Lösungsvarianten gibt. Einige sind jedoch nicht für hochdimensionale Problemstellungen wie die von BDD geeignet. Bei der Auswahl der Lösungsvariante muss generell abgewogen werden, ob es sich lohnt, die Zusammenhänge zwischen den Labels stärker zu berücksichtigen. Denn dies geht in der Regel mit einem erhöhten Rechenaufwand einher [ZZ14]. Da die Berücksichtigung der Label-Korrelationen bei BDD nicht an erster Stelle steht, wurde für diese Arbeit kein speziell für die Multi-Label Klassifikation angepasster Algorithmus verwendet.

Stattdessen wurde in dieser Arbeit genauer untersucht, ob die Reparaturprognosen von BDD durch den Einsatz von ANNs verbessert werden können. ANNs unterstützen Multi-Label Problemstellungen implizit und sind dafür bekannt, gut mit komplexen Datensätzen umgehen zu können. Des Weiteren sind ANNs in der Lage, die Labels mit den entsprechenden Wahrscheinlichkeiten vorherzusagen [Mit97]. Zu betonen ist, dass es einige optimierte Algorithmen für das Training und zur Regularisierung von ANNs gibt.

Im Hinblick auf den Einsatz von ANNs in der BDD-Pipeline wurde im Rahmen dieser Arbeit eine Anwendung in Python unter Nutzung von scikit-learn, Keras und TensorFlow entwickelt. Diese konzentriert sich auf die Datenaufbereitung und das Training der Modelle. Ein wichtiger Schritt bei der Datenaufbereitung ist z. B. die Auswahl der vorhersagbaren Reparaturcodes, da einige Schäden nicht anhand der Fehlercodes aus den Steuergeräten vorhergesagt werden können. In dieser Arbeit wurde diese Auswahl mit Hilfe von ANNs getroffen. Dabei konnten die Reparaturcodes der Mercedes-Benz A-Klasse von ca. 16000 auf knapp 1700 reduziert werden. Zur Optimierung der Reparaturvorhersagen von BDD wurden in dieser Arbeit verschiedene Netzwerkarchitekturen getestet, die auf grundlegenden Ideen zur Verbesserung der aktuellen Lösungsvarianten basieren. Welche der Netzwerkarchitekturen am erfolgversprechendsten ist,

wurde bei einer finalen Evaluation in Bezug auf die Daten der Mercedes-Benz A-Klasse überprüft. Zuvor wurde eine systematische Hyperparameter Optimierung für die verschiedenen Netzwerkarchitekturen durchgeführt. Genauer gesagt wurde eine teilautomatisierte Grid Search im Zusammenspiel mit einer 3-fachen Kreuzvalidierung verwendet. Die einzelnen Hyperparameter wurden dabei immer ausgehend vom besten Modell einzeln angepasst und immer wieder aufs Neue durchprobiert. Auf diese Weise konnte ein extremer Rechenaufwand bei der Grid Search vermieden werden. Die Auswahl der besten Modelle wurde mit Hilfe von dreidimensionalen Streudiagrammen getroffen.

Die Ergebnisse der finalen Evaluation zeigen, dass durch den Einsatz von ANNs eine deutliche Leistungssteigerung von BDD möglich ist. Im Vergleich zur aktuellen Lösungsvariante ist dies insbesondere bei den Metriken Attempt Rate und Classes Largely Correct zu sehen. Die besten Resultate lieferten NA1 und NA3. Bei diesen Netzwerkarchitekturen wurden die drei Oberklassen von Reparaturcodes getrennt trainiert. Bei NA3 wurden darüber hinaus noch die Daten von mehreren Fahrzeugbaureihen (A-, CLA- und GLA-Klasse) zusammen trainiert. NA3 ist die erfolgversprechendste Netzwerkarchitektur, weil die Ergebnisse nochmal leicht besser als bei NA1 sind und weil der Ansatz noch mit den Daten von anderen Fahrzeugbaureihen erweitert werden kann.

Letztlich zeigen die Ergebnisse dieser Arbeit, dass der Einsatz von künstlichen neuronalen Netzen eine echte Alternative zum Produktivsystem von BDD darstellen könnte. Um dies genauer zu überprüfen, gilt es im nächsten Schritt die Evaluation für alle Fahrzeugbaureihen durchzuführen und mit den Ergebnissen der aktuellen Lösungsvariante zu vergleichen. Bei Daimler wird im Zusammenhang mit BDD aktuell auch an einer Fahrzeugdiagnose gearbeitet, die eine vorausschauende Wartung (Predictive Maintenance) für die Daimler-Fahrzeuge ermöglicht. Im Hinblick auf die Ergebnisse dieser Arbeit könnte es sich anbieten, den Einsatz von ANNs für die Predictive Maintenance Version von BDD zu testen.

Literaturverzeichnis

- [AAB⁺16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [ARAA⁺16] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 472:473, 2016.
- [Ben12] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.
- [Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [BLSB04] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004.
- [C⁺18] François Chollet et al. Keras. <https://keras.io>, Abgerufen am 10.06.2018.
- [CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international*

- conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.
- [Cho18] Francois Chollet. *Deep learning with Python*. Manning Publications, 2018.
- [CK01] Amanda Clare and Ross D King. Knowledge discovery in multi-label phenotype data. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 42–53. Springer, 2001.
- [Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [EW02] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687, 2002.
- [Für02] Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2(Mar):721–747, 2002.
- [GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.
- [GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [Gér17] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Ö'Reilly Media, Inc.", 2017.
- [HSS17] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

- [IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.
- [KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [KTV08] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD*, volume 18, 2008.
- [LYNB08] Lucian Vlad Lita, Shipeng Yu, Stefan Niculescu, and Jinbo Bi. Large scale diagnostic code classification for medical patient records. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*, 2008.
- [Mit97] Thomas M. Mitchell. *Machine learning*. WCB. McGraw-Hill, 1997.
- [MKGD12] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern recognition*, 45(9):3084–3104, 2012.
- [ML07] Zdravko Markov and Daniel T Larose. *Data mining the Web: uncovering patterns in Web content, structure, and usage*. John Wiley & Sons, 2007.
- [MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

- [PVG⁺11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- [RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.
- [Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [RPHF09] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 254–269. Springer, 2009.
- [SHK⁺14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [SRYD14] Chong Sun, Narasimhan Rampalli, Frank Yang, and AnHai Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *Proceedings of the VLDB Endowment*, 7(13):1529–1540, 2014.
- [SSS⁺17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.
- [TK06] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3), 2006.

- [TRN09] Lei Tang, Suju Rajan, and Vijay K Narayanan. Large scale multi-label classification via metalabeler. In *Proceedings of the 18th international conference on World wide web*, pages 211–220. ACM, 2009.
- [TS10] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.
- [Tur09] Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.
- [TV07] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *European conference on machine learning*, pages 406–417. Springer, 2007.
- [XWCL15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.
- [ZZ06] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.
- [ZZ07] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.
- [ZZ14] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837, 2014.

Abbildungsverzeichnis

1.1	Maschinelles Lernen im Vergleich zur klassischen Programmierung [Cho18]	2
1.2	Einordnung von Deep Learning, Maschinellern Lernen und Künstlicher Intelligenz [Cho18]	4
1.3	Darstellung der BDD Prognosen im Diagnoseprogramm XENTRY	6
1.4	Die vier Schritte der BDD-Pipeline	7
2.1	Auswirkungen der Modell Komplexität auf das Bias-Varianz Dilemma [GBCB16]	16
2.2	Initialer BDD-Datensatz nach dem Abzug aus AQUA	18
2.3	Gruppierte Häufigkeiten der Fehlercodes	21
2.4	Gruppierte Häufigkeiten der Reparaturcodes	21
2.5	Analyse der Werkstattbesuche mit Fokus auf die Fehlercodes . .	22
2.6	Analyse der Werkstattbesuche mit Fokus auf die Reparaturcodes	22
2.7	Analyse der Werkstattbesuche mit Fokus auf die ASRA-Codes .	23
2.8	Analyse der Werkstattbesuche mit Fokus auf die PARTS-Codes	23
2.9	Analyse der Werkstattbesuche mit Fokus auf die SSL7-Codes . .	24
2.10	Zusammenfassung der BDD-Problemstellung	25
3.1	Einordnung von Multi-Label Metriken [MKGD12]	28
3.2	Multi-Label Lösungsvarianten [MKGD12]	34
3.3	ANN mit Wahrscheinlichkeiten als Outputs	39
4.1	Aufbau eines Perzeptrons [Mit97]	41
4.2	ReLU Funktion [GBCB16]	43
4.3	Sigmoid-Funktion [GBCB16]	43

Abbildungsverzeichnis

4.4	Skizzierung des Backpropagation Algorithmus [Cho18]	45
4.5	Problem von lokalen Minima beim Gradientenverfahren [GBCB16]	48
4.6	Unterschiedliche Konvergenz des Batch-, Mini-Batch und stochastischen Gradientenverfahrens [Gér17]	49
4.7	Verlauf des Trainings- und Validierungsfehlers in Bezug auf die Anzahl an Iterationen [Mit97]	50
4.8	Konzept der 3-fachen Kreuzvalidierung [Cho18]	52
4.9	Anwendung einer Dropout-Rate von 0.5 bei einer Aktivierungsmatrix [Cho18]	53
4.10	Varianten der ReLU Aktivierungsfunktion [XWCL15]	55
5.1	Grundlegendes ANN-Design im Hinblick auf die BDD-Problemstellung	60
5.2	Aktuelle Lösungsvariante von BDD mit XGBoost und OvR	63
5.3	Ziele des Transfer Lernens [TS10]	67
5.4	Transfer Lernen beim Training des SSL7-Modells	68
5.5	Lernkurve des besten ASRA-Modells	74
5.6	Dreidimensionaler Scatter Plot zur Auswahl des besten ASRA-Modells bei NA1	75
5.7	Zweidimensionaler Scatter Plot mit Fokus auf Attempt Rate und Top-3 Precision	76
5.8	Zweidimensionaler Scatter Plot mit Fokus auf Attempt Rate und Classes Largely Correct	77

Tabellenverzeichnis

2.1	Beispieldatensatz zur Metrik Classes Largely Correct	15
2.2	Charakterisierung der BDD-Daten	19
3.1	Beispieldatensatz zum Makro- und Mikro-Durchschnitt	31
3.2	Beispieldatensatz zur den Methoden der Problemtransformation	34
3.3	Beispieldatensätze zur binären Relevanz [TK06]	35
3.4	Beispieldatensätze zur paarweisen Zerlegung	36
3.5	Beispieldatensatz zur Label Potenzmenge [TK06]	36
5.1	Vergleich zwischen dem Datensatz mit allen Baureihen und dem mit den Daten der A-Klasse	60
5.2	Verwendete Hyperparameter für die Auswahl der vorhersagba- ren Reparaturcodes	62
5.3	Trainings- und Testdatensatz zu NA1	64
5.4	Trainings- und Testdatensatz zu NA2	65
5.5	Anzahl der vorhersagbaren Reparaturcodes bei der A-, CLA- und GLA-Klasse	65
5.6	Trainings- und Testdatensatz zu NA3	66
5.7	Trainings- und Testdatensatz zu NA4	68
5.8	Beispielhafte BoW-Matrix	70
5.9	Trainings- und Testdatensatz zu NA5 und NA6 mit Fokus auf die Input Varianten	70
5.10	Getestete Werte bei der Hyperparameter Optimierung	73
5.11	Optimierte Hyperparameter für die finale Evaluation	77
5.12	Ergebnisse der finalen Evaluation	80

A Eidesstattliche Erklärung

Ich versichere, dass die Masterarbeit mit dem Titel „Verbesserung der automatischen Fahrzeugdiagnose mit Maschinellen Lernverfahren“ nicht anderweitig als Prüfungsleistung verwendet wurde und diese Masterarbeit noch nicht veröffentlicht worden ist. Die hier vorgelegte Masterarbeit habe ich selbstständig und ohne fremde Hilfe abgefasst. Ich habe keine anderen Quellen und Hilfsmittel als die angegebenen benutzt. Diesen Werken wörtlich oder sinngemäss entnommene Stellen habe ich als solche gekennzeichnet.

Augsburg, den 15. Juni 2018

Unterschrift